Project Report

on

AUTOMATIC QUERY GENERATOR WITH AI BOT

In Partial Fulfilment of the Requirements for the Degree of

Bachelor of Technology

in

Information Technology

Submitted By:

Abhay Kumar (1873413001)

Mritunjai Maurya (1873413029)

Rajeshwar Arya (1873413041)

Yash Kumar (1873413060)

Under the Guidance

of

Dr. Vibhash Yadav
Associate Professor

Mr. Shantanu Shukla
Assistant Professor

Department of Information Technology

RAJKIYA ENGINEERING COLLEGE BANDA

(Affiliated to Dr. A.P.J. Abdul Kalam Technical University Lucknow, UP)

June 2022

DECLARATION

We, **Abhay Kumar, Mritunjai Maurya, Rajeshwar Arya, Yash Kumar** students of B.Tech. (4^{th} Year) Information Technology, hereby declare that the report titled **"Automatic Query Generator with AI bot"** which is submitted by us to the Department of Information Technology, Rajkiya Engineering College Banda in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology has not previously formed the basis for the award of any Degree, Diploma, Associateship, Fellowship or other similar title or recognition.

Name of Students	**Signatures**
1. Abhay Kumar	
2. Mritunjai Maurya	
3. Rajeshwar Arya	
4. Yash Kumar	

RAJKIYA ENGINEERING COLLEGE, BANDA

CERTIFICATE

On the basis of declaration submitted by **Abhay Kumar, Mritunjai Maurya, Rajeshwar Arya, Yash Kumar** students of B.Tech. (IT 4th), I hereby certify that the report titled **"Automatic Query Generator with AI bot"** which is submitted to the Department of Information Technology, Rajkiya Engineering College Banda, is an original contribution with existing knowledge and faithful record of research carried out by them under my guidance and supervision.

To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Banda

Date:

Dr. Vibhash Yadav
Associate Professor
(Project Guide & HOD)

ACKNOWLEDGEMENT

We would like to thank **Prof. S. P. Shukla (Director, Rajkiya Engineering College Banda)** without whose guidance we would not have been able to come under one umbrella to work for this project.

We would like to express our vote of thanks towards the **Dr. Vibhash Yadav**
Head of Department and our **Project Guide** who allotted such an insightful guide to us for the project.

We would also like to express our sincere gratitude towards **Mr. Shantanu Shukla** who provided his guidance in every possible way whenever we were stuck to help us out of the dilemma and help us throughout the duration of the project and clearing our doubts.

We also thank all the faculty members for their help rendered.

Abhay Kumar	**1873413001**
Mritunjai Maurya	**1873413029**
Rajeshwar Arya	**1873413041**
Yash Kumar	**1873413060**

ABSTRACT

This project aims to develop a system which converts a natural language statement into MySQL query to retrieve information from respective database. The system mainly focuses on creation of complex queries which includes nested queries with more than two-level depth, queries with aggregate functions, having clause, group by clause and co-related queries which are formed due to constraint on aggregate function. The natural language input statement taken from the user is passed through various OpenNLP natural language processing techniques like Tokenization, Parts of Speech Tagging, Stemming and Lemmatization to get the statement in the desired form. The statement is further processed to extract the type of query, the basic clause, which specifies the required entities from the database and the condition clause, which specifies constraints on the basic clause. The final query is generated by converting the basic and condition clauses to their query form and then concatenating the condition query to the basic query. Currently, the system works only with MySQL database.

Keywords: NLP, SQL, Syntax semantic parsing, Parsing, Data Dictionary, NLTK, Tkinter, NLIDB,

TABLE OF CONTENTS

DECLARATION……..**ii**

CERTIFICATE……….iii

ACKNOWLEDGEMENT..…….. .**iv**

ABSTRACT…….…….. **v**

LIST OF FIGURES…….…viii

LIST OF ABBREVIATIONS……...ix

CHAPTER 1 (INTRODUCTION)

1.1 Motivation:.....1

1.2 Objective:.....2

1.3 Problem Statement:.....2

1.4 Existing Approaches:.....2

1.5 About Project:.....3

1.6 Achievements:...……..……..……..3

1.7 Natural Language Processing: 4

1.8 SQL:.....4

CHAPTER 2 (LITERATURE SURVEY)

2.1 Survey of Existing Systems.....5

2.1.1 Formation of SQL from Natural Language Query using NLP 5

2.1.2 Conversion of Natural Language Query to SQL Query 5

2.1.3 MyNLIDB : A Natural Language Interface to Database.....6

2.1.4 A Question Answer on Structured Data Using NLIDB Approach.....6

2.2 Limitation of existing System……..……..……..…...7

CHAPTER 3 (PROPOSED METHODOLOGY)

3.1 Hardware and Software used……8

3.1.1 Hardware used 8

3.1.2 Software used..... 8

3.2 System Block Diagram 9

3.3 Flowchart 11

3.4 Working 13

3.5 Project Module..... 13

3.5.1 Natural Language Toolkit 13

3.5.2 Database for SQL Query execution..... 13

3.5.3 Desktop Application..... 14

3.6 Database……..…14

3.6.1 Database Creation……..14

3.7 Dictionary.……...15

CHAPTER 4 (IMPLEMENTATION DETAILS)

4.1 Module Description 16

CHAPTER 5 (RESULT AND ANALYSIS)

5.1 Snapshots 19

CHAPTER 6 (CONCLUSION).…….…….……..…24

CHAPTER 7 (FUTURE SCOPE)..……..…….25

REFERENCES……..…….26

APPENDIX: CODE IMPLEMENTATION…….……..…28

LIST OF FIGURES

Figure 3.1 System Block Diagram 9

Figure 3.2 Flowchart 11

Figure 4.1 Implementation Output 17

Figure 5.1 Test Case 1 19

Figure 5.2 Test Case 2 20

Figure 5.3 Test Case 3 21

Figure 5.4 Test Case 4 22

Figure 5.5 Test Case 5 23

LIST OF ABBREVIATIONS

NLIDB	Natural Language Interface to Database
NLP	Natural Language Processing
SQL	Structured Query Language
GUI	Graphical User Interface
AI	Artificial Intelligence
POS	Part of Speech
TLA	Table Linking Algorithm
NLTK	Natural Language Toolkit

CHAPTER 1
INTRODUCTION

1.1 Motivation

Almost all applications in today's world make use of collected data to fulfill the intended requirements and to enhance their functionalities. The main objective remains the efficient storage and fast retrieval of this data. Databases provide a better provision and are the most suitable solution which addresses these objectives. The relational databases provide a structured way to store the huge collection of data and provide real-time accessibility. Relational database management system is representation of domain entities and their respective attributes in the tabular form.

Many organizations and social networking sites make use of relational databases for storage and analysis. In this modern world, everyone aims to be more dynamic in terms of information gathering, retrieval and sharing using existing systems of database storage, but are not enough well versed with the underlying technology. Users need to learn the underlying database language and database properties to generate queries. Natural language is used in day-to-day life, and if information sharing can be made easy with the use of natural language, it will reduce the cost of learning and understanding the technology used for it.

Natural Language Processing is a branch of Artificial Intelligence that is used to create intelligent computers that can interact with humans in the same way that humans do. It connects the human-machine gap. The primary goal of Natural Language Query Processing is to have a computer interpret English Sentence structures. Despite these problems, it is commonly used for research purposes. Natural Language Processing is used to access the database by asking Natural Language questions and receiving the necessary results. Asking questions via natural language to databases is a very convenient and easy method of data access, particularly among users who are unfamiliar with complex database query languages such as SQL (Structured Query Language). This system is intended for users who deal with student databases but lack SQL expertise. The system suggests an algorithm for handling the English Query fired by the user in order to obtain a SQL query using feedback as text or expression. Tokenization, syntactic, and semantic analysis, as well as the use of dictionaries and grammars used for such analysis, can be performed using the natural language to SQL query conversion tool.

A query can be entered in natural language by the user. When the user enters the query in English, it is translated to a SQL query. There are many difficulties in converting natural language queries to SQL queries, such as complexity, which implies that a single term may have several meanings. In this case, a single word may be mapped to several meanings. Another difficulty is the development of complex SQL queries.

1.2 Objective

The objective of our project is to generate accurate and valid SQL queries after parsing natural language using open-source tools and libraries. Users will be able to obtain SQL statement for the major 5 command words by passing in an English sentence or sentence fragment. We wish to do so in a way that progresses the current projects towards robustness and usability. The System proposes the architecture for processing the English Query fired by user to get SQL query using input as the text orspeech. Speech recognition mainly deals with the ability of a machine or program to identify words and phrases in spoken language and convert them to a machine-readable format. Keyword based interface accepts query sort of program query which retrieves keyword from the input query and converts into SQL by applying rules from the generated knowledge domain. Natural Language Interface is applied on various formats of knowledge for natural language input query. The system will contain supporting features such as GUI. On GUI, there the user can enter in natural language. When user will enter of in English Language, query will get converted to SQL query. The user can either view equivalent SQL query or can execute it directly. Objective of this paper is to convert a natural language query into a SQL to simplify data extraction.

1.3 Problem Statement

This project makes use of natural language processing techniques to work with text data to form SQL queries with the help of a corpus which we have developed. En2sql is given a plain english language as input returns a well structured SQL statement as output.

1.4 Existing Approaches

The existing approach is to generate the query from the knowledge of SQL manually. But certain improvement done in recent years helps to generate more accurate queries using Probabilistic Context Free Grammar (PCFG). The current implemented standard is QuePy and similar, disjoint projects like them. These projects use old techniques; QuePy has not been updated in over a year. The QuePy website has a interactive web app to show how it

works, which shows room for improvement. QuePy answers factoid questions as long as the question structure is simple. Recent research such as SQLizer presents algorithms and methodologies that can drastically improve the current open source projects. However, the SQLizer website does not implement the natural english to query aspect found in their 2017 paper. We wish to prove these newer methods.

1.5 About Project

This system is a type of NLIDB system which concentrates on formulating complex queries, along with simple queries. In this project, we take the user natural language query input in English language through the graphical user interface. In Java, OpenNLP library provides various natural language processing modules. The input is then processed through various Natural language processing processes like tokenizing the sentence by the space delimiter, mapping each token with its Parts of Speech (POS) tag using POS Tagger, lemmatizing each token to convert it into its basic form, and these lemmas and tags are stored. The converted statement is now broken down into two parts: Basic clause and Condition clause. The Basic clause identifies the attributes used to form the query and the Condition clause identifies the constraints on these attributes and are mapped to their respective attribute. A Table Linking Algorithm (TLA) is incorporated to find the links between different tables of the query. The basic query and condition query is then joined together to form the final resultant query.

1.6 Achievement

This project claims the following achievements:

1. Developed a model based on Natural Language processing that handle all the NLP queries to convert them into SQL.
2. Generated SQL query will be executed on the database and related information will be displayed on the console window.
3. Accuracy is major factor of any model, and our model is giving the accuracy 95%.

1.7 Natural Language Processing (NLP)

Data generated from conversations, declarations or even tweets are examples of unstructured data. Unstructured data doesn't fit neatly into the traditional row and column structure of relational databases and represent the vast majority of data available in the actual world. It is messy and hard to manipulate. Nevertheless, thanks to the advances in disciplines like machine learning a big revolution is going on regarding this topic. Nowadays it is no longer about trying to interpret a text or speech based on its keywords (the old-fashioned mechanical way), but about understanding the meaning behind those words (the cognitive way). This way it is possible to detect figures of speech like irony, or even perform sentiment analysis.

Natural Language Processing or NLP is a field of Artificial Intelligence that gives the machines the ability to read, understand and derive meaning from human language.

1.8 SQL

SQL is a language to operate databases; it includes database creation, deletion, fetching rows, modifying rows, etc. SQL is an ANSI (American National Standards Institute) standard language, but there are many different versions of the SQL language. SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database. SQL is the standard language for Relational Database System. All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.

SQL is a short-form of the structured query language, and it is pronounced as S-Q-L or sometimes as See-Quell. This database language is mainly designed for maintaining the data in relational database management systems. It is a special tool used by data professionals for handling structured data (data which is stored in the form of tables). It is also designed for stream processing in RDSMS. You can easily create and manipulate the database, access, and modify the table rows and columns, etc. This query language became the standard of ANSI in the year of 1986 and ISO in the year of 1987. If you want to get a job in the field of data science, then it is the most important query language to learn. Big enterprises like Facebook, Instagram, and LinkedIn, use SQL for storing the data in the back end.

CHAPTER 2

LITERATURE SURVEY

We conducted research by reading the research papers and publications on earlier work.

2.1 Survey of Existing Systems

2.1.1 Formation of SQL from Natural Language Query using NLP

This approach uses natural language processing (NLP), with a structured natural language question as input and a SQL query as output, to quickly access information from the railway reservation database. Tokenization, lemmatization, parts of speech tagging, parsing, and mapping are all stages in this process. There are 2880 formal natural language queries on train fares and available seats in the dataset used by the proposed scheme. They reached a precision of 98.89 percent.

Methodology

They have implemented the method in two stages:

i) NLP Phase

-Tokenization

-Lemmatization

-Syntactical Analysis

-Semantic Analysis

ii) Mapping Phase

- Attribute Identification

-SQL Query Formation

Objective

Formation of SQL query using Natural Language Processing on train seats and fare datasets.

Limitations

They have used just one table and have taken input in text format.

2.1.2 Conversion of Natural Language Query to SQL Query

This paper describes a method for converting Natural Language Query to SQL Query that is automated. The user can use speech to enter the question in this system. Speech would be converted to text by the system. This query is going to be converted to a SQL query. The system will run the question and return the results to the customer.

Methodology

They used two dictionaries for their analysis: one for lexical analysis and the other for syntax or semantic analysis. Tokenization, Lexical Analysis, Syntactic Analysis, and Semantic Analysis are all part of the process.

Objective

To make it easier for users to retrieve and handle student data from a database.

Limitations

The findings of this method are not quite accurate.

2.1.3 MyNLIDB: A Natural Language Interface to Database

In this article, they present a method called MyNLIDB that performs well in terms of keyword mapping. They used a Schema-Graph constructed from the underlying database, a Stanford part-of-speech parser, and a dependency parser to convert NL Query to SQL using pipeline analysis. MyNLIDB is an information management system that is domain and database independent. They were able to achieve a high level of precision with this method.

Methodology

Input→Pre-Processing→POS tagger→Node generator→SQL generator→Database→Result

Objective

To build a database with a natural language interface. (i.e., the interface used to access data from a database).

Limitation

It is made only for simple queries.

2.1.4 A Question Answering on Structured Data using NLIDB Approach

In this paper, they used an intermediate query approach to implement a Natural Language Interface to Database (NLIDB) scheme. Their process is shown with a chatbot for the Movie domain, but it can also be extended to other domains. The architecture has performed admirably and can manage the vast majority of user requests pertaining to the database in question.

Methodology

User→NL Query (English)→Tokenizer→POS Tagger→Dependency Parsing→Build SQL query→Execute Query From database→Result

Objective

Develop NLID for alumni database using CFG base system.

Limitations

The accuracy is very poor, at just 47

2.2 Limitations of existing systems and research gaps

In all system discussed in research papers, they have used simple database with only one table which can solve simple queries. Also, every person can speak different wordings but having same meaning which is tedious to get accurate query.

CHAPTER 3

PROPOSED METHODOLOGY

We are planning to use certain methodologies to develop the project, they are Database, Natural Language Processing (NLP), Tkinter for building the front end, etc. We are planning to build a desktop application that will contain an input field where we will give our input and SQL query will be displayed along with the data which will be fetched from the database given.

3.1 HARDWARE AND SOFTWARE USED

3.1.1 Hardware used :-

- Intel Core i5-9300H CPU @ 2.40 GHz
- 8 GB DDR4 Ram
- 4 GB NVIDIA GeForce GTX 1650 Graphic

3.1.2 Software used :-

- **Python:-**

  Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently whereas other languages use punctuation, and it has fewer syntactical constructions than other languages

- **Jupyter: -**

  The IPython Notebook concept was expanded upon to allow for additional programming languages and was therefore renamed "Jupyter". "Jupyter" is a loose acronym meaning Julia, Python and R, but today, the notebook technology supports many programming languages.

- **MySQL:-**

  MySQL is a database management system.

- **Xampp Server**:-

  XAMPP helps a local host or server to test its website and clients via computers and laptops before releasing it to the main server.

3.2 System Block Diagram

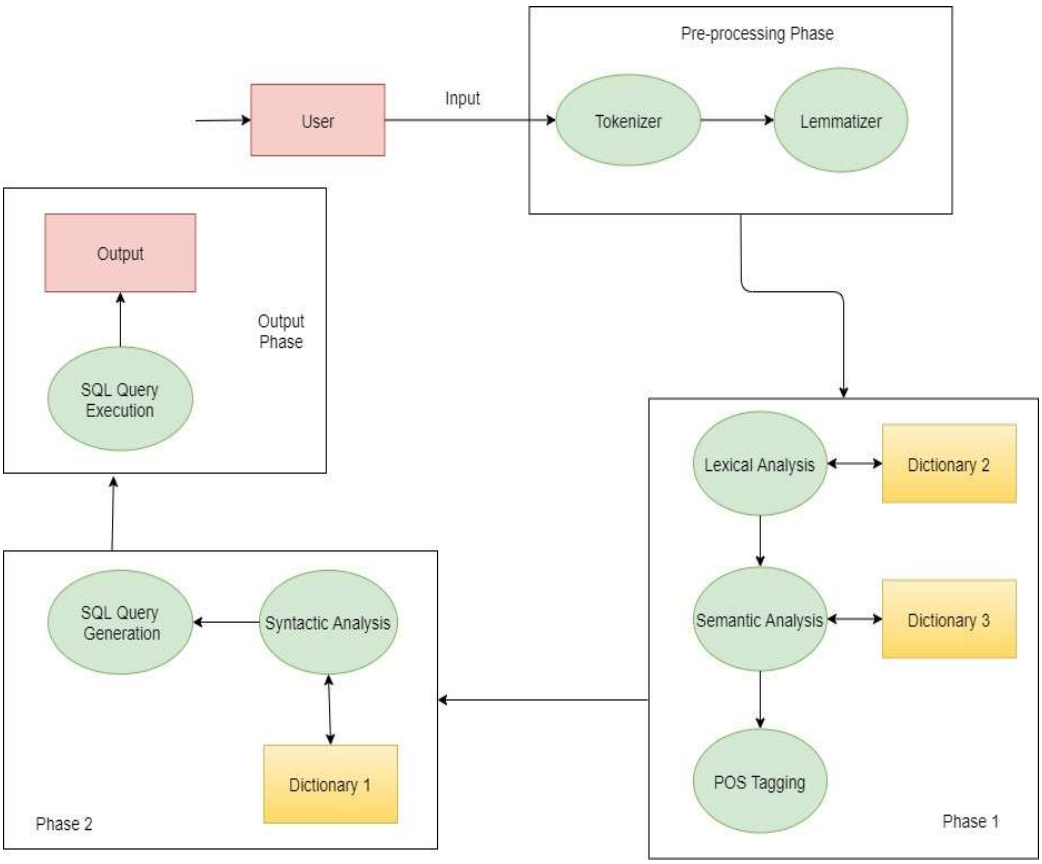


Figure 3.1 System Block Diagram

The block diagram given above represents our approach to the problem. The description of each block of the diagram is given below:

- **Pre-processing Phase:** The aim of this step is to process the input provided according to the algorithm's needs. It aims to remove the unwanted words so that our model can benefit from this improved data to work on.

(i)Tokenizer: In this step, we take input i.e., the natural language and break them into smaller chunks known as tokens.

(ii)Lemmatizer: In this step, we take input that is the tokens, and we focus on use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of word.

- **Lexical Analysis:** The aim of this step is to convert the high-level input into the sequence of the tokens. The output is a sequence of tokens that is sent to the parser for syntax analysis.
- **Semantic Analysis:** In this step we process the statements if it is semantically correct or not. Type checking is an important part of semantic analysis.
- **POS Tagging**: The aim of this step is to process the input if converting a sentence to forms – list of words, list of tuples. The tag in case of is a part-of-speech tag, and signifies whether the word is noun, adjective, verb, and so on.
- **Syntactic Analysis:** In this step is we draw exact meaning, or you can say the exact meaning from the text. It also checks the text meaningfulness comparing to the rules of formal grammar.
- **SQL Query Generation:** In this step after executing all the above steps we conclude by the generation of the query of the natural language provided.
- **SQL Query Execution:** The aim of this step is to process the input provided which is converted in the SQL query in query generation phase to fetch the information from the database which is provided. This is the final output phase of the model.

3.3 Flow Chart

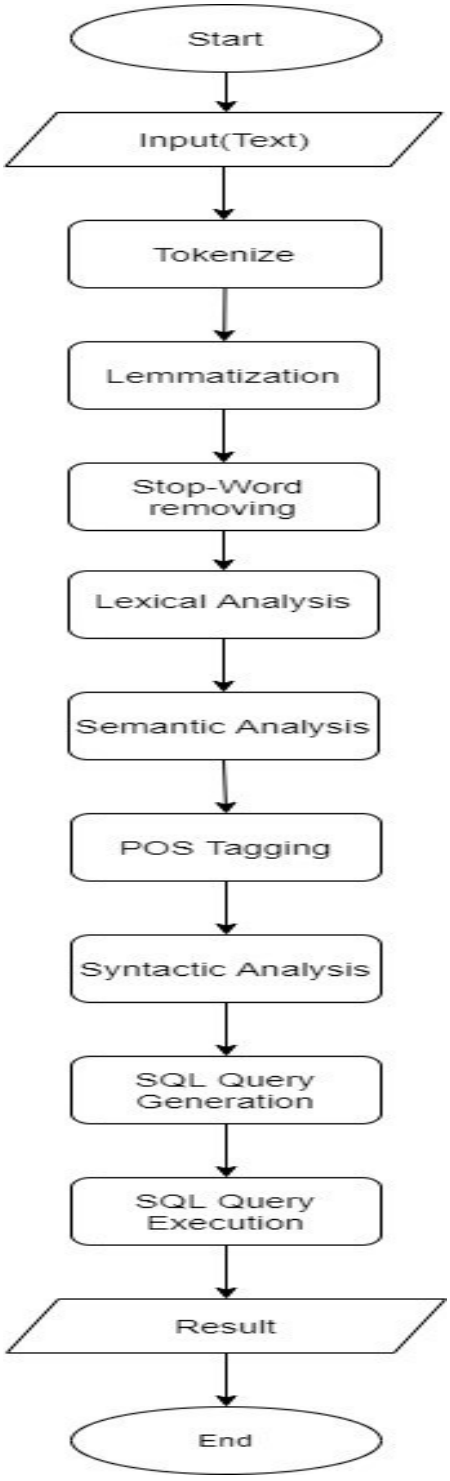


Figure 3.2 Flowchart

Step 1: Query entered by the user (in text) in Natural Language is stored in a string.

Example: fetch all the information of the students who have scored less than 80 in 12^{th}.

Step 2: Considering the spaces in a sentence, it is split into individual words (tokens). These tokens are stored in a separate list.

These are the tokens for above query.

['fetch', 'all', 'the', 'information', 'of', 'the', 'students', 'who', 'have', 'scored', 'less', 'than', '80', 'in', '12th']

Step 3: We will use lemmatization to get the lemma of the tokens generated.

['fetch', 'all','the', 'information', 'of', 'the', 'student', 'who', 'have', 'scored', 'less', 'than', '80', 'in', '12th']

Step 4: Tokens are compared with the ignore list, only important tokens are considered for further processing.

['fetch', 'all', 'student', 'who', 'have', 'less', '80', '12th']

Step 5: Selected tokens are mapped with database words. These tokens gets replaced by their synonyms.

['SELECT', '*', 'FROM student', 'WHERE', 'less', '80', 'hsc']

Step 6: Identify the conditions or values specified by the user in his/her query, if any. Those conditions are mapped with the database words. (For example, Less than is replaced with " < " symbol)

['SELECT', '*', 'FROM student', 'WHERE', '<', '80', 'hsc']

Step 7: Parts of Speech tagging of tokens is done here. The tag in case of is a part-of-speech tag, and signifies whether the word is a noun, adjective, verb, and so on.

[('SELECT', 'NNP'), ('*', 'NNP'), ('FROM student', 'NNP'), ('WHERE', 'NNP'), ('<', 'VBZ'), ('80', 'CD'), ('hsc', 'NN')]

Step 8: Syntax analysis checks the text for meaningfulness comparing to the rules of formal grammar.

['SELECT'] ['*'] ['FROM student'] ['hsc'] ['<'] ['80'] []

Step 9: SQL Query generation

SELECT * FROM student WHERE hsc < 80

Step 10: SQL Query will be executed on the database and user will be provided with output.

3.4 Working

a) The natural language will be given by the user in the input field for which the query is to be generated.

b) Input will be passed through the various phases and the SQL generation will take place.

c) This SQL query will be used in the SQL execution phase to fetch the information from the database.

d) SQL Query and information fetched from the database will be displayed on the screen of the user.

3.5 Project Module

There are three modules in our project which is used to develop the application for SQL Query Generation. They are:

- Natural Language Toolkit(nltk)
- Database for SQL Query execution
- Desktop Application

3.5.1 Natural Language Toolkit(nltk)

The Natural Language Toolkit (NLTK) is a platform used for building Python programs that work with human language or the natural language data for applying in statistical natural language processing (NLP).

It contains text processing libraries for tokenization, parsing, classification, stemming, tagging and semantic reasoning. It also includes graphical demonstrations and sample data sets as well as accompanied by a cookbook and a book which explains the principles behind the underlying language processing tasks that NLTK supports.

It also contains the libraries such as lemmatizer, tokenizer, punkt package, wordnet package, and averaged perceptron tagger (POS tag).

Database for SQL Query execution

The SQL Query generated from the SQL Query generator is used for the fetching the information from the database which is created.

Database consists of the columns i.e., id, names, marks in the 10^{th}(ssc) and 12^{th} (hsc) etc. which can be updated as per the need. Information fetched from the database is displayed to user on console.

3.5.2 Desktop Application

The desktop application consists of the GUI (Graphical User Interface) which is use for the displaying of the SQL query along with the data fetched from the database.

Desktop Application is made up of the Tkinter framework which has the various inbuild function which we have used for building the GUI.

This application is then connected to database using the nltk library i.e., mysql-connector

for connecting the database with our web app.

3.6 Database

In this model database is the main component of working as model is generating the SQL query related to natural language query and this generated query will be corresponding to any database as SQL query contains some attributes of database. Now model has generated a SQL query and this generated query will be executed on the user's (attached with model) database so that users can fetch the desired output. Since model can be used by different type of organizations or users that means database attached with model also can be changed according to customer's requirements. For the development of this model, students database is attached with the model having six attributes (id, age, name, ssc, hsc, grad) and rows or entries depends on user's data.

3.6.1 Database Creation

```
CREATE TABLE 'student' (
   'id' int (10) NOT NULL,
   'age' int (10) NOT NULL,
   'name' varchar (50) NOT NULL,
   'ssc' varchar (10) NOT NULL,
   'hsc' varchar (10) NOT NULL,
   'grad' varchar (10) NOT NULL
 )
```

3.7 Dictionary

Dictionary is one of the most important parts of this model as there are some sets of rules to be followed by the model for generating the SQL query. Since the generation of SQL query have some steps to be followed and each step is associated with some dictionaries for example step template generation has a dictionary and lexical analysis also has a dictionary and semantic analysis also has a dictionary attached. These dictionaries also work as a filter or synonym checker for tokens.

CHAPTER 4

IMPLEMENTATION DETAILS

4.1 Module Description

There are three modules in our project which is used to develop the application for bird detection. They are:

- Tokenize module
- Lemmatized and Stop-Word module
- Lexical module
- Semantic module
- POS_tagging module
- Syntactic module
- SQL query generation module

Tokenize module

System will perform tokenization on the entered query by separating it into single words. Each word represents a token. Then these words will be stored in a separate list and passed to Lemmatized and Stop-Word module.

Lemmatized and Stop-Word module

System will perform lemmatization on the output of the tokenized module and stop word module removes all the unwanted words from the list using ignore list. Then this will be stored in separate list and pass to the lexical module.

Lexical module

The lemmatized list will be mapped with the dictionary. These words will get replaced by the database words from the dictionary and passed to syntactic analysis.

Semantic module

System will find words which represent conditions or symbols, and that word will get mapped with the dictionary. (For Example: If there is "less than or equal to" in the query, it will get mapped with the symbol "<=").

POS_tagging module

Parts of Speech tagging of tokens is done here. The tag in case of is a part-of-speech tag, and signifies whether the word is a noun, adjective, verb, and so on.

Syntactic module

In this step dictionary of table names, attributes and keywords are maintained. Each tokenized word gets mapped with attributes in the dictionary.

SQL query generation module

In this module, SQL query will be generated using the output of the syntactic module.

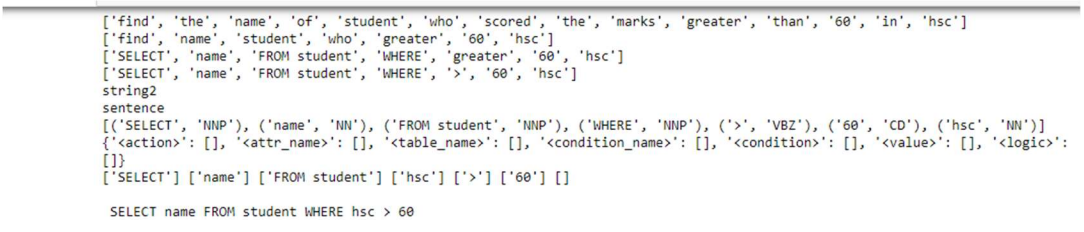


Figure 4.1 Implementation Output

CHAPTER 5

RESULT AND ANALYSIS

This system converts Natural Language Query to SQL Query efficiently and effectively. Tokenization, syntactic, and semantic analysis, as well as the use of dictionaries and grammar is used for such analysis is performed using the natural language to SQL query conversion tool.

Libraries used – MySQL, Tkinter, word_tokenize, wordNetLemmatizer, pos_tag by nltk. System has achieved accurate results for many natural language queries.

5.1 Snapshots :

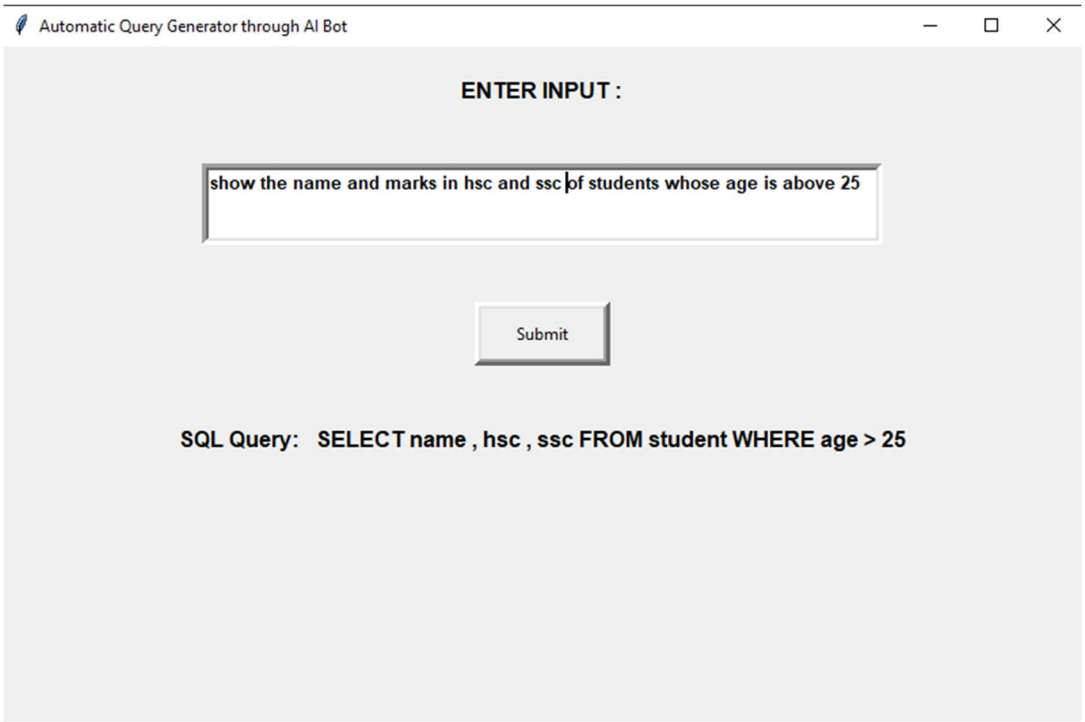


Figure 5.1 Test case 1

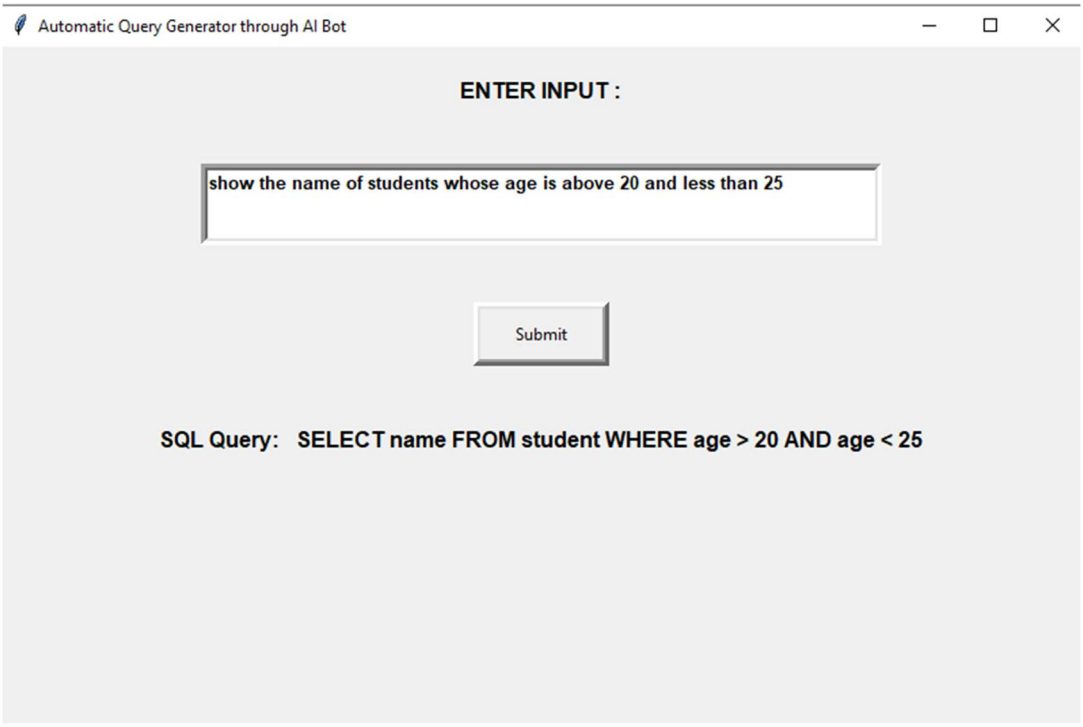


Figure 5.2 Test Case 2

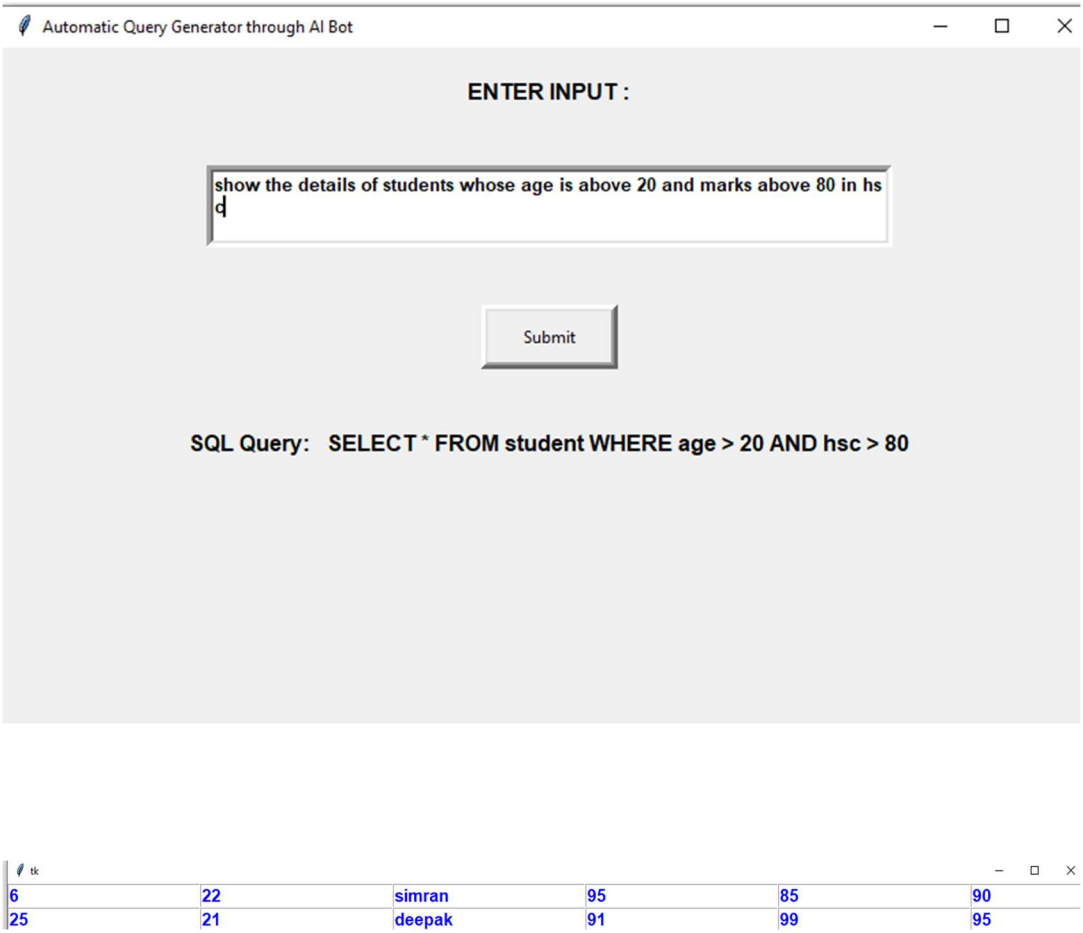
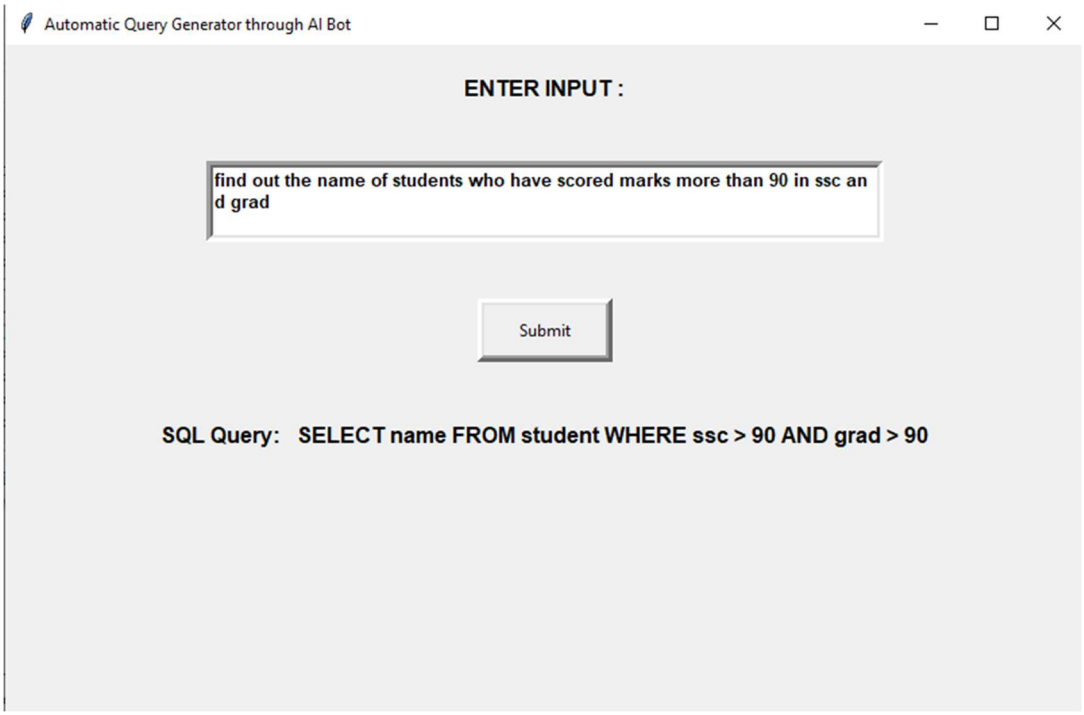


Figure 5.3 Test Case 3



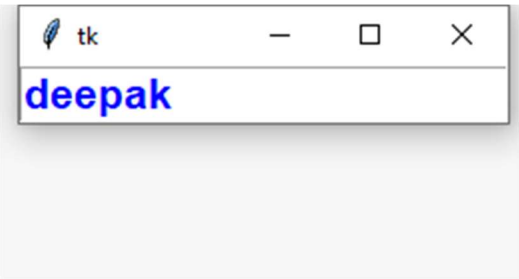
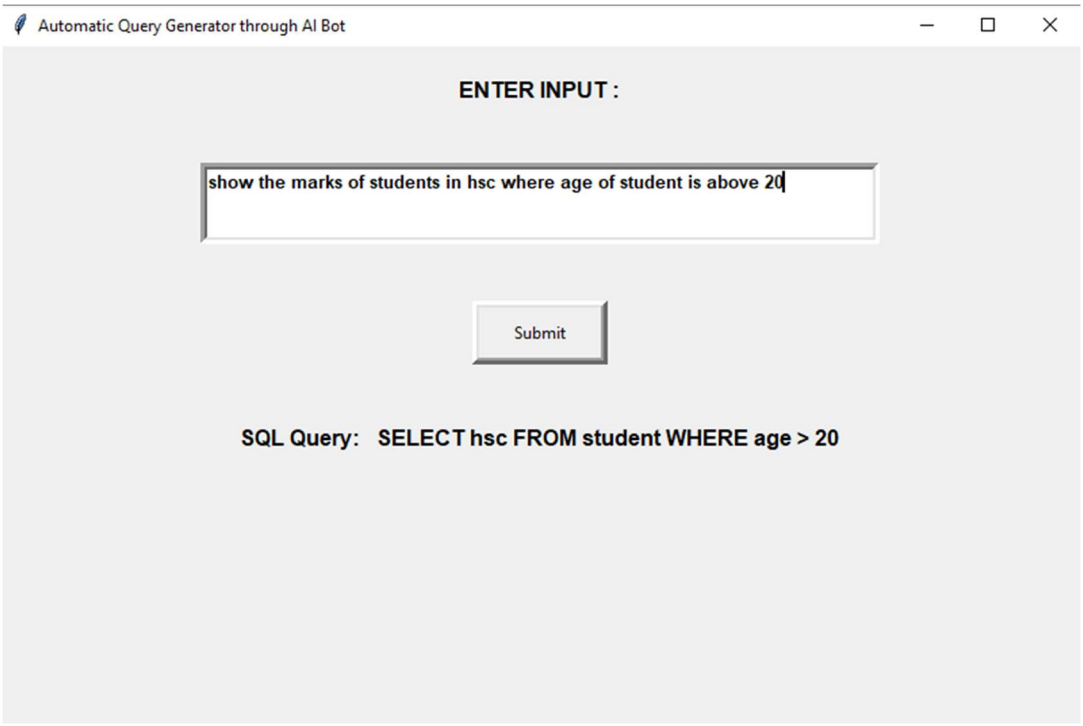


Figure 5.4 Test Case 4



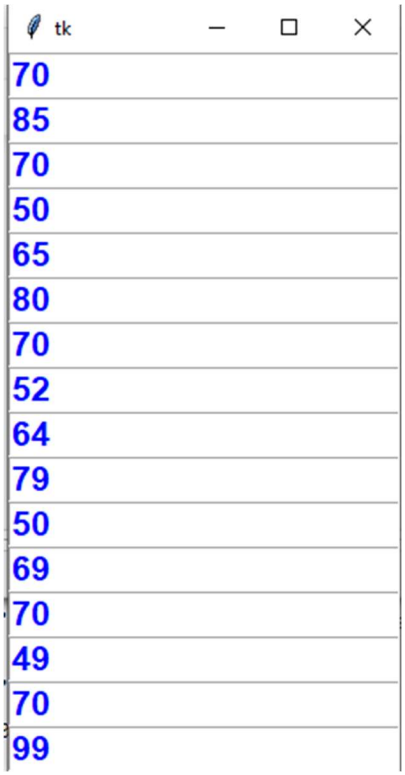


Figure 5.5 Test Case 5

CHAPTER 6

CONCLUSION

Any human being can benefit from using Natural Language. Using plain English, this system will assist users in retrieving and managing data from the student database. The user does not need to learn SQL or any other complicated query language. The system is user-friendly due to its ability to receive input in text format. Our system will convert natural language queries into SQL language queries and supply users with the necessary information from the student database.

CHAPTER 7

FUTURE SCOPE

The following are some potential enhancements that might be implemented: The input can be in the form of audio, which may then be transformed to text. The SQL query might be more complicated. In terms of attributes and tuples, the database might be bigger. There may also be many tables with relevant data that may be accessed with the JOIN keyword. To make the system more interactive, the output may be transformed into a sentence and then into audio format. This work could also be expanded to other languages.

REFERENCES

[1] Uma, M., Sneha, V., Sneha, G., Bhuvana, J., & Bharathi, B. (2019). Formation of SQL from Natural Language Query using NLP. 2019 International Conference on Computational Intelligence in Data Science (ICCIDS).

[2] Das, A., & Balabantaray, R. C. (2019). MyNLIDB: A Natural Language Interface to Database. 2019 International Conference on Information Technology (ICIT).

[3] Kate, A., Kamble, S., Bodkhe, A., & Joshi, M. (2018). Conversion of Natural Language Query to SQL Query. 2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA).

[4] Reshma, E. U., & Remya, P. C. (2017). A review of different approaches in natural language interfaces to databases. 2017 International Conference on Intelligent Sustainable Systems (ICISS).

[5] Li, Q., Li, L., Li, Q., & Zhong, J. (2020). A Comprehensive Exploration on Spider with Fuzzy Decision Text-to-SQL Model. IEEE Transactions on Industrial Informatics, 16(4), 2542–2550.

[6] Mahmud, T., Azharul Hasan, K. M., Ahmed, M., & Thwoi Hla Ching Chak. (2015). A rule-based approach for NLP based query processing. 2015 2nd International Conference on Electrical Information and Communication Technologies (EICT).

[7] Sangeeth, N., & Rejimoan, R. (2015). An intelligent system for information extraction from relational database using HMM. 2015 International Conference on Soft Computing Techniques and Implementations (ICSCTI).

[8] Kumar, R., & Dua, M. (2014). Translating controlled natural language query into SQL query using pattern matching technique. International Conference for Convergence for Technology-2014

[9] Agrawal, R., Chakkarwar, A., Choudhary, P., Jogalekar, U. A., & Kulkarni, D. H. (2014). DBIQS — An intelligent system for querying and mining databases using NLP. 2014 International Conference on Information Systems and Computer Networks (ISCON).

[10] Shah, A., Pareek, J., Patel, H., & Panchal, N. (2013). NLKBIDB - Natural language and keyword-based interface to database. 2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI).

[11] Siasar djahantighi, F., Norouzifard, M., Davarpanah, S. H., & Shenassa, M. H. (2008). Using natural language processing in order to create SQL queries. 2008 International Conference on Computer and Communication Engineering.

[12] Adwait Ratnaparkhi, "Learning to Parse Natural Language with Maximum Entropy Models," Machine Learning, vol. 34, issue 1-3, Feb. 1999, pp. 151-175.0

[13] I. Androutsopoulos, G. D. Ritchie and P. Thanisch, “Natural Language Interfaces to Databases –An Introduction,” in Natural Language Engineering, vol. 1, part 1, 1995, pp. 29- 81

[14] George A. Miller (1995), “WordNet: A Lexical Database for English,” in Communications of the ACM Vol. 38, No. 11: 39-41.

[15] Ana-Maria Popescu, Oren Etzioni and Henry Kautz, “Toward a Theory of Natural Language Interfaces to Databases,” in IUI Proceedings 2003

APPENDIX
CODE IMPLEMENTATION

```
pip install mysql-connector
// It is installation of mysql-connector module which is used to connect the database to model.

pip install word2number
//It is installation of word2number module which is used to convert number (one) word to numeric digit (1).

from word2number import w2n
// Importing module into model.

import tkinter as tk
from tkinter import *
// tkinter is Package which is used create GUI interface.

import nltk
// It is a toolkit build for working with NLP in python.

from nltk.tokenize import word_tokenize
//We use the word_tokenize method to split a sentence into tokens or words

from nltk.stem import WordNetLemmatizer
//

import mysql.connector
// Importing mysql connector for connecting to the database to the model.


import dict
```

```
// Importing the dictionary to the model.

import nltk
nltk.download('punkt')
// It is a tokenizer.

nltk.download('wordnet')
// It is a English dictionary used in NLP.

nltk.download('averaged_perceptron_tagger')
// This is the tagger used to tag the part of speech using averaged.

frame = tk.Tk()
//defining the tkinter as frame.

frame.title("TEXT TO SQL")
// Setting up the title of the GUI.

frame.geometry('800x500')
// setting up the console size.

dict1= {
   "<action>":[],
   "<attr_name>":[],
   "<table_name>":[],
   "<condition_name>":[],
   "<condition>":[],
   "<value>":[],
   "<logic>":[]
    }
dictionary1=dict.dictionary11
```

```
dictionary3=dict.dictionary3

// Creating and using dictionary understanding the synonyms words to model.




def printInput():
    word_list=[]
    word_list=module1()
    print(word_list)
    tokens_without_sw=[]
    tokens_without_sw=module2(word_list)
    print(tokens_without_sw)
    string1=[]
    string1=module3(tokens_without_sw)
    print(string1)
    string2=[]
    string2=module4(string1)
    print(string2)
    sentence=[]
    sentence=module5(string2)
    print(sentence)


    action,attr_name,table_name,condition_name,condition,value,logic=module6(sentence)
    print(action,attr_name,table_name,condition_name,condition,value,logic)

    string4=""
```

```
    string4=module7(action, attr_name,table_name,condition_name,condition,value,logic)
    print(string4)

    module8(string4)

// This is used to tokenize the sentence into words.

def module1():

    text = inputtxt.get(1.0, "end-1c")
    text = text.lower()
    word_list = word_tokenize(text)
    return word_list

// This module is used to lower the capital text and send back to word_list array.

def module2(word_list):
    lemmatizer = WordNetLemmatizer()
    lem=[]
    for r in word_list:
        lem.append(lemmatizer.lemmatize(r))


    ignore_list =['the','record','database','table','information','a','are','is','to','marks','mark','of','i
n','than',"'s",',','.','me','you','us','an','score','scored']
    tokens_without_sw = [word for word in lem if not word in ignore_list]
    return tokens_without_sw

// Lemmatize is the process of grouping together of different form into single items.
// Ignore list is the list of words which is ignore the unnecessary words form NLP query.

def module3(tokens_without_sw):
```

```
global dictionary1



to_be_append=""
location_is=100
selected_is=0

string1=[]
counter=0
set_word=0
for x in tokens_without_sw:
    i=0
    if x in dictionary1["ssc"]:
        tokens_without_sw[i]="ssc"
    elif x in dictionary1["aggregate"]:
        tokens_without_sw[i]="aggregate"
    elif x in dictionary1["name"]:
        tokens_without_sw[i]="name"
    elif x in dictionary1["hsc"]:
        tokens_without_sw[i]="hsc"
    elif x in dictionary1["DESC"]:
        tokens_without_sw[i]="DESC"
    elif x in dictionary1["ASC"]:
        tokens_without_sw[i]="ASC"
    elif x in dictionary1["SELECT"] and selected_is==0:
        tokens_without_sw[i]="SELECT"
        selected_is=1
    elif x in dictionary1["*"]:
```

```
    if location_is > i:
        if counter==0 :
            tokens_without_sw[i]="*"
            counter=1

elif x in dictionary1["WHERE"]:
    tokens_without_sw[i]="WHERE"
elif x in dictionary1["AND"]:
    tokens_without_sw[i]="AND"
elif x in dictionary1["OR"]:
    tokens_without_sw[i]="OR"

elif x in dictionary1["FROM student"]:
    if location_is==100:

        tokens_without_sw[i]="FROM student"
        location_is=i
elif x in dictionary1["ORDER BY"]:
    tokens_without_sw[i]="ORDER BY"
elif x in dictionary1["word"]:
     tokens_without_sw[i]=str(w2n.word_to_num(x))



else:
    tokens_without_sw[i]=x
if to_be_append!=tokens_without_sw[i]:
    to_be_append = tokens_without_sw[i]
    string1.append(tokens_without_sw[i])
if counter==1 :
      tokens_without_sw[i]=""
```

```
            counter=2

        i=i+1
    return string1

// This module is used to match the words from dictionaries of NLP and SQL query words.

def module4(string1):
    global dictionary1
    global dictionary3
    string2=[]
    i=0
    countercounter=0
    to_append=""
    len_string1=len(string1)

    for x in string1:

        if x in dictionary3["<"]:

            if string1[i-1]=="not":
                string1[i]=">"
            else:
                string1[i]="<"
        elif x in dictionary3[">"]:

            if string1[i-1]=="not":
                string1[i]="<"
            else:
                string1[i]=">"
```

```
elif x in dictionary3["="]:

    string1[i]="="


elif x in dictionary3["* FROM"]:

    string1[i]="* FROM"

elif x in dictionary1["AND"]:

    countercounter=0
elif x in dictionary1["OR"]:

    countercounter=0

elif x in dictionary1["<number>"]:

    if i < len_string1:
        if countercounter==0:
            try:
                if string1[i+1] in dictionary1["<number>"]:
                    string1[i]=str(int(string1[i+1])+int(x))


                    countercounter=1
            except:
                print()

else:
    string1[i]=x
if to_append!=string1[i] and string1[i]!="":
```

```
            to_append = string1[i]
            string2.append(string1[i])

        if countercounter==1 :
            string1[i+1]=""
            countercounter=2



        i=i+1


    return string2
```

// This module is used to convert the words to symbol like more than and less than.

```
def module5(string2):
    print("string2")
    sentence=nltk.pos_tag(string2)
    print("sentence")
    return sentence
```

// This module is used for part of speech tagging of words.

```
def module6(sentence):
    global dict1
    print(dict1)
    global dictionary1
    global dictionary3

    dict= {
```

```
  "NN": ["ssc","hsc","name","id","student","*"],
  }
i=0
locc=0
for x in sentence:
    x1,x2=x
    if(x1=="WHERE"):
        locc=i
    i=i+1
i=0

set_select=0
for x in sentence:

    x1,x2=x
    if i<locc and locc!=0:
        if x1 in dictionary1["SELECT"]:
            if set_select==0:
                dict1["<action>"].append(x1)
                set_select=1
        elif  x2=="NN" and x1!="FROM student":
            dict1["<attr_name>"].append(x1)
        elif  x2=="VB" and x1=="aggregate":
            dict1["<attr_name>"].append(x1)
        elif x1=="FROM student":
            dict1["<table_name>"].append(x1)
        elif x2!="NN":
            if x1 in dict["NN"]:
                dict1["<attr_name>"].append(x1)
    elif i>locc and locc!=0:
        if x2=="NN" and x1!="FROM student":
```

```
            dict1["<condition_name>"].append(x1)
        elif  x2=="VB" and x1=="aggregate":
            dict1["<condition_name>"].append(x1)
        elif x2!="NN":
            if x1 in dict["NN"]:
                dict1["<condition_name>"].append(x1)
    else:

        if x1 in dictionary1["SELECT"]:
            if set_select==0:
                dict1["<action>"].append(x1)
                set_select=1
        elif  x2=="NN" and x1!="FROM student":
            dict1["<attr_name>"].append(x1)
        elif  x2=="VB" and x1=="aggregate":
            dict1["<attr_name>"].append(x1)
        elif x1=="FROM student":
            dict1["<table_name>"].append(x1)
        elif x2!="NN":
            if x1 in dict["NN"]:
                dict1["<attr_name>"].append(x1)


    if x2=="$" or (x1 in dictionary3):
        dict1["<condition>"].append(x1)
    elif x1 in dictionary1["AND"]:
        dict1["<logic>"].append(x1)
    elif x1 in dictionary1["OR"]:
        dict1["<logic>"].append(x1)
    elif x2=="CD":
        dict1["<value>"].append(x1)
```

```
    i=i+1

action = dict1["<action>"]

attr_name = dict1["<attr_name>"]
arr=[]
j=1
for i in attr_name:
    if i=="*" and len(attr_name)>1:
        attr_name.remove("*")
for i in attr_name:
    if i!="*":
        arr.append(i)
        if j<len(attr_name):
            arr.append(",")
    else:
        arr.append("*")
    j=j+1
attr_name=arr




table_name = dict1["<table_name>"]

condition_name = dict1["<condition_name>"]
count=0
for i in condition_name:
    if i=="":
        count=count+1
for j in range(count):
```

```
    condition_name.remove("")

  condition = dict1["<condition>"]
  value= dict1["<value>"]
  logic= dict1["<logic>"]
  dict1.clear()
  dict1={
  "<action>":[],
  "<attr_name>":[],
  "<table_name>":[],
  "<condition_name>":[],
  "<condition>":[],
  "<value>":[],
  "<logic>":[]
    }

  return action,attr_name,table_name,condition_name,condition,value,
logic

// take the output of previous module and compare this from dictionary of proceeding to
query generation.

def module7(action,attr_name,table_name,condition_name,condition,value,logic):
  len_condition_name = len(condition_name)


  len_condition = len(condition)


  len_value = len(value)

  len_logic = len(logic)
```

```
phase1 = action + attr_name+ table_name

phase2=""
#lenx=maximum(len_condition_name, len_condition, len_value)

a=len_condition_name
b=len_condition
c=len_value
def maximum(a,b,c):

    if (a>=b) and (a>=c):
        largest = a

    elif (b >= a) and (b >= c):
        largest = b
    else:
        largest = c

    return largest


# Driven code

lenx=maximum(a, b, c)

if lenx==len_condition_name:
    print()
else:
```

```
    cond=condition_name[len_condition_name-1]

    for x in range(lenx-1):
        condition_name.append(cond)
len_condition_name=len(condition_name)

for i in range(len_condition_name):
    phase2 = phase2 + " " + condition_name[i]

    if i <len_condition:
        phase2 = phase2 + " " + condition[i]
    elif i>= len_condition:
        phase2 = phase2 + " " + condition[len_condition-1]
    if i <len_value:
        phase2 = phase2 + " " + value[i]
    elif i>= len_value:
        phase2 = phase2 + " " + value[len_value-1]
    if len_condition_name > len_logic and i < len_logic:
        phase2 = phase2 + " " + logic[i]




result =""
for i in phase1:
    result = result + " " +i
if phase2!="":
    string4=(result +" "+"WHERE"+ phase2)
else:
    string4=(result)
return string4
```

```
// In this module SQL Query generation takes place from NLP Query.

myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "",database = "student")

// This line is used to connect the database to the model for proceeding the SQL query.

def module8(string4):

    resultlist=[]
    myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "",database = "student")

    #creating the cursor object
    cur = myconn.cursor()

    try:
        #Reading the Employee data
        cur.execute(string4)

        #fetching the rows from the cursor object
        result = cur.fetchall()
        resultlist = result
        #printing the result

        #print(resultlist)
        #for x in resultlist:
```

```
except:
    print("error")
    myconn.rollback()

myconn.close()
class Table:

    def __init__(self,root):

        # code for creating table
        for i in range(total_rows):
            for j in range(total_columns):

                self.e = Entry(root, width=20, fg='blue',
                               font=('Arial',16,'bold'))

                self.e.grid(row=i, column=j)
                self.e.insert(END, resultlist[i][j])
if resultlist != [] :
    total_rows = len(resultlist)
    total_columns = len(resultlist[0])

    # create root window
    root = Tk()
    t = Table(root)
    root.mainloop
    lbl.config(text = "SQL Query:  "+string4)
```

```
    else :
        print("No Data Found")
        lbl.config(text = "SQL Query:  "+string4)

        lbl1.config(text = "No Data Found")
```

// This module is used to Proceed the SQL query and fetching the data from SQL database and give the desired output according to asked query.

```
lbl10= tk.Label(frame, text = "ENTER INPUT :",font='Helvetica 12 bold')
lbl10.pack(pady=20)
inputtxt = tk.Text(frame,
                   height = 3,
                   width = 70,
                   bd=5,
                   font='Helvetica 10 bold'

                  )
lbl6 = tk.Label(frame, text = "")
lbl6.pack()
inputtxt.pack()
lbl4 = tk.Label(frame, text = "")
lbl4.pack()
lbl5 = tk.Label(frame, text = "")
lbl5.pack()

# Button Creation
printButton = tk.Button(frame,
                        text = "Submit",
                        bd=5,
                        height = 2,
                        width = 12,
```

```
                    command = printInput)
printButton.pack()
lbl8 = tk.Label(frame, text = "")
lbl8.pack()
lbl9 = tk.Label(frame, text = "")
lbl9.pack()
# Label Creation
lbl = tk.Label(frame, text = "",font='Helvetica 12 bold')
lbl.pack()
lbl2 = tk.Label(frame, text = "")
lbl2.pack()
lbl1 = tk.Label(frame, text = "")
lbl1.pack()
// This is the creation of GUI for taking input from the user.


frame.mainloop()
// this line is used to run the GUI from which user can give the input to the system.
```