A Major Project-II Report

On

NLP to SQL

Translating Natural Language to Database Query Language

Submitted in Partial fulfilment of the Requirement for the Degree of

Master of Technology

in

Computer Science and Engineering

Submitted By

Amit Chaudhary

2K17/CSE/03

Under the Guidance of

Ms. Minni Jain

Assistant Professor

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Shahabad Daulatpur, Main Bawana Road, Delhi-110042

June 2019

CERTIFICATE

This is to certify that Project Report entitled" (**NLP to SQL) Translating Natural Language to Database Query Language"** submitted by **Amit Chaudhary** (2K17/CSE/03) in partial fulfilment of the requirement for the award of degree Master of Technology (Computer Science and Engineering) is a record of the original work carried out by him under my supervision.

Project Guide
Ms. Minni Jain
Assistant Professor
Department of Computer Science & Engineering
Delhi Technological University

DECLARATION

I hereby declare that the Major Project-II work entitled" **(NLP to SQL) Translating Natural Language to Database Query Language"** which is being submitted to Delhi Technological University, in partial fulfilment of requirements for the award of the degree of Master of Technology (Computer Science and Engineering) is a bona fide report of Major Project-II carried out by me. I have not submitted the matter embodied in this dissertation for the award of any other degree or diploma.

Amit Chaudhary
2K17/CSE/03
M. Tech (Computer Science & Engineering)
Delhi Technological University

ACKNOWLEDGEMENT

First of all, I would like to express my deep sense of respect and gratitude to my project supervisor Ms. Minni Jain for providing the opportunity of carrying out this project and being the guiding force behind this work. I am deeply indebted to him for the support, advice and encouragement he provided without which the project could not have been a success.

Secondly, I am grateful to Dr. Rajni Jindal, HOD, Computer Science & Engineering Department, DTU for her immense support. I would also like to acknowledge Delhi Technological University library and staff for providing the right academic resources and environment for this work to be carried out.

Last but not the least I would like to express sincere gratitude to my parents and friends for constantly encouraging me during the completion of work.

Amit Chaudhary

Roll No – 2K17/CSE/03

M. Tech (Computer Science & Engineering)

Delhi Technological University

ABSTRACT

Using Query language to fetch and visualize the data in specified format requires knowledge and expertise about domain and technologies in which data are being stored in databases. The gap between what information user wants which is going to satisfies the need at the desired moment and what information is being fetched requires clear communication between database expert and user requesting the data.

In this project I understudy the different methodologies which was proposed in the domain of natural language to query language such as **Seq2SQL, Neural Enquirer, ONLI (Ontology-based Natural Language Interface), NaLIR (Natural Language Interface for Relational Database), SQLNet and ln2sql** put the comparison and features of databases which are being used to train the model. Training Datasets are WikiSQL, Spider, DBpedia, Geoquery, ATIS, Overnight, WebQuestions, Freebase917 and WikiTableQuestions.

The main challenges of data retrieval are the heterogenous form of data storage which requires complex and different domain specific queries. In extension of this work, I propose how complex queries can be fetched from multiple normalized database tables.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

S. No	**Abbreviation**	
1	NLP	Natural Language Processing
2	SQL	Structured query language
3	RNN	Recurrent neural network
4	SEQ2SQL	Sequel to Structured query language
5	Ln2SQL	Natural language to SQL
6	ONLI	Ontology-based Natural Language Interface
7	NaLIR	Natural Language Interface for Relational Database
8	SQLNet	SQL Network
9	AITS	Administrative information technology services
10	DBpedia	Database pedia

Chapter 1: Introduction

Due to rapid computing scenario, Information retrieval methods are being highly accustomed to help organizations, education institutions, on demand information which is required while communicating with users and client meeting to present them with accurate data. Despite large amount of data can be fetched efficiently and accurately from Relational Databases whoever it requires personnel to master the semantics to formulate the formal queries. Linguistics (Natural Language processing) and Artificial Intelligence capabilities can form an association to understand information in natural language and generate database query language such as NoSQL and fetch information from selected database.

Methodologies which helps to convert natural language to database query language are Seq2SQL, Neural Enquirer, ONLI (Ontology-based Natural Language Interface) and these models depending upon different datasets to train their model. Datasets such as WikiSQL, Spider, DBpedia, Geoquery, ATIS, Overnight, WebQuestions, Freebase917, WikiTableQuestions.

General solution architecture NLP to SQL is to take the text/ (speech to text) is given as input to the model lexical analyser. Lexical analysis performs the pre-processing of input text by help of tokenization, lemmatization/stemming and POS tagging and pass the words for semantic analysis. After processing of semantic analyser relational mapping to database table names, column names and condition clauses are fetched so that query generator can generate the query.

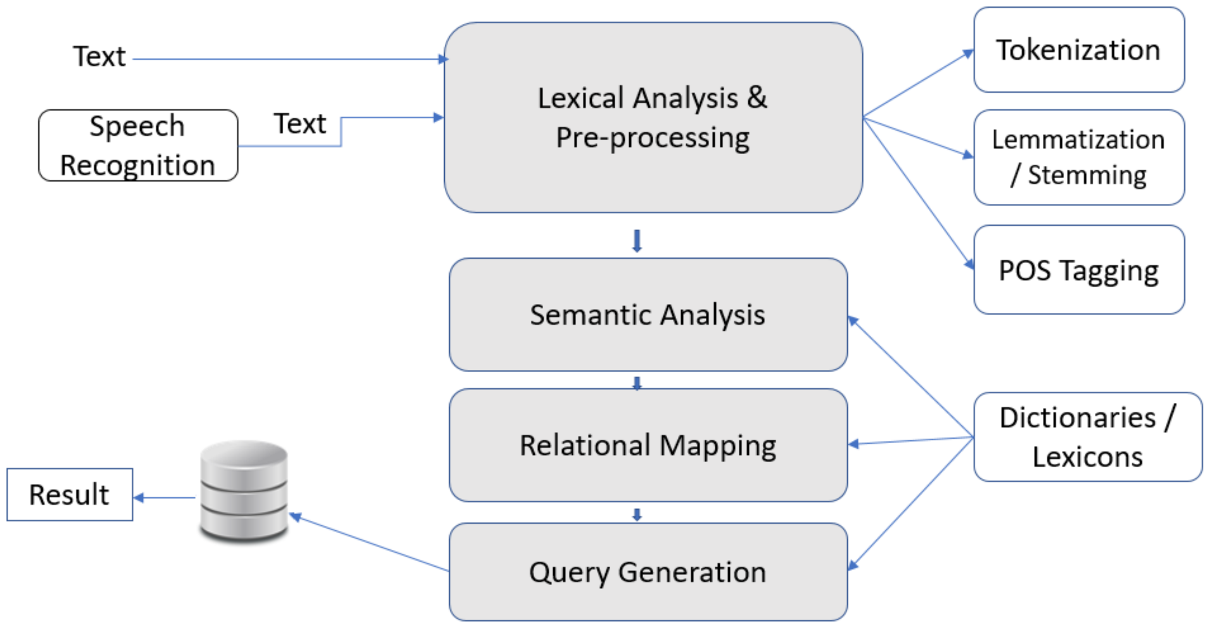


Figure 1. Simple Architecture of NLP to SQL

The Architecture shown in figure one can be understood in simple steps with help of lexical analysis, semantic analysis, relational mapping and query generation.

Lexical analysis & pre-processing collects words in a sentence, it involves tokenization, lemmatization, stemming, POS tagging.

Tokenization means breaking the text to chunks of individual words. These tokens used to find relevant information from the sentence.

For Example

Sentence: - find the unique names of all employee with age 24.

Tokens: - {"find"," unique"," names"," all"," employee"," with"," age"," 24"}.

Stemming means cutting of the suffix or prefix of words.

For Example

Form	**Stem**
Eating	eat
studying	study
walking	walk
morning	morn

Lemmatization means to take care of inflectional variance, considers meaning of words dictionaries are needed to process, the morphological linguistics makes it more complicated.

The word good is lemma for better, which got missed by stemmer.

Word	**Stemming**	**Lemmatization**
Beautiful, beautifully	beauti	beautiful

Here stemming produces beauti which have no meaning, lemmatization produces correct form of word which is beautiful.

Both stemming, and lemmatization is part of normalization of word to bring it to the root word

Stemming is easy to design, but it also loses the meaning sometimes and the context in which word is being said.

1.1 Related work

Many researchers have focusing on natural language to SQL conversion, which is very useful for consumers to efficiently produce and populate information in real time.

Below are some of NlP to SQL translators developed by many researchers such as Seq2SQL, Neural Enquirer, ONLI (Ontology-based Natural Language Interface), NaLIR (Natural Language Interface for Relational Database), SQLNet and ln2sql put the comparison and features of databases which are being used to train the model. Training Datasets are WikiSQL, Spider, DBpedia, Geoquery, ATIS, Overnight, WebQuestions, Freebase917 and WikiTableQuestions.

There are other translators based on machine learning, knowledge base, parsing the tokens of natural language and creating a logical dependency graph all of them have their unique key feature and problem area which were addressed and tried to be optimized.

The detailed comparisons of translators and dataset are explained in chapter 2 and chapter 3. There are many researchers focused in optimizing and trying to make the efficient system which is reliable and can be used at real-time by the consumers which benefits their business scenario's.

Below are some of NlP to SQL translators developed by many researchers.

1. **Seq2SQL**

   To translate questions to analogous SQL queries, Seq2sql uses on deep neural network. Reinforcement (RNN) learning to train the model with WikiSQL.

2. **Neural Enquirer**

   Termed as completely neutralized, it looks for distributed representations of knowledge base tables and queries. Executors are used to generate queries in series at different levels. Intermediate results are in form of table annotations at multiple layers.

3. **ONLI (Ontology-based Natural Language Interface)**

   Uses ontology model to represent the Question context and syntactic question composition. Model used permits hypothesize expected answer type by the user with help of classification of questions.

4. NaLIR (Natural Language Interface for Relational Database)

To generate parse tree from natural language NaLIR uses off-the-shelf NL parser. Interactive Communicator is used to make the translation transparent and while translating getting feedback from the user if it is following the desired path to get the efficient and accurate result.

5. SQLNet

To generate query, it uses sketch-based approach. Sketch represents the syntactical representation of SQL query. Major issue addressed by this approach is to optimize generating where clause.

6. Ln2SQL

Ln2SQL is generated for French language, it's a very simple model to parse the sentence for each section of query generation such as column name extraction, where clause, table name, order by and group by. It's based on parsing the natural language using different libraries. UI is created to interact with the user to get SQL query generated.

Detailed Architecture is explained in Chapter 3 NLP to SQL. There are other translators based on machine learning, knowledge base, parsing the tokens of natural language and creating a logical dependency graph all of them have their unique key feature and problem area which were addressed and tried to be optimized. In past few years accuracy and efficiency has been increased from 21% to 83%. In upcoming years, the results will be accurate and efficient as the researchers are focusing their efforts into it.

Chapter 2: NLP to SQL

Many researchers have focusing on natural language to SQL conversion, which is very useful for consumers to efficiently produce and populate information in real time.
Below are some of NlP to SQL translators developed by many researchers.

1. Seq2SQL

To translate questions to analogous SQL queries, Seq2sql uses on deep neural network. Training of model is performed using WikiSQL dataset, which have 87726 annotated set of questions and queries. To induce unordered parts of query rewards are used in execution of loop query over the database. Reinforcement learning is used to train with WikiSQL as execution environment.

Seq2SQL is composed of 3 parts:

1. **Aggregation Classifier** which produces the summary of column/ row selected by the Query.
2. **SELECT column pointer** which finds the column names which needs to be part of output result.
3. **Where clause decoder pointer** which finds the filtering conditions to get the desired outcome.

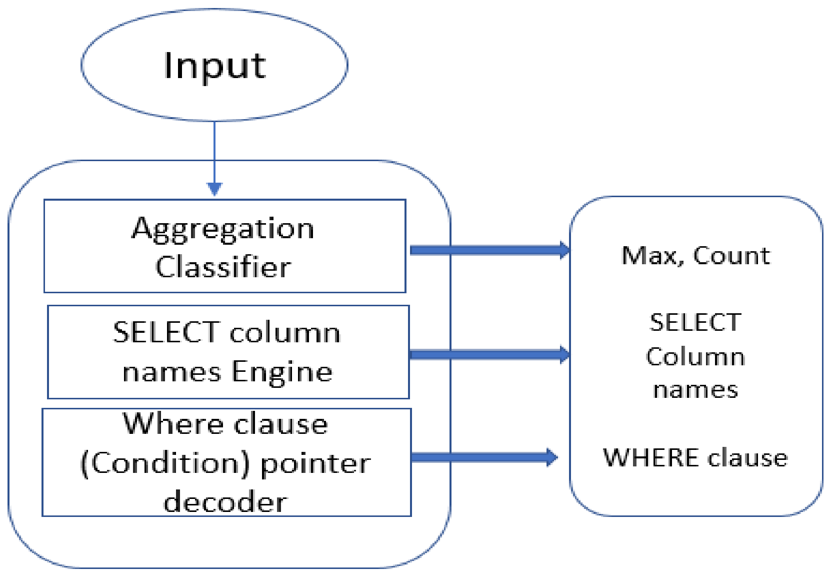


Fig 2 Seq2SQL Architecture

2. Neural Enquirer

Termed as completely neutralized, it looks for distributed representations of knowledge base tables and queries. Executors are used to generate queries in series at different levels. Intermediate results are in form of table annotations at multiple layers.

Neural Enquirer composed of Query Encoder speculate semantics of the input text, Table Encoder drives embedding tables by encoding entries in table in form of distributed vectors and Executor which computes with help of table and annotations. In each layer's annotations are stored which is used by next layer executor.

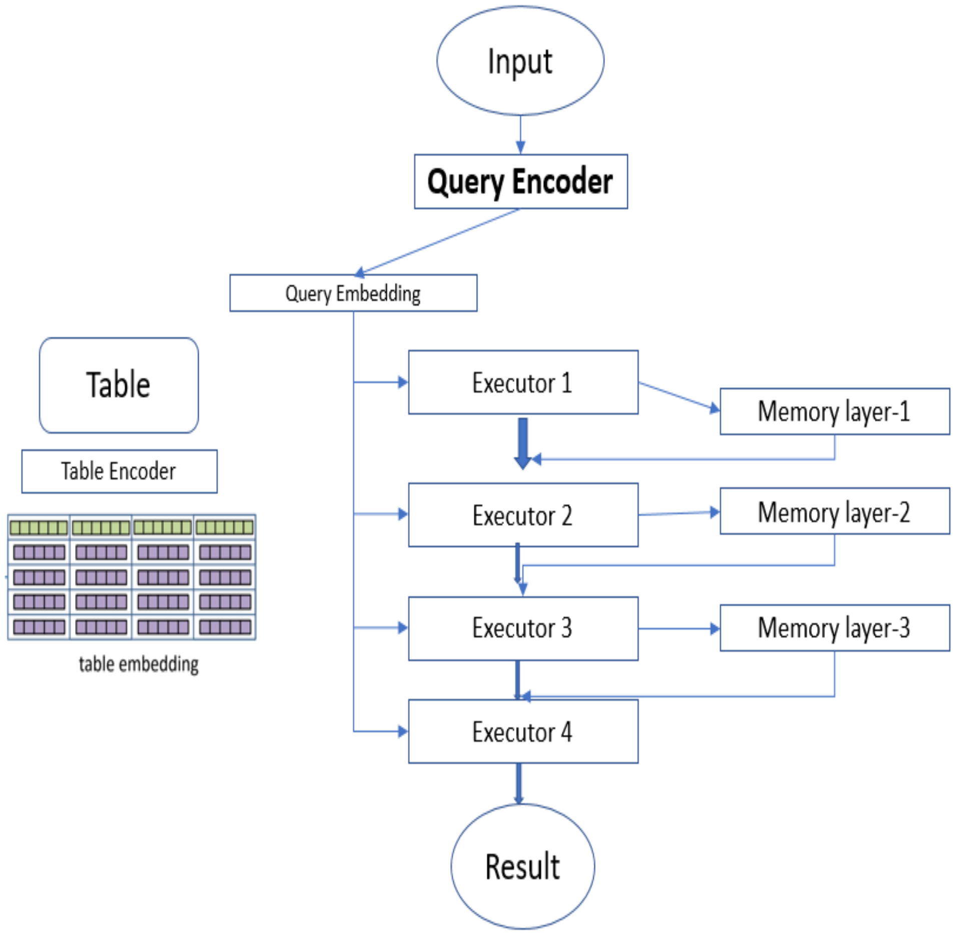


Figure 3 Neural Enquirer with four Executors

3. ONLI (Ontology-based Natural Language Interface)

Uses ontology model to represent the Question context and syntactic question composition. Model used permits hypothesize expected answer type by the user with help of classification of questions.

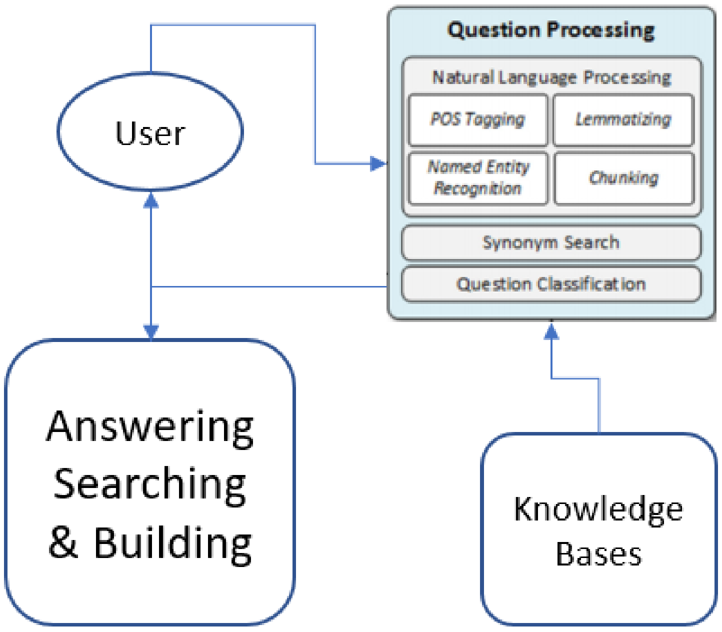


Figure 4 ONLI system Architecture

4. NaLIR (Natural Language Interface for Relational Database)

For querying relational databases, it can accept complex English language and generates variety of SQL query which is composition of aggregation joins nesting of sub query with other multiple where clause conditions with multiple application domains NaLIR can be used to generate complex queries.

To generate parse tree from natural language NaLIR uses off-the-shelf NL parser.

Interactive Communicator is used to make the translation transparent and while translating getting feedback from the user if it is following the desired path to get the efficient and accurate result.

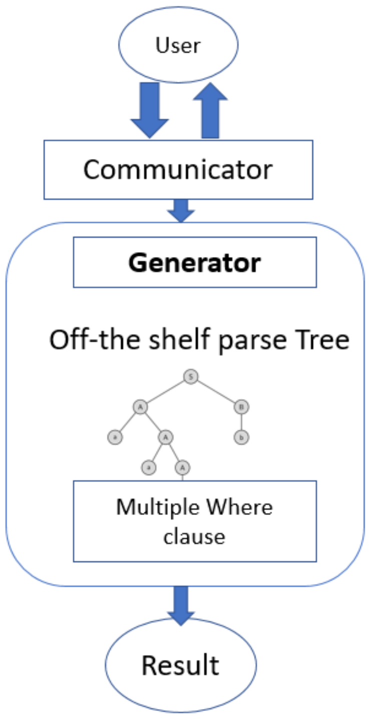


Figure 5. Architecture of NaLIR

5. SQLNet

To generate query, it uses sketch-based approach. Sketch represents the syntactical representation of SQL query. Major issue addressed by this approach is to optimize generating where clause. To do that it employs the sketch which provides dependency relationship among different condition slots, so it helps in determining which condition slots are depends on each other and which are independent. To train the model it uses WikiSQL dataset. When order is not required it avoids sequence_to_sequence structure.

SQLNet introduced column attention and sequence-to-set. First is to capture relationship specified in the sketch, second is to determine unordered set of conditions slots rather of ordered condition slots.

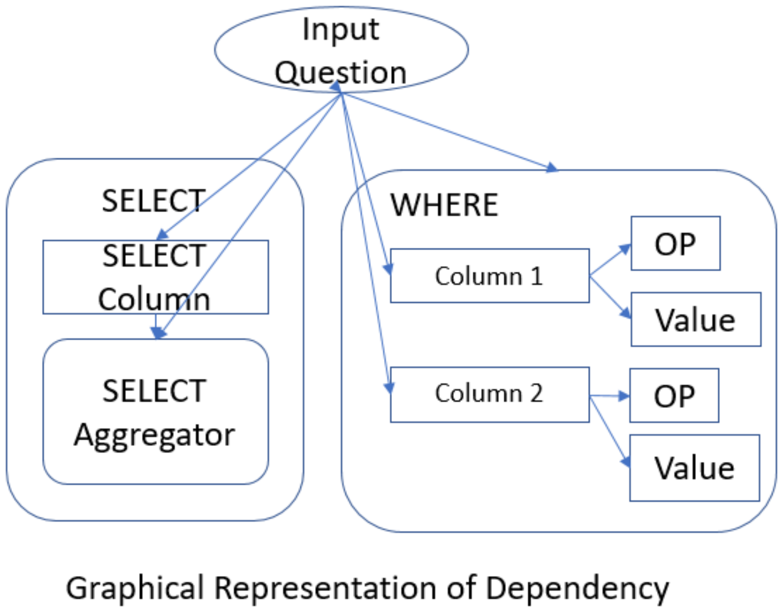


Figure 6 SQLNet Graphical representation of processing

6. Ln2SQL

Ln2SQL is generated for French language, it’s a very simple model to parse the sentence for each section of query generation such as column name extraction, where clause, table name, order by and group by. Its based on parsing the natural language using different libraries. UI is created to interact with the user to get sql query generated.

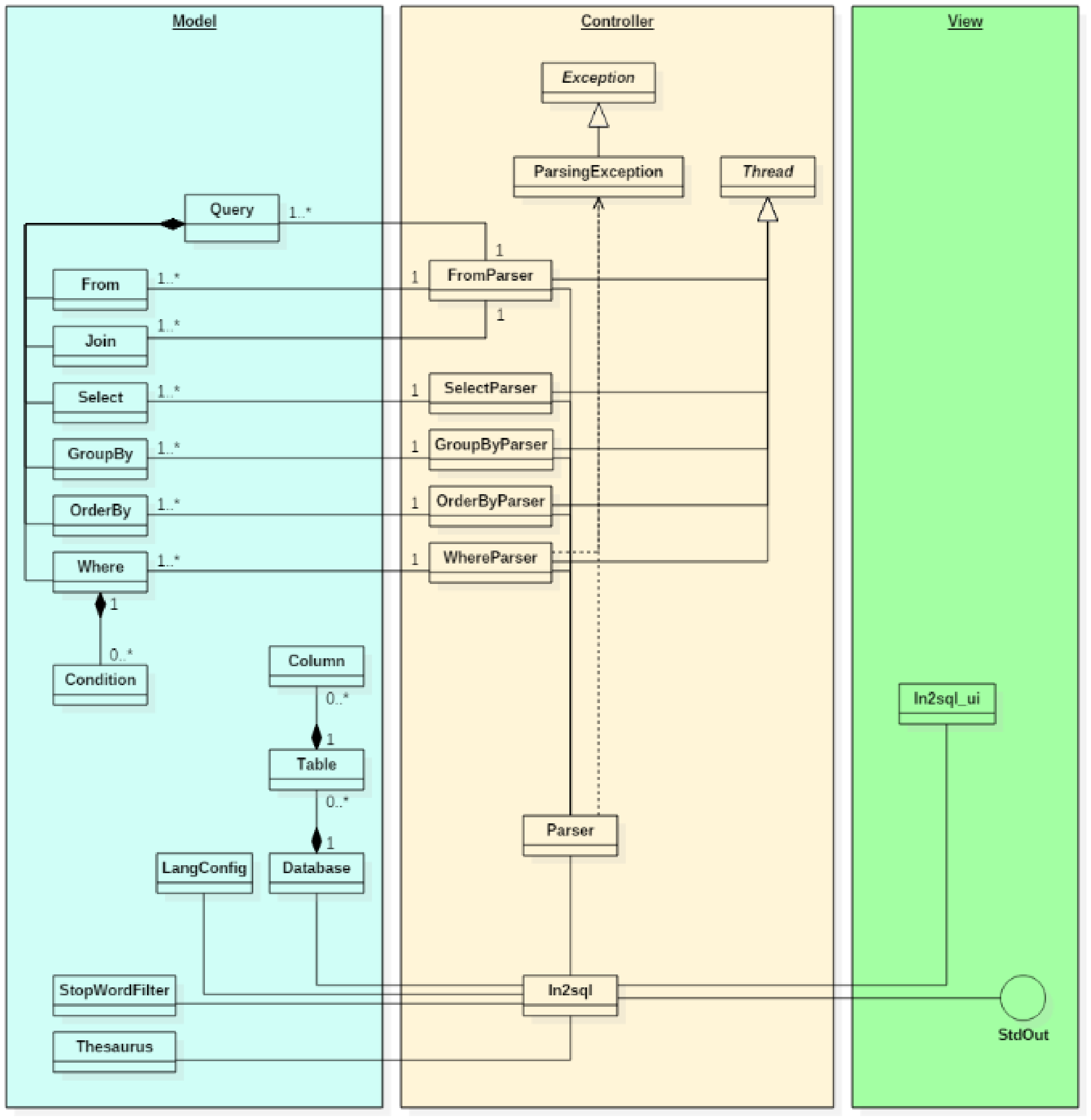


Figure 7 Ln2SQL Architecture

Chapter 3: Comparison Tables

Comparison of Translators

Translators such as Seq2sql, Neural Enquirer, ONLI, NaLIR, SQLNet are compared based on domains, type of datasets used to produce the result, some of them are machine learning based and some are sequence to sequence model based. Both methodologies have own complexity and accuracy results, few translators are tested over standard dataset which is available to compare and analysis the performance of translators. In table 1 comparison of translators is shown with attributes such as year of release, domains covered, dataset used, methods used to generate the query and accuracy on standard dataset.

ONLI have better accuracy to generate the structured query with 83%.

Table 1. Comparison of Translators

Translators Features	Seq2SQL	Neural Enquirer	ONLI	NaLIR	SQLNet	Ln2SQL
Year of release	2017	2017	2015	2018	2017	2015
Domains covered	Single	Single	Multiple	Multiple	Single	Database dependent
Dataset used	WikiSQL	Simulated table data + Real table data	DBpedia	NA	WikiSQL	Thesaurus
Method Used	Policy Based Reinforcement learning	RNN Neural Network	Ontology model	Parser and interactive communicator	Sketch-based (dependency graph)	Parser-based
Accuracy on standard dataset	60.8%	59%	83%	67%	69.8%	--

Comparison of Datasets

Multiple datasets are used by these translators and new dataset is being formulated to tackle the issues and cover more domains and different type of SQL queries. In table 2 comparison of different datasets are shown with attributes such as year of release, Feature of that dataset that makes it different from other dataset, Size and schema of the dataset. And does it have annotated logical forms.

Spider dataset covers more area with multiple variances of SQL query coverage as shown in figure

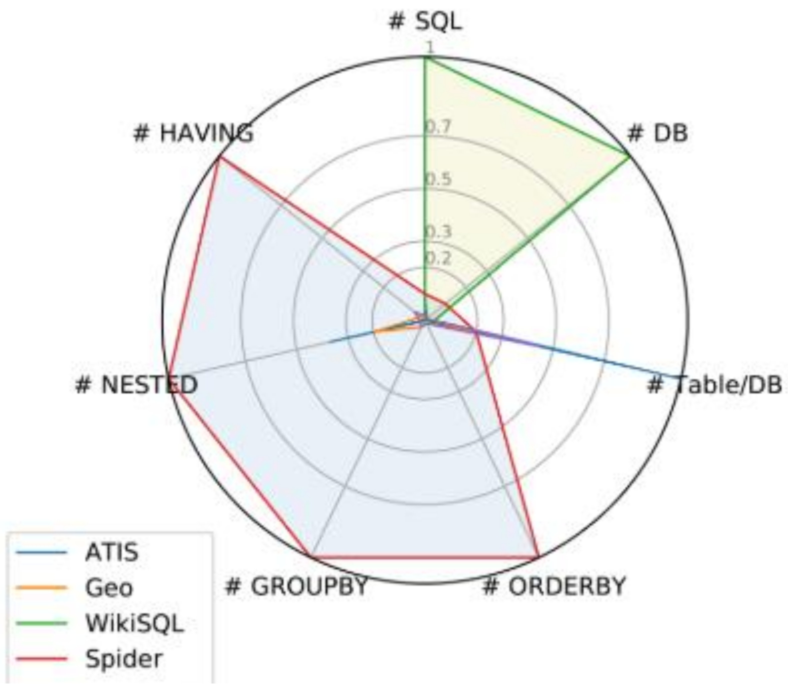


Fig 8. Coverage of Spider Dataset.

Table 2. Comparison of different dataset

Dataset	Year of release	Feature	Size	Schema	Annotated logical form
WikiSQL	2017	Single table base	87,726	26375	True
Spider	2018	Cross- domain text	10,181	200	True
DBpedia	2016	Based on Ontology	13 B	50+	True
Geoquery	2001	Sentences with their logical form	880	8	True
AITS	1990	Proposed as Slot filling task.	5871	141	True*
Freebase917	2015	Each API page is counted as different domain	917	81	True
WebQuestions	2013	Unique Id and annotations for each question x	22,033	2420	False
Overnight	2015	Domain knowledge base with grammar	26098	8	True
WikiTableQuestions	2015	Tables with varying questions	22033	2108	False

Chapter 4: Proposed Translator

4.1 Proposed Architecture

After analysis of translators and dataset features, I analysed to apply reinforcement learning and transfer learning at classification layer to improve the efficiency and to train our model we propose to use spider dataset because it have cross-domain text-to-query which covers almost all varying combination of query formation with multiple conditions attributes with Observer which observe for user to visualize and communicate with the system if step taken by system needs to be retract its mistake.

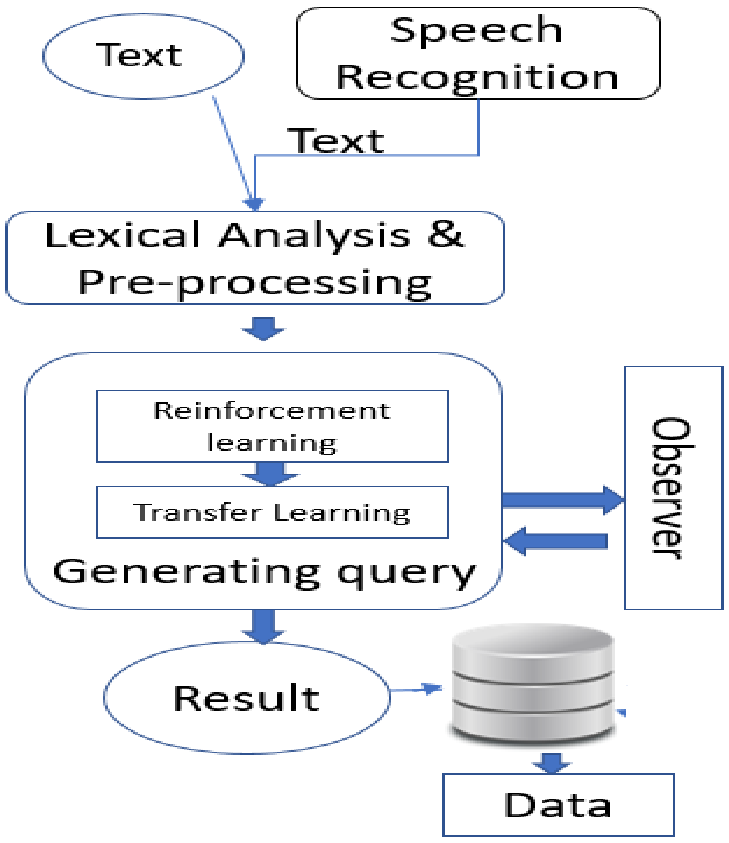


Figure 9. Architecture of proposed translator

Lexical analyser takes text as input and performs analysis and pre-processing using tokenization, lemmatization, removing stop words making input clean for processing by system so that it can be used by classifier to accurately identify cross-ponding query with the relevant tokens identified. With help of observer will user will be able to visualize each step and correct any mistakes which will help improvise the system to predict the correct query. Generated query then will be run on stored database to fetch the result whose accuracy will be compared with the expert generated query.

Query generation can be done by using spider dataset which have varying variations of SQL queries. To generate the queries, we can use reinforcement learning on top of that at final layer will be trained by transfer learning which will learn from already crated model and keep on learning as new forms are being introduced, which will increase the efficiency to train the system and reduce the time to train the model

The final generated query will be executed over database and fetch the data as per query.

The types of queries which should be generated in explained in next topic and few variations are tested and implemented in this project as they are not based on this architecture.

4.2 Variations of Queries

Types of queries which should be generated by translator are.

- SELECT operation with one column multiple columns and all columns.
- SELECT distinct on columns.
- Aggregate functions such as count-select, sum-select, avg-select, min-select, max-select
- JOIN inner join natural join
- WHERE clause with one condition multiple conditions with operators such as equal operator,
- not equal operator, greater-than operator, less-than operator, like operator, between operator (not 100% efficient)
- ORDER BY with ASC, DESC
- GROUP BY multiple queries
- Handling of Synonyms of Table and column names

Chapter 5: Implementation

In this project, I implemented basic natural language to SQL query translator with a graphical user Interface through which user can enter the query in natural language and got the query in SQL form.

5.1 Tools

Graphical User Interface is implemented using Tkinker library in python.

The logical of parser and SQL translator is written in python.

The basic UI is shown below which takes Query in natural language and a generate button which executes query generation and shows the result. It also generates the output.jason file format which will be used as payload for REST web services which will be used to fetch data from the actual database table .

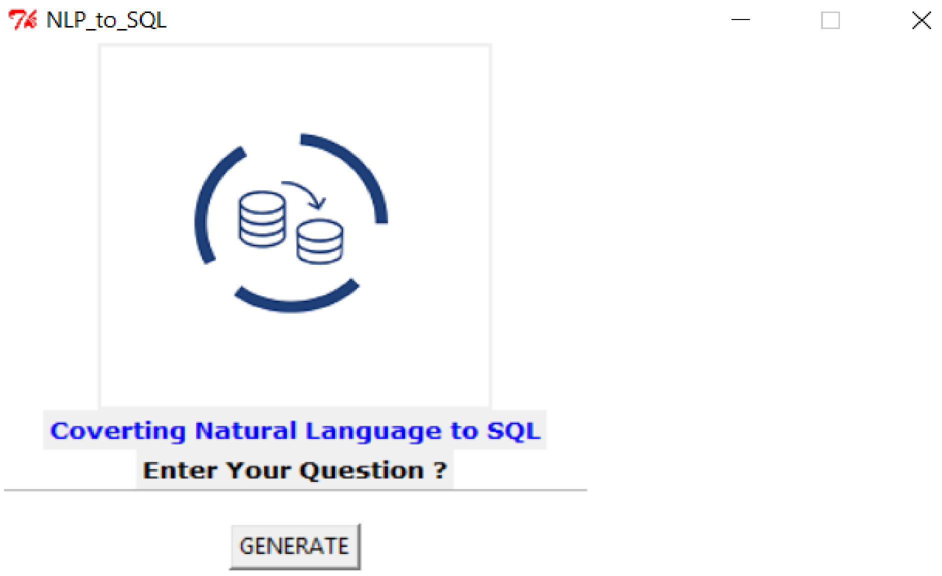


Figure 10. User interface of NLP2SQL

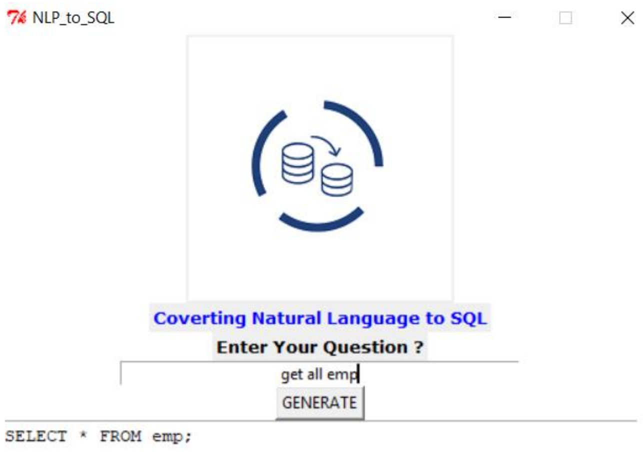
There is no need to connect to the database only SQL metadata dump file is required to get names of all the tables which it needs to fetch. A

After query is generated successfully it shown on UI in output area as shown by figure 11.

Simple query generated to get all employee data.

Natural language: - get all emp.

SQL Query Generated: - SELECT * FROM emp;



get all emp

```
SELECT * FROM emp;
```

Figure 11. SQL generated

5.2 Types of Queries

Types of queries which can be generated by NLP_to_SQL translator are.

- SELECT operation with one column multiple columns and all columns.
- SELECT distinct on columns.
- Aggregate functions such as count-select, sum-select, avg-select, min-select, max-select
- JOIN inner join natural join
- WHERE clause with one condition multiple conditions with operators such as equal operator,
- not equal operator, greater-than operator, less-than operator, like operator, between operator (not 100% efficient)
- ORDER BY with ASC, DESC
- GROUP BY multiple queries
- exception and error handling
- Detection of values

5.3 Database Tables

Tables being used to demonstrate the working of NLP_TO_SQL Conversion, the tables names are shown below

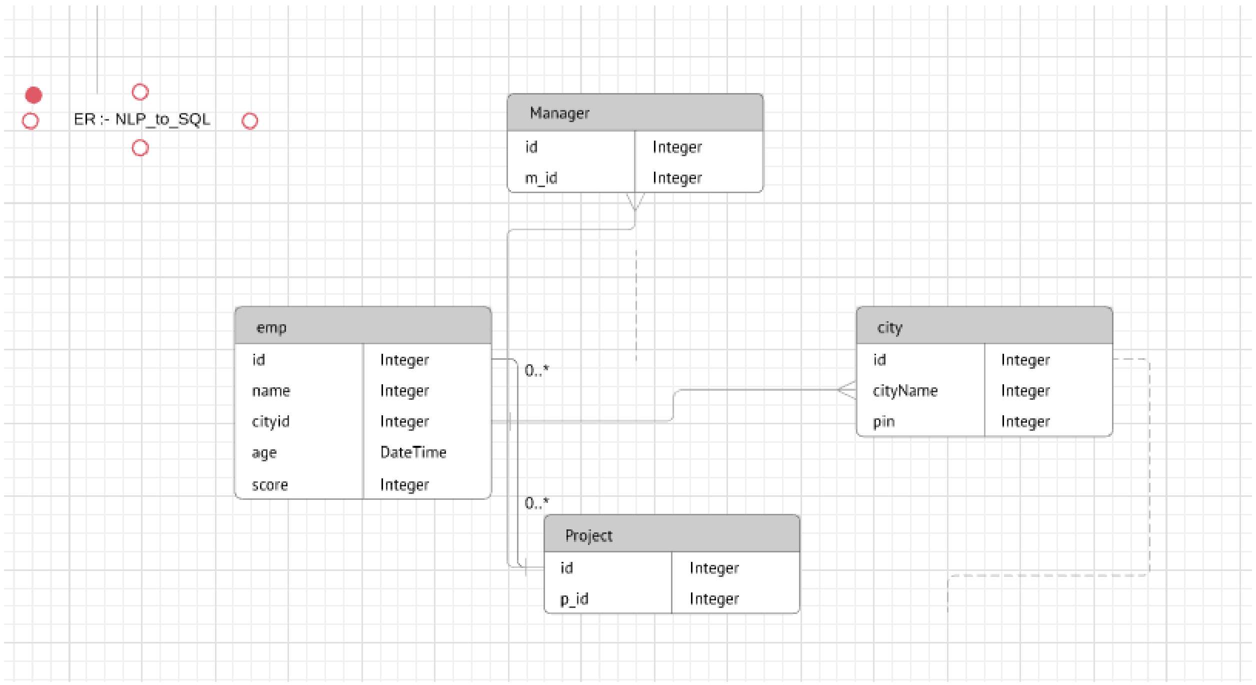


Figure.12 ER diagram of NLP_TO_SQL

In this ER diagram four tables is shown

Manager with id_id, m_id

Emp (Employee) with id, name, cityid, age, score.

Project with id, p_id

City with id, CityName, pin.

Chapter 6: Verification & Testing

6.1 Introduction to Testing

Broadly there are two testing categories Whit Box Testing and black Box Testing

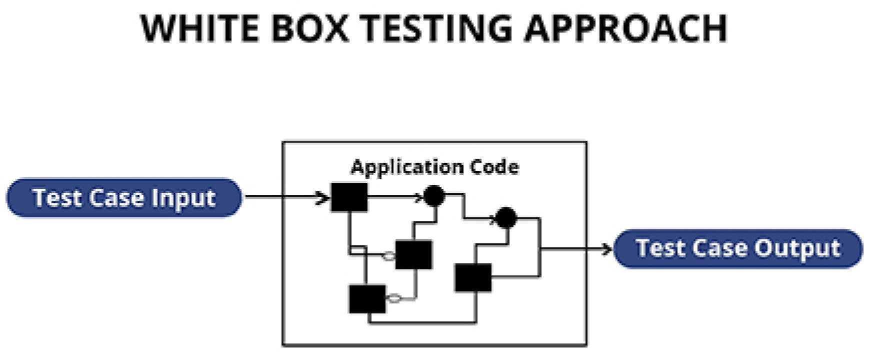


Figure 13 White box testing

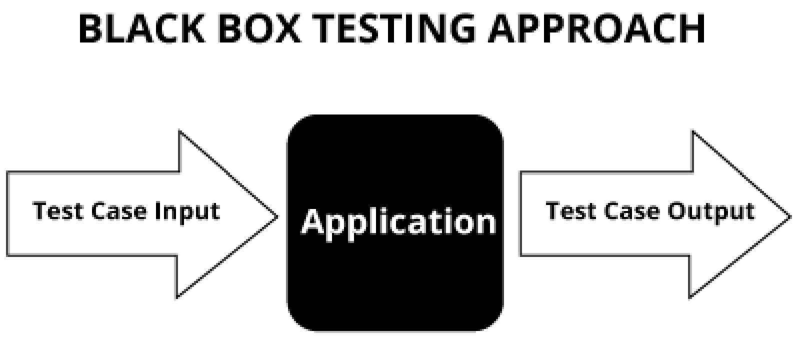


Figure 14 Black box testing

6.2 Testing NLP_to_SQL

Testing NLP_to_SQL to test which types of queries can be generated by the system, I created test case plan which covers multiple variation of SQL queries and combinations which should ideally be performed by the translator to qualify as the efficient system. Testing is done using Black box testing methodology.

6.3 Test Case Plan

To test scenarios to check what can be achieved by the system.

S. No	**Use Cases**	**Test Passed**
1	Retrieving Table	**Yes**
2	Selecting columns from the Table	**Yes**
3	Fetch data using constraints	**Yes**
4	Using Aggregation to compute &fetch data	**Yes**
5	Joining Multiple Tables	**Yes**
6	Like operator	**Yes**
7	Fetch Data in Sorted Order	**No**
8	Exception & error handling	**Yes**
9	Nested query generation	**No**
10	Synonyms matching with table name	**No**

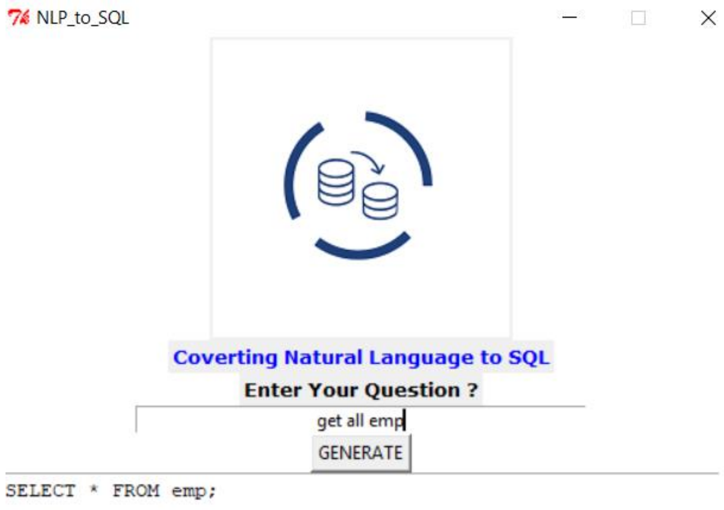
Table 3. Test case plan

6.4 Test Case Execution

1. Retrieving Table

Natural Language: - Get all emp

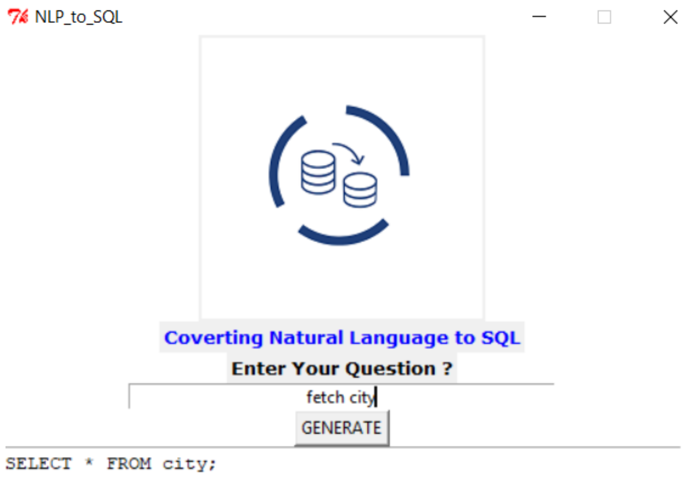
SQL Generated: - SELECT * FROM emp;



```
SELECT * FROM emp;
```

Natural Language: - Fetch city

SQL Generated: - SELECT * FROM city;

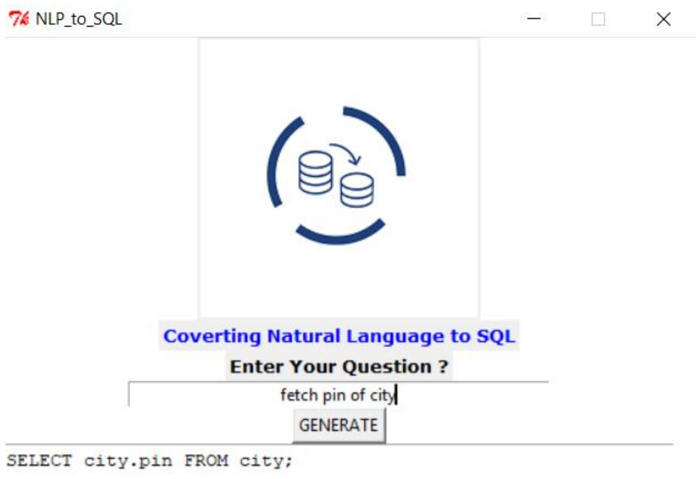


```
SELECT * FROM city;
```

2. **Selecting columns from the Table**

   Natural Language: - fetch pin of city

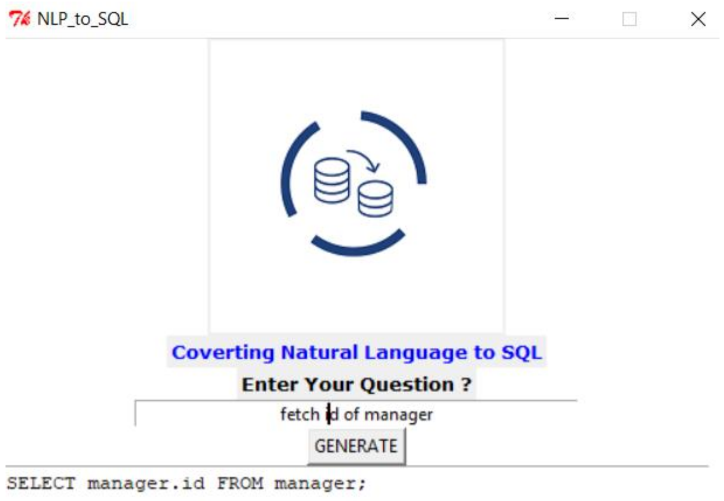
   SQL Generated: - SELECT city.pin FROM city;



```
SELECT city.pin FROM city;
```

Natural Language: - fetch id of city

SQL Generated: - SELECT manager.id FROM manager;



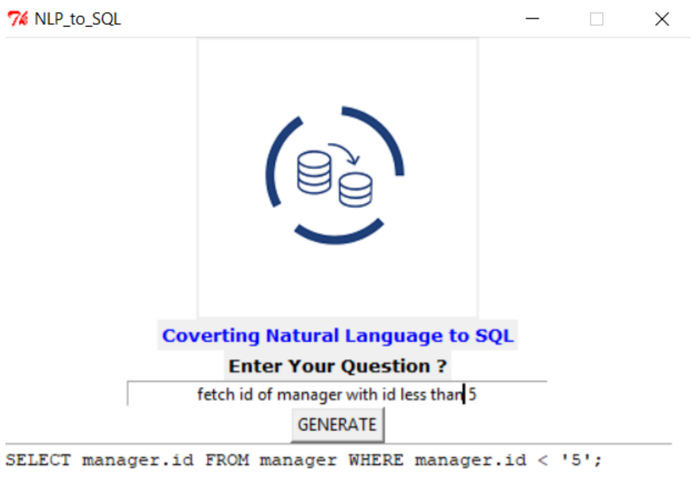
```
SELECT manager.id FROM manager;
```

3. Fetch data using constraints

Natural Language: - Fetch id of manager with id less than 5

SQL Generated: -

SELECT manager.id FROM manager WHERE manager.id < '5';



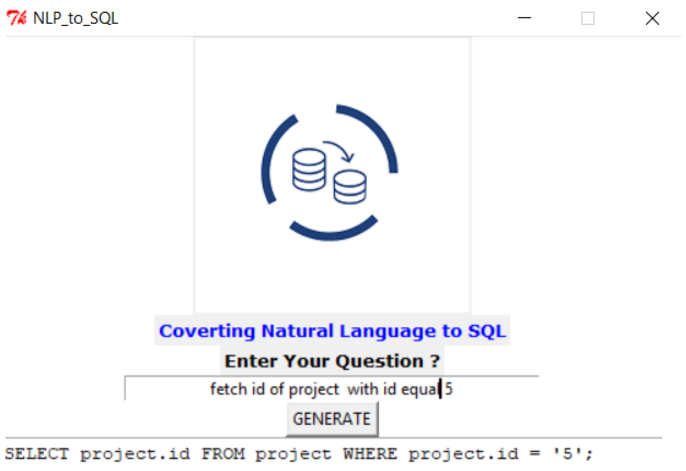
fetch id of manager with id less than 5

```
SELECT manager.id FROM manager WHERE manager.id < '5';
```

Natural Language: - Fetch id of project with id equal to 5

SQL Generated: -

SELECT manager.id FROM manager WHERE manager.id < '5';



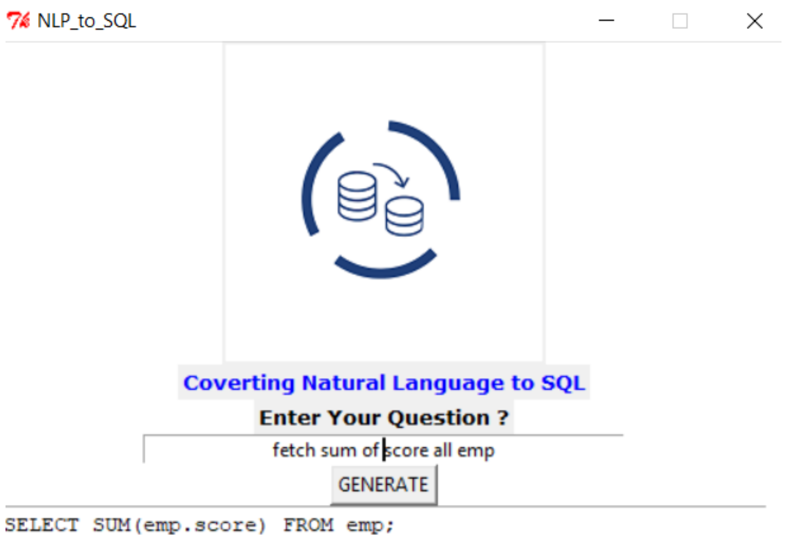
fetch id of project with id equal 5

```
SELECT project.id FROM project WHERE project.id = '5';
```

4. Using Aggregation to compute &fetch data

Natural Language: - Fetch sum of score all emp

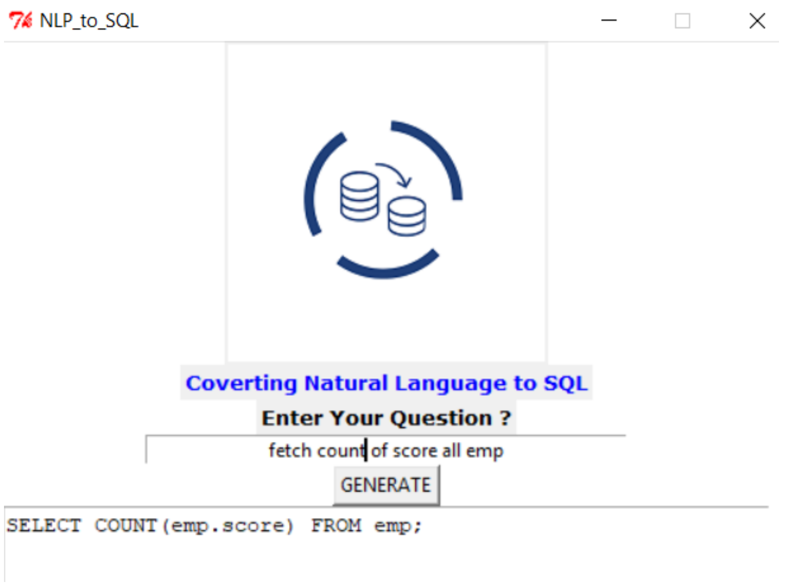
SQL Generated: - SELECT SUM (emp.score) FROM emp;



```
SELECT SUM(emp.score) FROM emp;
```

Natural Language: - Fetch Count of score all emp

SQL Generated: - SELECT COUNT(emp.score) FROM emp;

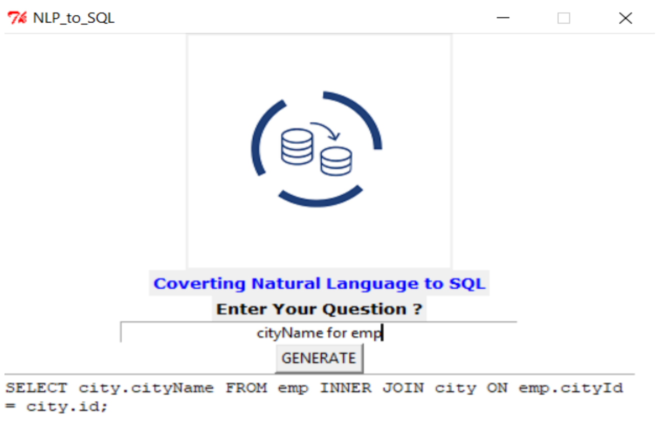

NLP_to_SQL
Coverting Natural Language to SQL
Enter Your Question ?
fetch count of score all emp
GENERATE

```
SELECT COUNT(emp.score) FROM emp;
```

5. Joining Multiple Tables

Natural Language: - CityName for emp

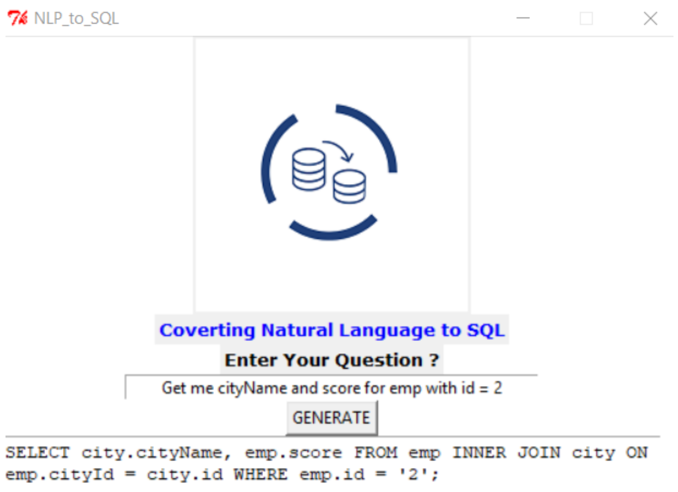
SQL Generated: - SELECT city.cityName FROM emp INNER JOIN city ON emp.cityId = city.id;



```
SELECT city.cityName FROM emp INNER JOIN city ON emp.cityId
= city.id;
```

Natural Language: - Get me cityName and score for emp with id = 2

SQL Generated: - SELECT city.cityName, emp.score FROM emp INNER JOIN city ON emp.cityId = city.id WHERE emp.id = '2'

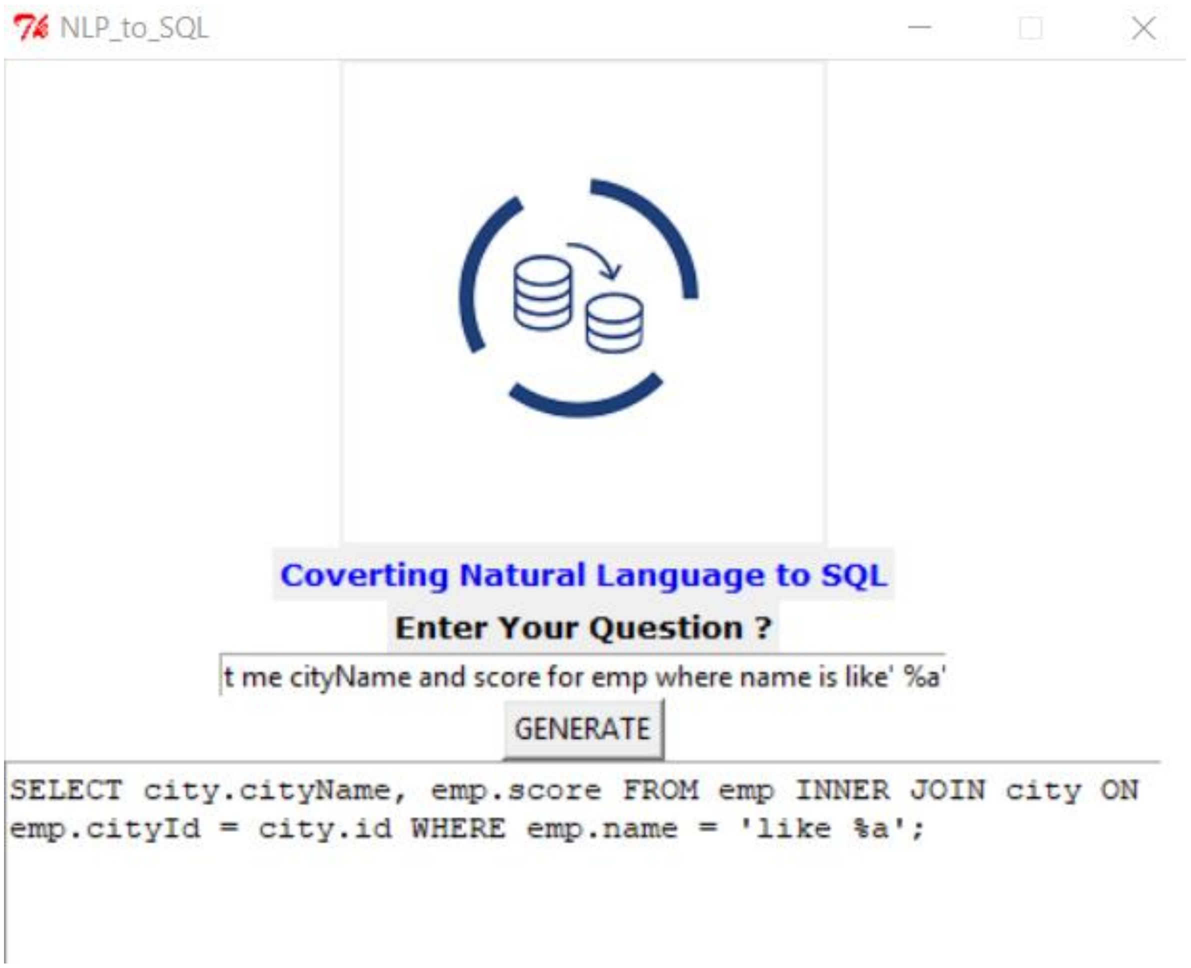

NLP_to_SQL
Coverting Natural Language to SQL
Enter Your Question ?
Get me cityName and score for emp with id = 2
GENERATE

```
SELECT city.cityName, emp.score FROM emp INNER JOIN city ON
emp.cityId = city.id WHERE emp.id = '2';
```

6. Like Operator

Natural Language: - Get me cityName and score for emp where name is like' %a'

SQL Generated: - SELECT city.cityName, emp.score FROM emp INNER JOIN city ON emp.cityId = city.id WHERE emp.name = 'like %a';



t me cityName and score for emp where name is like' %a'

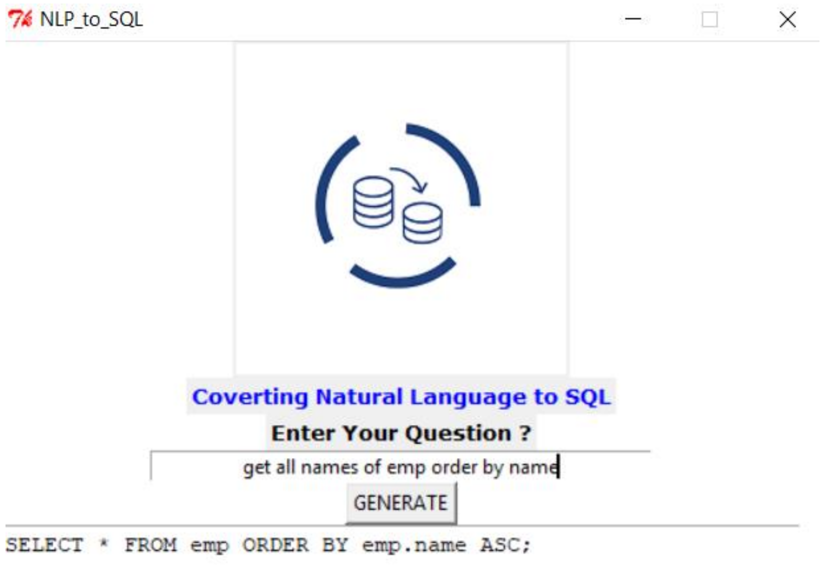
```
SELECT city.cityName, emp.score FROM emp INNER JOIN city ON
emp.cityId = city.id WHERE emp.name = 'like %a';
```

Did not get accurate result as like operator was not working properly.

7. Fetch Data in Sorted

Natural Language: - get all names of emp order by name

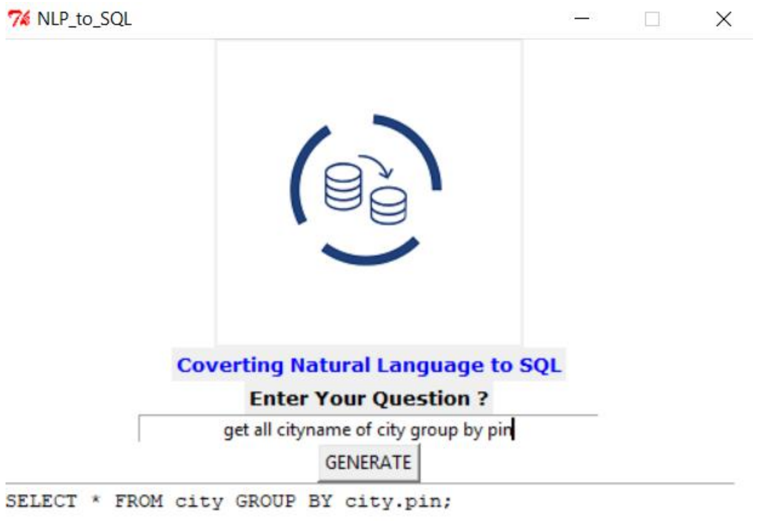
SQL generated: - SELECT * FROM emp ORDER BY emp.name ASC;


NLP_to_SQL
Coverting Natural Language to SQL
Enter Your Question ?
get all names of emp order by name
GENERATE

```
SELECT * FROM emp ORDER BY emp.name ASC;
```

Natural Language: - get all city name of city group by pin

SQL generated: - SELECT * FROM city GROUP BY city.pin;


NLP_to_SQL
Coverting Natural Language to SQL
Enter Your Question ?
get all cityname of city group by pin
GENERATE

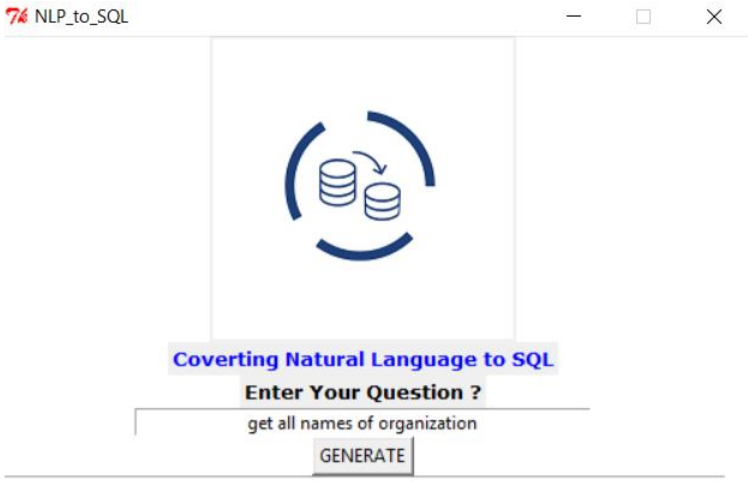
```
SELECT * FROM city GROUP BY city.pin;
```

8. Exception & error handling

Putting names of table which is not in our SQL dump file

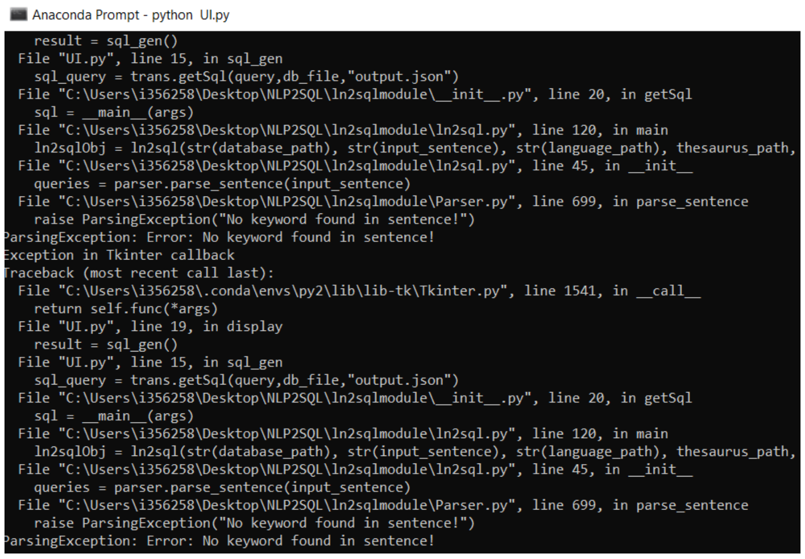
Natural Language: - get all names of organization

SQL generated: - No query generated



get all names of organization

Message given at time of computation , and message is given for every time system is not able to handle unknown keywords.



```
    result = sql_gen()
  File "UI.py", line 15, in sql_gen
    sql_query = trans.getSql(query,db_file,"output.json")
  File "C:\Users\i356258\Desktop\NLP2SQL\ln2sqlmodule\__init__.py", line 20, in getSql
    sql = __main__(args)
  File "C:\Users\i356258\Desktop\NLP2SQL\ln2sqlmodule\ln2sql.py", line 120, in main
    ln2sqlObj = ln2sql(str(database_path), str(input_sentence), str(language_path), thesaurus_path,
  File "C:\Users\i356258\Desktop\NLP2SQL\ln2sqlmodule\ln2sql.py", line 45, in __init__
    queries = parser.parse_sentence(input_sentence)
  File "C:\Users\i356258\Desktop\NLP2SQL\ln2sqlmodule\Parser.py", line 699, in parse_sentence
    raise ParsingException("No keyword found in sentence!")
arsingException: Error: No keyword found in sentence!
xception in Tkinter callback
raceback (most recent call last):
  File "C:\Users\i356258\.conda\envs\py2\lib\lib-tk\Tkinter.py", line 1541, in __call__
    return self.func(*args)
  File "UI.py", line 19, in display
    result = sql_gen()
  File "UI.py", line 15, in sql_gen
    sql_query = trans.getSql(query,db_file,"output.json")
  File "C:\Users\i356258\Desktop\NLP2SQL\ln2sqlmodule\__init__.py", line 20, in getSql
    sql = __main__(args)
  File "C:\Users\i356258\Desktop\NLP2SQL\ln2sqlmodule\ln2sql.py", line 120, in main
    ln2sqlObj = ln2sql(str(database_path), str(input_sentence), str(language_path), thesaurus_path,
  File "C:\Users\i356258\Desktop\NLP2SQL\ln2sqlmodule\ln2sql.py", line 45, in __init__
    queries = parser.parse_sentence(input_sentence)
  File "C:\Users\i356258\Desktop\NLP2SQL\ln2sqlmodule\Parser.py", line 699, in parse_sentence
    raise ParsingException("No keyword found in sentence!")
arsingException: Error: No keyword found in sentence!
```

Figure 15 Exception handling

9. Nested query generation

These types of queries are not covered in the current system architecture

10. Synonyms matching with table name

Synonyms contained in natural language of queries are not covered in the current system architecture

Chapter 7: Modern Translators

As per growth in research of optimising the algorithm and techniques to improvise the NLP to SQl translator modern translator will seems like as per my analysis of this work till now.

The modern translators should be able to overcome these challenges: -

Perceiving natural language: - The ability of system to perceive the intended meaning of question which are ambiguous, diverse and random.

Generating/Decoding Complex Query: - Queries can be amalgamation of nested queries and varying multiple conditions.

Reducing Irrelevant information: - Where Clause conditions on which rows are going to be filter may or may not depend upon the previous, which needs to analyse more accurately.

Designing algorithm to incorporate heterogeneous database system which will increase the scalability and adaptability of the system. Covering the synonyms of table column names to cover the which might increase the accuracy.

After analysis of translators and dataset features, we decided to apply reinforcement learning and transfer learning at classification layer to improve the efficiency and to train our model we propose to use spider dataset because it have cross-domain text-to-query which covers almost all varying combination of query formation with multiple conditions attributes with Observer which observe for user to visualize and communicate with the system if step taken by system needs to be retract its mistake.

Lexical analyser takes text as input and performs analysis and pre-processing using tokenization, lemmatization, removing stop words making input clean for processing by system so that it can be used by classifier to accurately identify cross-ponding query with the relevant tokens identified. With help of observer will user will be able to visualize each step and correct any mistakes which will help improvise the system to predict the correct query. Generated query then will be run on stored database to fetch the result whose accuracy will be compared with the expert generated query.

Types of queries which should be generated by translator are.

- SELECT operation with one column multiple columns and all columns.
- SELECT distinct on columns.
- Aggregate functions such as count-select, sum-select, avg-select, min-select, max-select
- JOIN inner join natural join
- WHERE clause with one condition multiple conditions with operators such as equal operator,
- not equal operator, greater-than operator, less-than operator, like operator, between operator (not 100% efficient)
- ORDER BY with ASC, DESC
- GROUP BY multiple queries

Chapter 8: Results

The result of NlP_to_SQL translator shows that it cans successfully generated SQL queries with column names, from multiple tables, with single condition values and on join conditions in some cases.

The checklist of successfully generated queries are shown below:

S. No	Use Cases	Accuracy (%)	Test Passed
1	Retrieving Table	**100**	**Yes**
2	Selecting columns from the Table	**80**	**Yes**
3	Fetch data using constraints	**70**	**Yes**
4	Using Aggregation to compute &fetch data	**60**	**Yes**
5	Joining Multiple Tables	**70**	**Yes**
6	Like operator	**40**	**No**
8	Exception & error handling	**80**	**Yes**

Table 4 Result

In terms of generating human generated query and NLP_to_SQL generated query the overall accuracy is 62.5%.

This result can be improved in future few points were mentioned about it in Future work section of my collusion of my thesis presnted there were few challenges encountered which I mentioned in challenges section in upcoming section, and how we can improve the translator using modern translator.

Conclusion

Performance of ONLI on standard test dataset have better performance with 83% which is best till now among all systems compared. WikiSQL have large number of SQL queries and question set which gives advantage while training the model, but the limitations of only one single table. Spider table which is proposed covers cross domain a multiple table with varying combinations of queries containing joins, having, etc. clauses. The feedback interactor improves and gives more control in generating the accurate and efficient query.

The translator, I presented is simple to design which generates query based on natural language. It exactly able to generate query which can fetch complete columns, selecting all the data, putting constraints on the selected data to make data more relevant, using aggerate functions to calculate and get overview of fetched data.

This system can also be configured to handle synonyms of table and column names, generated multiple dependent complex queries.

Challenges

There are many points of improvement which can be improved, concerning the point of functionality which will improve the accuracy and make system more efficiently.

The following points can be considered designing the efficient NLP to SQL translator: -

Perceiving natural language: - The ability of system to perceive the intended meaning of question which are ambiguous, diverse and random.

Generating/Decoding Complex Query: - Queries can be amalgamation of nested queries and varying multiple conditions.

Reducing Irrelevant information: - Where Clause conditions on which rows are going to be filter may or may not depend upon the previous, which needs to analyze more accurately.

Future Work

The system can be enhanced by incorporating the below points:

To handle nested queries, recursive algorithm can be designed, which helps create independent queries for condition clause to optimize the result.

Algorithms can be optimizing by formulating multiple queries which cover the same intent of the user semantic.

Designing algorithm to Incorporate heterogeneous database system which will increase the scalability and adaptability of the system.

Covering the synonyms of table column names to cover the which might increase the accuracy.

Addressing the abbreviations in the system to increase the scope at which it can map to relational database.

List of Publication

In area of natural language papers published

- "Role of Sentiment Lexicons in Sentiment Analysis and their Performance"
  Proceedings of the 12 th INDIACom; INDIACom-2018; IEEE Conference ID: 42835
  2018 5 th International Conference on "Computing for Sustainable Global Development", 14th– 16th March 2018 Bharati Vidyapeeth Institute of Computer Applications and Management (BVICAM), New Delhi (INDIA)

References

1. P. Reis, N. Mamede, and J. Matias, "Edite - A Natural Language Interface to Databases: a New Dimension for an Old Approach", Proceeding of the Fourth International Conference on Information and Communication Technology in Tourism, Edinburgh, Scotland, 1997.
2. Enikuomehin A.O, Okwufulueze D.O "An algorithm for solving natural language query execution problems on relational databases" , Editorial Preface, 2012.
3. Benoˆıt Couderc, Ier'emy Ferrero. fr2sql: Querying databasesees en fran¸cais. 22`eme TAutomatic translation of Natural Languages, Jun 2015, Caen, Francid. 2015.
4. Amit Pagrut, Ishant Pakmode, Shambhoo Kariya, Vibhavari Kamble and Yashodhara Haribhakta"AUTOMATED SQL QUERY GENERATOR BY UNDERSTANDING A NATURAL LANGUAGE STATEMENT" International Journal on Natural Language Computing (IJNLC) Vol.7, No.3, June 2018
5. P Yin, Z Lu, H Li, B Kao "Neural Enquirer: Learning to Query Tables in Natural Language", - 1512.00965, 2015
6. "NaLIR: An interactive natural language interface for querying relational databases" V Jagadish, Hosagrahar, 2014/06/18 Proceedings of the ACM SIGMOD International Conference on Management of Data.
7. Victor Zhong, Caiming Xiong, Richard Socher, "Seq2SQL: Generating Structured Queries from Natural Language Using Reinforcement Learning", 16 Feb 2018 ICLR 2018 Conference Blind Submission
8. "An algorithm to transform natural language into SQL queries for relational databases"
9. MA Paredes-Valverde, MÁ Rodríguez-García… "ONLI: an ontology-based system for querying DBpedia using natural language paradigm "- Expert Systems with Applications, 2015 – Elsevier.

10. learning, X Xu, C Liu, D Song “Sqlnet: Generating structured queries from natural language without reinforcement” - 1711.04436, 2017

11. “Natural Language text to SQL query “ COEN-296 Natural language Processing. Amey Baviskar, Akshay borse, Eric white, Umang shah, Winter Quarter 2017.

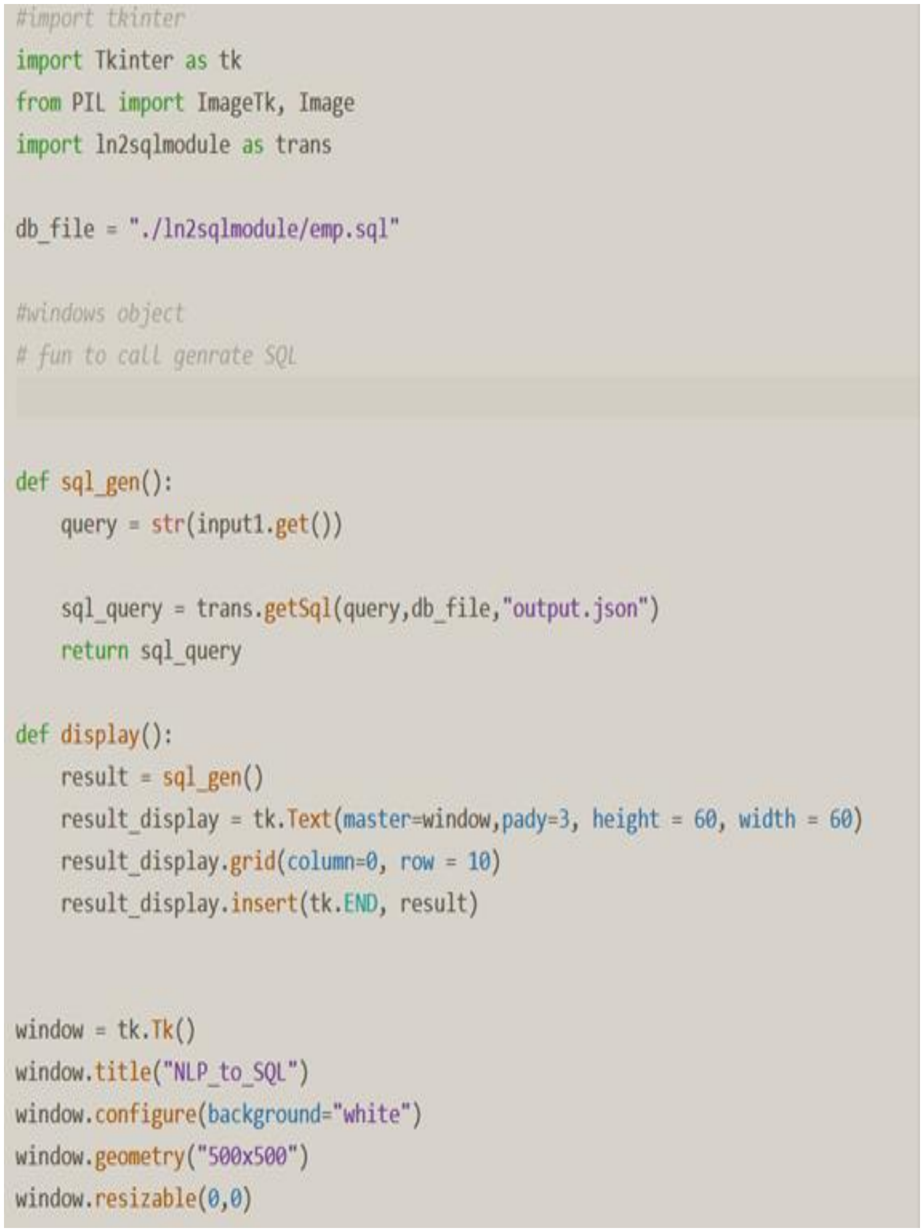
12. “An algorithm to transform natural language into SQL queries for relational databases Garima Singh, Arun Solanki Selforganilogy 2016,3(3):100-116, 12 April 2016,

13. “Automated SQL Query Generator by Understanding a Natural language statement” International journal on Natural Language Computing(IJNLC) Vol 7,No 3 June 2018.

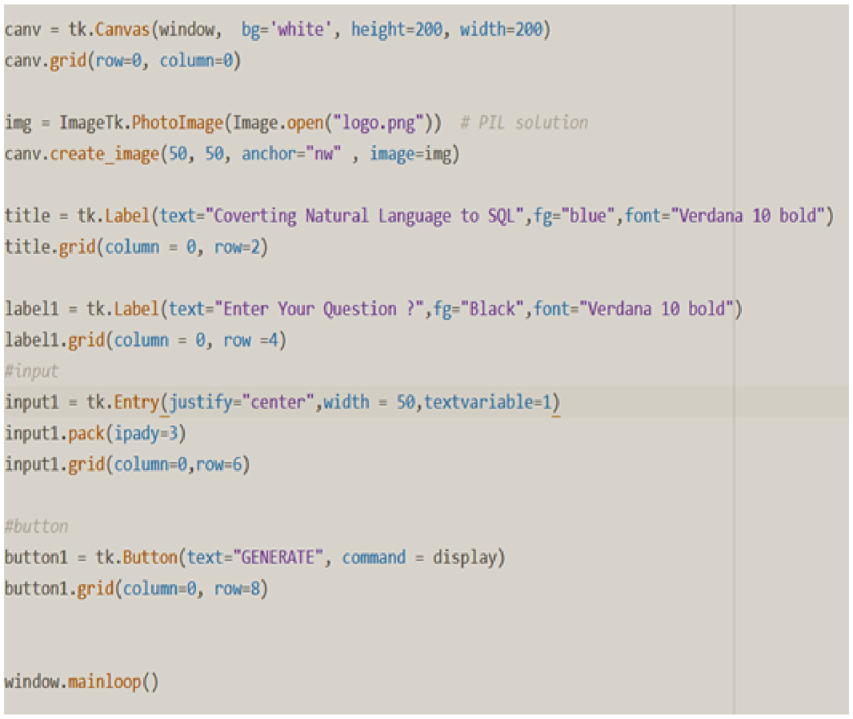
14. fr2sql: Querying databases'ees en fran¸caisBenoˆıt Couderc, Ier'emy Ferrero, TAutomatic translation of Natural languages, Jun 2015, Caen. Hal 01165914.

Appendices

Code for UI



```
#import tkinter
import Tkinter as tk
from PIL import ImageTk, Image
import ln2sqlmodule as trans

db_file = "./ln2sqlmodule/emp.sql"

#windows object
# fun to call genrate SQL


def sql_gen():
    query = str(input1.get())

    sql_query = trans.getSql(query,db_file,"output.json")
    return sql_query

def display():
    result = sql_gen()
    result_display = tk.Text(master=window,pady=3, height = 60, width = 60)
    result_display.grid(column=0, row = 10)
    result_display.insert(tk.END, result)


window = tk.Tk()
window.title("NLP_to_SQL")
window.configure(background="white")
window.geometry("500x500")
window.resizable(0,0)
```



```python
canv = tk.Canvas(window,  bg='white', height=200, width=200)
canv.grid(row=0, column=0)

img = ImageTk.PhotoImage(Image.open("logo.png"))  # PIL solution
canv.create_image(50, 50, anchor="nw" , image=img)

title = tk.Label(text="Coverting Natural Language to SQL",fg="blue",font="Verdana 10 bold")
title.grid(column = 0, row=2)

label1 = tk.Label(text="Enter Your Question ?",fg="Black",font="Verdana 10 bold")
label1.grid(column = 0, row =4)
#input
input1 = tk.Entry(justify="center",width = 50,textvariable=1)
input1.pack(ipady=3)
input1.grid(column=0,row=6)

#button
button1 = tk.Button(text="GENERATE", command = display)
button1.grid(column=0, row=8)


window.mainloop()
```

Output Jason file

```
{
    "select": {
            "column": "cityName",
            "type": "None"
    },
    "from": {
            "table": "city"
    },
    "join": {
    },
    "where": {
    },
    "group_by": {
            "column": "pin"
    },
    "order_by": {
    }
}
```