Natural Language Based Report Generator

By K. D. Ranidu Nadeesha

169322P

Supervised By Mr. Saminda Premarathne

Faculty of Information Technology

University of Moratuwa

February 2019

Natural Language Based Report Generator

Software tool providing easy report generating for web-based applications

K. D. Ranidu Nadeesha
169322P

Dissertation submitted to the Faculty of Information Technology, University of Moratuwa, Sri Lanka for the partial fulfillment of the requirements of the Degree of MSc in Information Technology.

Faculty of Information Technology
University of Moratuwa
February 2019

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another Masters, Degree or diploma at any university or other institution of tertiary education. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

Name of the Student:	Signature of the Student
K. D. Ranidu Nadeesha	………………………….. Date:
Supervised by:	Signature of the Supervisor
Mr. Saminda Premarathne	…………………………… Date:

Dedication

This dissertation is dedicated to my beloved parents, erandi and her parents, my teachers who gave me endless courage and support to achieve my task and goal in completing the research project.

Acknowledgment

My heartiest thanks go to my supervisor Mr. Saminda Premarathne for the guidance, assistance, encouragement, valuable bunch of advice on improving the research and providing this opportunity to carry out this research project.

Also, sincerely thanks to all my teachers who taught in MSc IT degree program. Things learned from these subjects made it easier to make this research project a successful one.

Last but not least, a sincere thank goes to everyone who supported especially teams from Redot Pvt Ltd and Venuerific Pte Ltd Singapore for contributing their valuable time on this research.

Abstract

Information plays a major role in our lives. It's a key element in day to day life. We use databases to store a large amount of data and retrieve it efficiently in less amount of time. It also plays a major role in computer systems. This information access by technical and nontechnical users. When it comes to non-technical users they should be able to know Structured Query Language to access data from databases. The goal of NLP[1] is to allow communication between people and computers without resorting to memorizing complex commands and procedures. In other words, NLP is a technique that can make the computer understand the languages used naturally by humans, but not by artificial language or created by man, as a programming language This minimize the communication gap between computer and humans. In order to archive this type of communication, we have to make the computer to understand what a human asks.

The natural language may be the symbol system easier to learn and use for people, it has proven to be the most difficult to master for a computer. Despite the challenges, natural language processing, or NLP, is widely regarded as a promising and critically important effort in the field of computer research. The general objective for the majority of computational linguists is to imbue the computer with the ability to understand and generate language so that, finally, people can approach their computers through text as although they were addressing another person. The applications that will be possible when NLP capabilities are fully realized are impressive computers that could process the natural language, translate languages with precision. and in real time, or extract and summarize information from a variety of data sources, depending on the requests of the users.

To use store data inside databases or retrieve information from databases, interaction with the database is important and for this purpose, technical knowledge is required. The problem is that it restricts the interaction between the naive user and the database. Only a few people who have knowledge of the formal language of the database can retrieve the desired information from the database. To overcome this problem, this proposed system would have the ability to analyze the statements of users written in different ways and, consequently, gives a response to the user. In this way, it helps a normal educated person who has no knowledge of the query language to easily interact with the system.

This project proposes a customized solution for avoiding communication gap between non-technical user and database. Reducing the time spent on retrieve data is the main objective of this project which ultimately leads to a software application. The proposed system is capable of retrieve data from database by understanding human natural language questions. By keeping real time data structure JSON[2] format it easily adapts every database changes as well as the provide UI interface to enter questions and display results on the frontend. Written controllers and models are capable of understanding and generating results based on the question. Keeping in mind of the latest technologies and the mobile technology evolvement, Tool is capable of accessing it from desktop or mobile. For better performance, I have used Python3[3] scripting and flask frontend library for UI generating. I have tested the system with the collogues at my workplace which all of them are developers. And currently using the system for generating first and second level prototypes.

Table of Contents

Table of Tables

Table of Figures

Chapter 1

1. Introduction

1.1 Prolegomena

The modern world has already been taken over by technology, as it has come to a point where it is not even possible to live without it. Before 20 years ago, no one has ever imagined that technology would become so advanced that technology would play a significant role in areas such as food processing, clothing, medical research, water management, electricity plants, etc. Considering the complicated day to day life of a human being and the rapid growth of the world population, it won't even be able to cater for basic human needs without the modern-day technology. While technology is playing a significant role in delivering basic human needs, it has also become the ultimate solution provider for economics, transportation, sales and marketing, banking and lots of other areas.

Since technology started connecting people all around the world with the innovation of the world wide web, websites and web pages were introduced just for delivering information. It was initially used for data communication and very rarely for marketing. When the technology started evolving and becoming more popular, WWW has begun to provide a vast variety of solutions to people around the world such as sales and marketing, entertainment, streaming technologies, mailing services, etc. As the requirement for these services increased rapidly, software development companies began getting a large number of information stored in their servers and processing that information for identifying patterns, etc.

One of the major sources of information in databases. Databases contain a collection of related data, stored in a systematic way to model a part of the world. In order to extract information from a database, one needs to formulate a query in such a way that the computer will understand and produce the desired output. However, not everybody is able to write such queries, especially those who lack a computer background.

Nowadays, applications use many types of databases as a data storage method. The complexity of these databases will continually increase due to the size and complexity between entities. The main problem is that to retrieve user information from the database, for that should be able to write structured queries, that will be difficult for normal users to do. A solution for this is, the developed a natural language interface to retrieve information from databases. Translating natural language into structured query language is not an easy task. Not only due to the ambiguity of natural language, but also because users can make mistakes when writing Natural language input, such as incorrect spelling. Natural language interface of the first days for Database systems was generally based on small-scale databases. That supports a small set of queries. Design of a natural language interface for the database (NLIDB)[4] for a system is proven by several investigations. Although we have some applications that are not limited to a number of queries. However, even so, these projects require human intervention to map words in English to SQL query terms and are limited to a few database engines. There is a need to design an NLIDB system that can address the scalability of applications which adapt to any database. The main objective of generating natural language report generator using NLIDB is to overcome report limitations of user generation.

This research suggests overcoming these problems by building a Natural language based report generator which adapts to any database and handling by complex queries. This is mainly a dynamic rule-based system. This work as a tool and easily can be included ina any project.

1.2 Background and Motivation

When the natural language interfaces to database (NLIDB) research started in the nineteen sixties, During that time they concerned about one database at a time as the implementation target, because of that NLIDB systems has been limited to one specific database at a time. The best example for NLIDB system in the late sixties in Lunar. This system which developed for a database which contained data about chemical analysis of moon rocks.

In the mid-eighties, the natural language interface to the database become a very popular research area. A numerous number of NLIDB systems had implemented. Some of the applications were even bought to commercial use but because of the lack of acceptance probably due to the difficulties to understand the natural language they did not get the expected gain.

There are few Natural languages based query generators available around the world. Some of them are already discontinued like Microsoft English Query for SQL server. Other famous applications are Kueri.me[5], quepy[6], etc. These applications work better with Natural language processing in order to generate Structured Query Languages. But the problem is all of them are not out of box solutions, technical knowledge required to configure those tools and designed to work with only one database.

All these tools are providing great advantages to the user by in process of retrieving data from databases from input natural language questions but still, there is no common tool available which support any database and scale according to it.

1.3 Problem domain

The limitation of such programs is that they restrict the interaction between the user and the database to a predefined set of queries. Only a few people who know the structure of databases and the formal language of databases can retrieve the desired information from the database. A novice user who has no knowledge of the structure of the database and the formal database query language cannot retrieve the desired information. It is not backed by a good thought application. Therefore, it is a need to improve the human-computer interface that allows people to interact with the database in their natural language. The problem of existing systems is mainly designed to work with one database. The proposed system work with any connected database. It will go through the database and define some dynamic rules according to the tables in the database and generate Structured Queries according to it.

1.4 Hypothesis

By conducting extensive research we have identified some limitations with existing works. Our proposed solution for this is an implementation of a software tool which captures user questionnaires and output results based on that.

This tool helps any non-technical users to retrieve data from databases without any knowledge of query languages. This tool can be integrated into software as a separate module. This is a dynamic rule-based tool because of that this tool can be adapted to any database.

1.4.1 Expected Features

- Able to work with any database.
- Support below SQL statements
    - Select
    - Distinct select
    - Multiple column select
    - Aggregate functions (Count, Sum, Average, Min and Max)
    - Conditions
        - Join
        - Where (Multiple Conditions, Equal, Greater than, Less than, Like, Equal and Not Equal)
    - Order By
    - Group By
- Generate SQL statements by understanding user questionnaires
- Can be integrated into any existing system

1.5 Aim and Objectives

1.5.1 Aim

The aim of the proposed system is the interaction between the naive user and the intelligent system. The proposed tool would have the ability to understand the natural language, know the meaning of the query and process it, and give the desired result.

1.5.2 Objectives

- Implement a software tool which will translate the user's normal language, such as English to the SQL query, to obtain the result of the database, where the user does not need to learn anything related to the database. and can interact directly with the database in its known language.
- Implementing this kind of tool help the naïve user to overcome the report generating limitations. Can be integrated with existing web-based systems and understand natural language questions and generate SQL queries based on that.

1.6 Structure of the Thesis

The best of the thesis is organized as follows. Chapter 1 critically reviews the literature on current prototyping tools and source code generation tools such as CRUD[7] tools. Chapter 2 is discussing current developments. As well as the issues and technologies different people/ organizations have used to overcome the issues with prototyping and automated code generation. Chapter 3 is about the technologies used in implementing the solution. Chapter 4 presents a new approach to build fully functional prototypes using schema definitions. Chapter 5 and 6 describe the design and implementation respectively. Chapter 7 is on the evaluation of the solution and it discusses 2 case studies conducted. Chapter 8 concludes the research with a note on further work.

1.7 Summary

This chapter describes the general description of the research and introduces the research problem and its solution. The next chapter is the literature review that will discuss the work of other researchers in the same field. It will provide comprehensive, detailed information on necessary project information based on a literature review, as well as information on current development challenges.

2. Current Development and Challenges

2.1 Introduction

Chapter 1 is a detailed description of the overall project described in this thesis. This chapter provides a critical review of the literature in relation to developments and challenges in natural language processing and a query generating. An earlier research, software and article review has been submitted under three main categories. Early development, modern trends and future challenges. In the end, this chapter defines the research problem.

2.2 Current Developments

The first attempts at NLP database interfaces are as old as any other NLP research. In fact, the database, NLP may be one of the most important successes in NLP since it began. Asking questions in natural language databases is a very convenient and simple method to access data, especially for occasional users who do not understand the complicated database query languages such as SQL[8]. Success in this area is due in part to the real-world benefits that can come from NLP database systems, and partly because NLP works very well in a single database domain. The databases generally provide domains small enough so that problems of ambiguity in natural language can be solved successfully. LUNAR[9] (Woods, 1973) involved a system that answered questions about rock samples brought from the moon. Two databases, chemical analyzes, and bibliographic references were used. The program used a transition network syntax analyzer (ATN) and Woods Processing Semantics. The system was informally demonstrated at the Second Annual Lunar Science Conference in 1971. LIFER / LADDER[10] was one of the first good NLP database systems. It was designed as a natural language interface for a database of information about US Navy ships. UU This system, as described in a document by Hendrix (1978), used semantic grammar to analyze questions and consult distributed databases. The

LIFERILADDER system could only support simple queries or multiple tables. consultations with easy JO In conditions. By the late seventies, more NLIDBs had appeared. RENDEZVOUS this was the first general-purpose NLIDB module. This engaged the user in dialogues to help him/her formulate his/her queries. ELF this was the first commercial NLIDB system. It provides UI based application to access a desktop database. This program understands plain-English and converts it into a Structures Query Language.

2.2.1 Quepy

This is a python framework to transform natural language questions into queries in a database query language. It can be easily customized to different kinds of questions in natural language and database queries. So, with some coding user can build your own system for natural language access to your database. Currently, it provides support for SPARQL and MQL query languages. This uses an abstract semantics as a language-independent representation that is then mapped to a query language. This allows your questions to be mapped to different query languages in a transparent manner.

2.2.2 CINDI

The digital library system being developed at Concordia University is a virtual library built to facilitate the registration of digital resources and their subsequent bibliographic search. CINDI [11]is based on the use of Semantic Headers that store relevant information for search and 7 discovery: these are the usual search terms such as authors name, The title of the contents of the digital resources, it's subject classification, abstract, etc. stored in CINDIs Semantic Header database. Cindi's interface sub-system uses an expert system to guide the user in the search tasks at hand. Once the user input is processed, the CINDI system uses the Semantic Headers database to retrieve the information from the resource catalog. CINDI thus provides a mechanism to register, manage and search a bibliography and provide access to the actual resources once the search is successful. The focus of the current paper is to report on the design and implementation of a Natural Language interface for the CINDI system.

2.2.3 NLS-to-SQL

Is the natural language statement which is the requirement specified by the user in terms of questions, which is given to the system. Internal system is divided into preprocessor and postprocessor and input is given to the preprocessor. Furthermore, the preprocessor will decide the type of the query (Select/Delete), substitute numerals and comparison operators, remove apostrophe and replace it by corresponding construction and the elimination of useless words and recognizing the keyword (extracts the noun clause). The extracted noun clauses may be the table or column name. Then the output from the preprocessor is given to the post processor as input in which it initially recognizes the items in the statement (table values), extracts the table name also aggregate function. Now the actual query formation starts. The postprocessor then decomposes the statement based on the words were whose, which, such that, etc. Then postprocessor decides the query template type. And using this template query is translated into its SQL equivalent.

2.2.4 Fr2Sql

Is a querying database in French This is a tool takes in input as a SQL dump file and generate the database model and a sentence and translate the latter in a valid SQL statement able to query the input data model. At first, the user selects the base of data that he wishes to interrogate and enters a request in the field provided for this purpose. The application will then extract the meaning-bearing words of the user-entered request. Then it will recover the structure and the information needed for its use in the database just selected. Using a thesaurus, a match is sought between the keywords extracted from the application user and the entities in the database. A division of the input request is made according to the correspondences found. A second search is made to find words that can indicate during selection. Once the application has determined which type of request was requested and on which elements, depending on the division made and the correspondences found, it generates a query. The generated query is then executed on the database and results will be displayed to the user.

2.2.5 Natural Language Web Interface for Database (NLWIDB)

Is concerned with translating user's query from English into its corresponding SQL query to retrieve the data from the relational database via the Internet. The result will then be displayed to the user. This NLWIDB[12] system specially develops for the English language as an initial step and other languages will be considered later. An algorithm has been developed efficiently to map a natural language question, entered in English, to convert an SQL statement for producing suitable answers. The algorithm has been implemented with PHP, Apache, MySQL and tested successfully. The NLWIDB system architecture which depicts the layout of the process included in converting user question in NL query into a syntactical SQL query to be fired on the RDBMS and getting answers from the database. The NLWIDB system architecture which depicts the layout of the process included in converting user question in NL query into a syntactical SQL query to be fired on the RDBMS and getting answers from the database.

2.3 Research Gap

Based on the above to identify the research gap, we evaluated the above-mentioned researches and generated a table to the following criteria. These criteria were selected based features provided by them and a varied range of support.

Research	Multiple database support	Multiple database driver support	Identify the relationship among tables	Advanced Query handling
Quepy	Yes	Yes	No	No
CINDI	No	No	Yes	No
NLS to SQL	No	No	No	No

Fr2Sql	No	No	Yes	Yes
NLWIDB	No	No	Yes	Yes
Proposed System	Yes	Yes	Yes	Yes

Table 2-1: Evaluation done for existing tools

2.4 Summary

In this chapter, the researchers' previous work has been critically evaluated by reading their research papers and articles. The benefits of each of the identified models were considered, an in-depth analysis was conducted to determine which measures were supported by the previously defined criteria, based on the literature review undertaken by previous research. In the next chapter, we will discuss the technology that has been adopted in the proposed solution, mainly focused on deliverables.

Chapter 3

3. Technology foundation of the solution

3.1 Introduction

In the previous chapter, an in-depth discussion and analysis of the researcher's work were conducted. And we identified the research gap to be filled. This chapter discusses the technologies that will be used in the proposed solution.

3.2 The technology used for the solution

Various up to date technologies were used to improve functionalities, support to some advanced features and easy up developments.

3.2.1 Python

Is a High-level language very readable language, Which supports rich text analysis. Python supports great Natural Language Processing libraries like NLTK[13], TextBlob, spaCy, etc. For this project, we using the NLTK library for analyzing sentences. In this case, this is the only language which supports Natural Language Toolkit (NLTK).

3.2.2 Natural Language Toolkit

NLTK is a leading platform for building Python programs to work with human language data. Provides easy-to-use interfaces to more than 50 body and lexical resources such as WordNet, along with a set of text processing libraries for classification, tokenization, derivation, labeling, parsing and semantic reasoning, wrappers for industrial strength NLP libraries. NLTK is intended to support research and teaching in NLP or closely related areas, including empirical linguistics, cognitive science, artificial intelligence, information retrieval, and machine learning.

3.2.3 Django

Django[14] is a free open-source full-stack Python framework. It contains by default all the necessary features instead of offering them as separate libraries. It makes the language more lightweight. Django ORM used to communicate with the database is more reliable because of that it isn't difficult to transfer from one database to another database.

3.2.4 Flask

Flask[15] is designed to develop more solid web-based applications because of that it's more lightweight. It supports extensions which can be added to the project depending on the developer requirement. This does not have a database layer to support data retrieving operations. More focused on frontend view rendering

3.2.5 JSON

JSON (JavaScript Object Notation) is a lightweight data interchange format. Easy to human to read and write. It is used primarily to transmit data between a server and web application, as an alternative to XML can be used to store information.

3.2.6 Bitbucket

Bitbucket[16] is a web-based source control repository hosting service management solution design by Atlassian.

3.3 Development tools

Development tools used to speed up development when implementing a solution.

3.3.1 Visual Code

Visual studio code[17] is an integrated development environment IDE developed by Microsoft. It includes support for debugging, embedded Git control, syntax highlighting, intelligent code completion, snippets, and code refactoring. It is also customizable.

3.3.2 PhpMyAdmin

phpMyAdmin[18] is a free and open source administration tool for MySQL and MariaDB. It helps to perform various tasks such as creating, modifying or deleting databases, tables, fields or rows; executing SQL statements; or managing users and permissions database with providing user friendly graphical interface written in PHP Also it support to export and import databases.

3.3.3 MAMP

The name MAMP[19] is an acronym representing the original components of the system: macOS, the operating system; Apache, the web server; MySQL, the database management system; and PHP, Perl, or Python, programming languages used for web development. The name is derived from LAMP, a similar stack of all open-source software widely used for web sites, but substituting proprietary macOS for open-source Linux. (Similar "AMP" stacks exist for other operating systems.) MAMP is not limited to these choices of components, however; Nginx can be used in place of Apache, for example. The developers of MAMP have also ported the system to Windows (but still called MAMP). Some of the software packages that comprise MAMP (particularly Apache and PHP) are pre-installed with macOS; compatible versions of the remainder are readily available for installation and use, which MAMP facilitates. MAMP is commonly used with popular CMS programs such as WordPress and Drupal for setting up a local development environment. MAMP Pro is a commercial extension to the MAMP base package, which adds features to aid in managing the development of WordPress-based web sites, enabling simultaneous installations of multiple web sites on a single development machine.

3.4 Hosting and Deployment technologies

3.4.1 AWS EC2

Amazon EC2[20] provides scalable computing capacity in the Amazon Web Services (AWS) cloud. Using Amazon EC2 saves you the expense of investing in hardware, allowing you to develop and deploy applications faster. You can use Amazon EC2 to launch as many virtual servers as you want, configure security and networking, and manage storage. Amazon EC2 allows you to expand or reduce your requirements to handle changing requirements or popularity spikes, reducing your need to forecast traffic.

3.4.2 Beanstalk

Beanstalk[21] service also offered from Amazon Web Service (AWS) cloud. Beanstalk reduces management complexity without restricting choice or control. The developer can easily upload a project using provided amazon CLI and the rest of the deployment part handle by beanstalk itself. Like capacity provisioning, load balancing, scaling and application health monitoring.

3.5 Summary

In this chapter addressed the technologies and tools used for the proposed solution. The Application fully developed using python 3 and use of Django and Flask frameworks, for the Natural Language Processing used NLTK library. There are other technologies to support development such as MySQL, PhpMyAdmin, MAMP, AWS EC2, and Beanstalk. Next chapter we will discuss how going to apply those technologies to develop the solution.

Chapter 4

4. A new approach to natural language processing

4.1 Introduction

In the previous chapter briefed about the technologies which adapt to build the proposed solution and this chapter will be discussing the approach used to language processing tool.

4.2 Requirement Gathering

To gather requirement to build this proposed tool a considerable amount of time spent on studying previous solutions and reading research papers, articles done by other researches. Secondly based on the personal experience as a work in software engineer for the past six years I have personally faced problems when building data gathering reports according to clients requirements for customized ERP solutions not only in Sri Lanka but also Singapore and Indonesia.

4.3 Issues faced by the clients in the industry

When it comes to issues faced by the clients, clients use systems to store and process there information's. As information storing method we use a database. The relational databases provide a better provision structured way to store the enormous collection of information and accessibility. To retrieve the correct information from the database user should have sufficient technical knowledge of structured query language (SQL). Because of that, every client would not be able to retrieve information from the stored database. This makes the client depend on the developer when retrieving new information from the database.

From this solution tool allow the client to enter necessary information questions in natural language and tool will process those questions into structured query language and provide the required information in real time.

4.4 Gaps identified hasn't been addressed by other researches

As extensive discussion conducted on gaps identified by other researchers in the literature review. Most of the researches targeted and design into one specific database when providing a solution for natural language processing into the structured query language. Because of that, those tools won't be able to use as a middleware tool for existing systems. Our solution tool able to convert natural language questions into structured query language with any connected database by understanding database structure, relations between tables and writing dynamic rules according to it.

4.5 Limitations with current development tools

We have discussed current development tools, and limitations of them in the literature review chapter and aim of this research to address all these significant issues faced and identified by us.

4.6 Hypothesis

The hypothesis was built on the requirement gathered as mentioned above. This research based on the focusing reduces the gap between non-technical user and the retrieving the data from the database. Our hypothesis is that the build tool which identifies user natural language questions and generates SQL structured queries based on dynamic rule based database structure and retrieves data according to the questions.

4.6.1 Natural Language Processing

Our main goal is to reduce the gap between non-technical user and data gathering from the database for that our system be able to understand natural language questions which asked by the user. We hypnotically propose a tool which can be used for mention situation. As an example, this tool can be used for existing Enterprises resource planning (ERP)[22] for report generating.

4.6.2 Query generators

The proposed system should be capable of generating SQL queries based on users natural language questionnaires. It should able to identify table names columns and the relationship between tables to provide an exactly expected query. At the same time, it should also generate a query which supports any database driver.

4.6.3 Rule based database schema

Generating rule-based database schema according to the database make an easier proposed system to identify database behavior. The system should be able to adapt to any given database to identify user questionnaires. I am having a rule-based database schema reduce the complexity of the reading through databases.

4.7 The user of the system

In various scenario, users can be a range of technical users to non-technical users. Everyone can get benefits from using the proposed tool.

4.8 The input to the system

As a first input of the system it identifies connected database structure, then going through it, it identifies all the relationships between tables and table structures then write them as a rule in the database schema file. It makes it easier to perform fast to generate queries for user input questionnaires.

As a secondary input tool captures user natural language questionnaires once it obtained by the tool it removes all stop words and sends to natural language processing unit, inside this unit sentence check for spellings using natural language toolkit. Once corrections are made the sentence is split into word tokens this called as tokenization Then all tokenize words are parsed into categorizing them as a Noun, Verb, Adjectives, Adverbs, Preposition, Conjunction, Pronoun, and Interjection this process called POS tagging. Once those are categorized where semantic analysis made, inside this process where tool identity tables names match words.

4.9 Outputs of the system

The output of this proposed tool generates SQL queries to retrieve data from the connected database based on asked questions from the user. This tool used the bag of words to match words with tokenizing words and Sentiment Analysis to identify what kind of questions the user has asked and what it means. This proposed tool able to generate join queries based queries by identifying relationships between tables and support other operations as well as column selection, table selection, multiple selections, count, multiple condition checks.

4.10 Features

When it comes to features of proposed natural language query generator we can split them into several categories.

4.10.1 System Specific Features

4.10.1.1 Database Identification

This tool can work with any given database currently it limited only to MySQL database model in future development plan to develop to support for multiple DBMS systems such as SQL, Postgres and NoSQL databases such as MongoDB.

When Tool startup it went through a connected database and write rules according to the database into JSON file. This reduces the time taken to identify table structure and relationship between them.

4.10.1.2 Question Support

This Tool able to identify any natural language question asked by the novel user. Once question process into the tool it converts question into Structured query language to meet with query standards. Able to handle up to some advanced query's as well. Current develop limit only to Query generation part but it can easily be extended into display results according to the query.

4.11 Summary

This chapter explained the approach that allowed us to propose a solution that could fill the research gap. This made it possible to formulate a hypothesis according to which, by Identifying database structure, Processing novel user question by processing into natural language identification we can generate structured query language as the output. And we discussed the relevant steps/approaches as well as the components needed for the solution. In the next chapter, we will discuss in detail the design of the system.

Chapter 5

5. Analysis and Design of the new natural language tool

5.1 Introduction

The previous chapter gave a full picture of the approach to the proposed natural language based query generator tool. This chapter describes the design of the solution for the process presented in the approach. We design the solution which will able to understand novel user questions and process them to structured query language to retrieve data from the database. Here we describe the top-level architecture of the design by elaborating on the role of each component of the architecture. We will discuss the System design, platform design, Infrastructure design of the system throughout this design chapter.

5.2 Research planning

Time is taken to planning and the scheduling of the proposed natural language based query generator tool is shown in the table below. Most of the time spent on learning on required technology, find solutions for the limitation on the development and literature survey.

Task	Q1			Q2			Q3			Q4		
	Feb	Mar	Apr	May	June	July	Aug	Sept	Oct	Nov	Dec	Jan
Litreature Review	▨	▨	▨									
Identify the Problem		▨	▨	▨	▨							
System Design				▨	▨							
Implementation						▨	▨	▨	▨	▨	▨	
Testing												▨

Table 5-1: Execution plan for the proposed system

5.2.1 Development methodology

The evolutionary prototyping methodology was used to implement the system, due to the research component involved in the project. Many Numbers of fine-tuning rounds were required to get the best possible prototype generated.

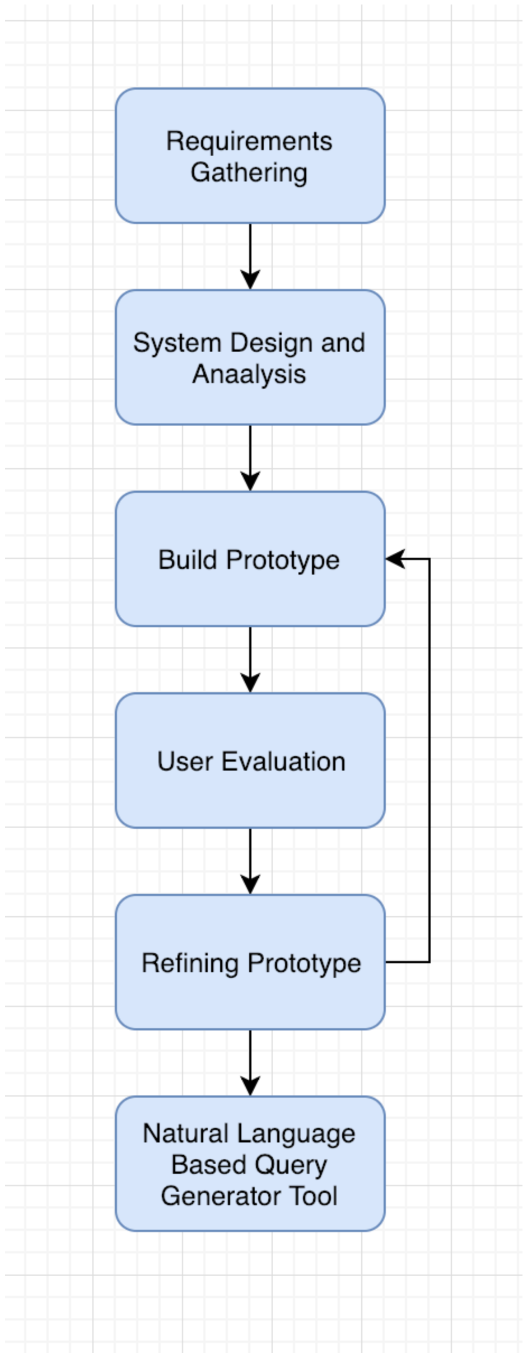

Requirements Gathering
System Design and Anaalysis
Build Prototype
User Evaluation
Refining Prototype
Natural Language Based Query Generator Tool

Figure 5-1: 2Evolutionary prototyping methodology used for the proposed system

5.3 Requirement analysis

Application usage can be described in the below steps.

- Identify connected database table structure and relationships.
- System process the natural input language questions from the user.
- System display generated Structured Query Language query based on questions.

5.3.1 Functional requirement of the solution.

The main goal of this research is to implement Query generator tool based on natural language questions This tool should allow users to enter natural language questions and following that question system should generate queries for data retrieving from the database.

5.3.2 Non-functional requirements of the solution.

Real-time query generator plus the goal of this project is to reduce the gap between non-technical user and data retrieving data from the database. This tool enables users to retrieve data from the a database without having prior knowledge of Structured Query Language.

5.4 Top level design architecture

As the figure is shown in below, top-level design architecture of the proposed system contains 3 main modules. Which is the database module, natural language processing module, database driver

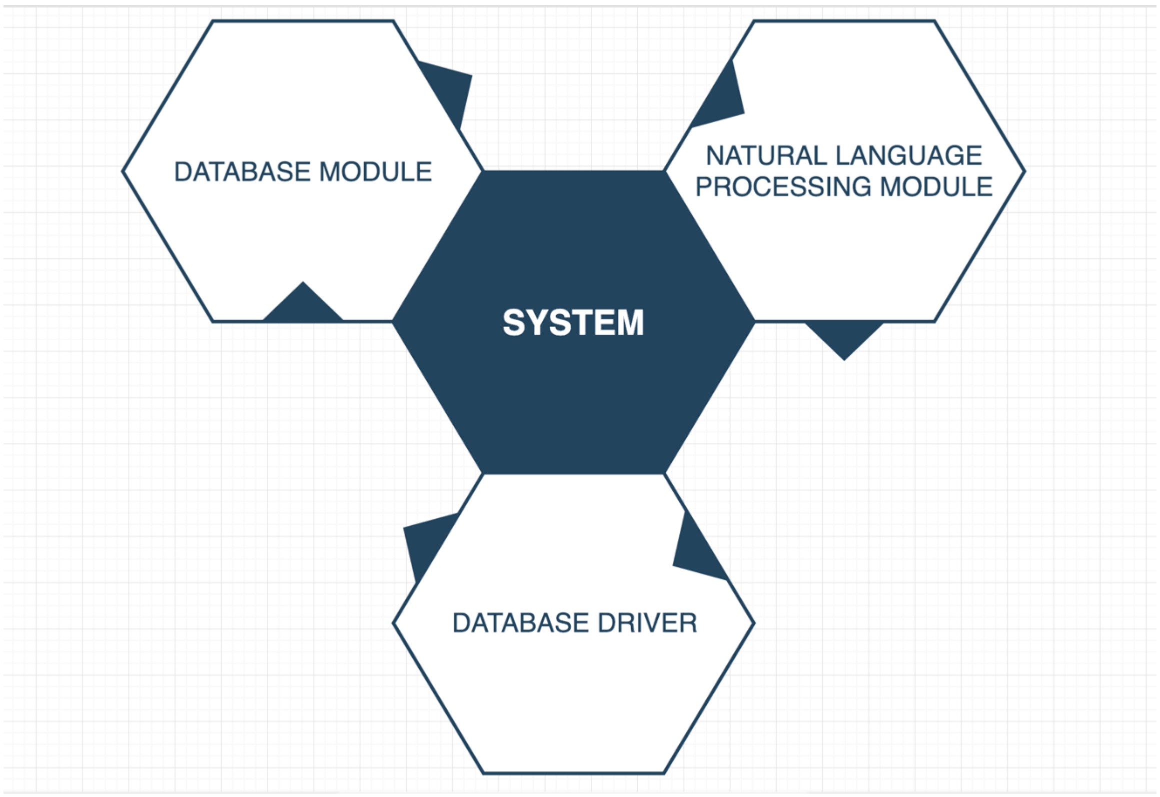


Figure 5-2: Top level design architecture of the proposed system

5.5 Module architecture

5.5.1 Database Module / Database Driver

For the database, the connection tool uses a module called PyMysql this package contains a pure-Python client library this driver provides transparent connectivity to the database. Once database connection established, using connection module tool read define database and re-write rules inside JSON file for make it easier to identify table structure and relationship between tables.

5.5.2 Natural Language Processing Module

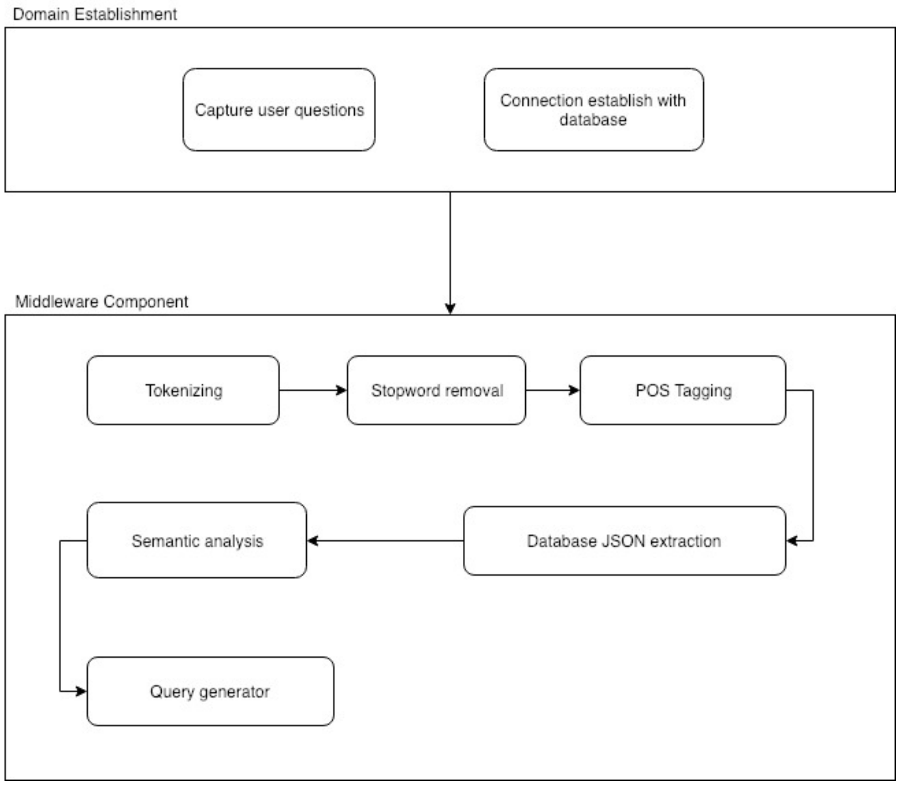


Figure 5-3: Modules contain in the projects

5.5.2.1 Domain Establishment

This module is responsible for getting input as natural language and identifying database model and creating dynamic rules of database tables structure and relations. The process is to recover the architecture of the database on which is going to perform the queries and this in order to know the entities such as columns, tables, primary keys, foreign keys and relationship among tables.

5.5.2.2 Middleware Component

This module is responsible for translating natural language input into a logical query. In this, the sentence is tokenizing and semantically analyzed and processed, and an intermediate query is generated by the following steps.

In this stage of the process, each keyword of the request entered by the user is extracted. The idea now is to look for a correspondence between the keywords of the sentence and the entities of the database to perform segmentation according to the correspondences found and know the best structure of the query to generate. All words are put in lowercase. Each keyword found is tagged according to whether it is a column or a table the database being queried, if a question contains no words similar to a table, it will necessarily be invalid and the message will pop indicating notable mentioned.

Then other words which are tagged used to identify the type of the question which has been asked by the user. Like counting, algebraic calculations, of a negation, etc., if a word referring to the count, example "how many" is found in the first segment of the sentence, the one corresponding to the SELECT, the system identifies the request to be generated as being a count request, that is to say SELECT COUNT (*)

Then it looks for the relationship exists, when it comes to joins Two types of inner joins exist, implicit and explicit. System check for is there any other targeted columns in the tagged keywords by going through rules which generated from studying database.

To fulfill those steps we are using below modules.

5.5.2.3 Stopword removal

Stop words are non-context bearing words, also known as noisy words which are to be excluded from the input sentence to speed up the process. For example, again, already, amongst.

5.5.2.4 Tokenizing

Using word tokenizing split sentence into words and identified tokens can be represented as attribute token, value token, core token, multi-token, continuous token.

5.5.2.5 POS Tagging

The part of speech how a word is used in a sentence. This reads the text in a language and assigns parts of speech to each word (and another token), such as noun, verb, adjective.

Noun (N)- Daniel, London, table, dog, teacher, pen, city, happiness, hope

Verb (V)- go, speak, run, eat, play, live, walk, have, like, are, is

Adjective(ADJ)- big, happy, green, young, fun, crazy, three

Adverb(ADV)- slowly, quietly, very, always, never, too, well, tomorrow

Proposition (P)- at, on, in, from, with, near, between, about, under

Conjunction (CON)- and, or, but, because, so, yet, unless, since, if

Pronoun(PRO)- I, you, we, they, he, she, it, me, us, them, him, her, this

Interjection (INT)- Ouch! Wow! Great! Help! Oh! Hey! Hi!

5.5.2.6 Semantic Analysis

Semantic Analysis is related to represent the meaning of linguistics sentences. It concerns how to determine and understand the meaning of each word. So, it is responsible for creating the logical query which acts as the input query to Database Query Generator. Hence this is another form of presenting the user tokens in the form of the semantic word and matching identified words with the database module.

5.6 Dependency management ground up to work on a generated prototype

For the proposed tool we have used several techniques to make development clean and reduce the time taken to some different tasks like for the testing we have used an automated testing library called unit test.

5.6.1 Version controlling

For the development of the proposed solution tool, we have used version controlling such as GitHub, it makes other developers also to contribute to the tool and contribute to future developments.

5.6.2 Dependency management

We have used Pipenv as the dependency manager. Because this is work as a middleware tool using dependency manager make it easier to set up on another environments as well. It bundles up all the required dependencies which required to run the tool.

5.6.3 Testing

For the making developing and output of the system faster and ensure its correctness we have integrated automated testing.

5.7 Summary

In this chapter, a detailed description of the system design has been discussed. Usage of different technologies was also mentioned in this chapter. In the next chapter, we will be discussing the implementation of the proposed system according to the designs which we have discussed.

Chapter 6

6. Implementation

6.1 Introduction

In the previous chapter discussed the full top-level design including modules separately and system architecture of the proposed solution. In this chapter describes the Implementation details of each module of our solution implementation. Proposed solution includes several major modules which used to identify natural language questions and process them as a structured query language query will be discussing on the implementation of the proposed system.

6.2 Overall solution

The overall solution has been developing an open source application tool which can be accessed by any environment. OS like Windows, Linux or MacOS since build prototype going to deploy on the cloud-based environment. This tool based on client-side architecture and it going to support multi-database driver as well as reporting generating with a bit more development.

6.3 Implementation of the Solution

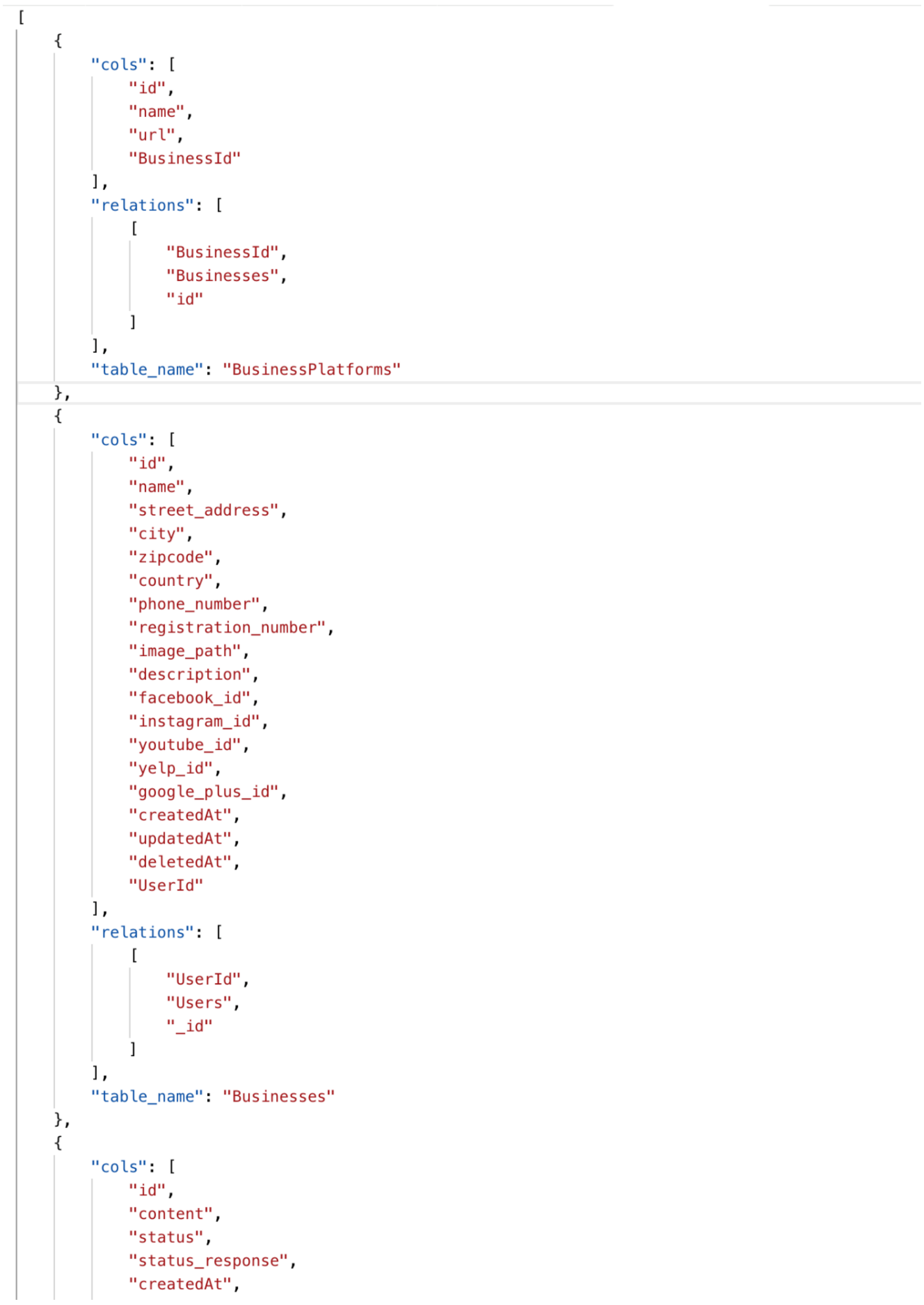
The proposed solution consists of several main modules. Each module using a different type of technologies mentioned in chapter 3. The final outcome of this proposed solution is to convert natural language question into a structured query language. Because of this, any non-technical user can retrieve data from a database without having prior knowledge of structured query language. With include of database driver output result query can retrieve data from any database language like MySQL, SQL, Postgres.

6.3.1 Programming

For cording the project we have used the Mircosoft Visual Code which provides a set of comprehensive tools to work with open source projects. Since it is run on any platform able to work on many platforms. The proposed tool was implemented using python 3 because it has a number of natural language processing toolkit which makes easier to process some tasks. Comparing to other programming language Python is mostly support language for natural language processing. For data manipulation, on the database we have used PHPMyAdmin which provides support for MySql databases. It provides a better interface to interact with databased. For Running MySql server we have used MAMP server.

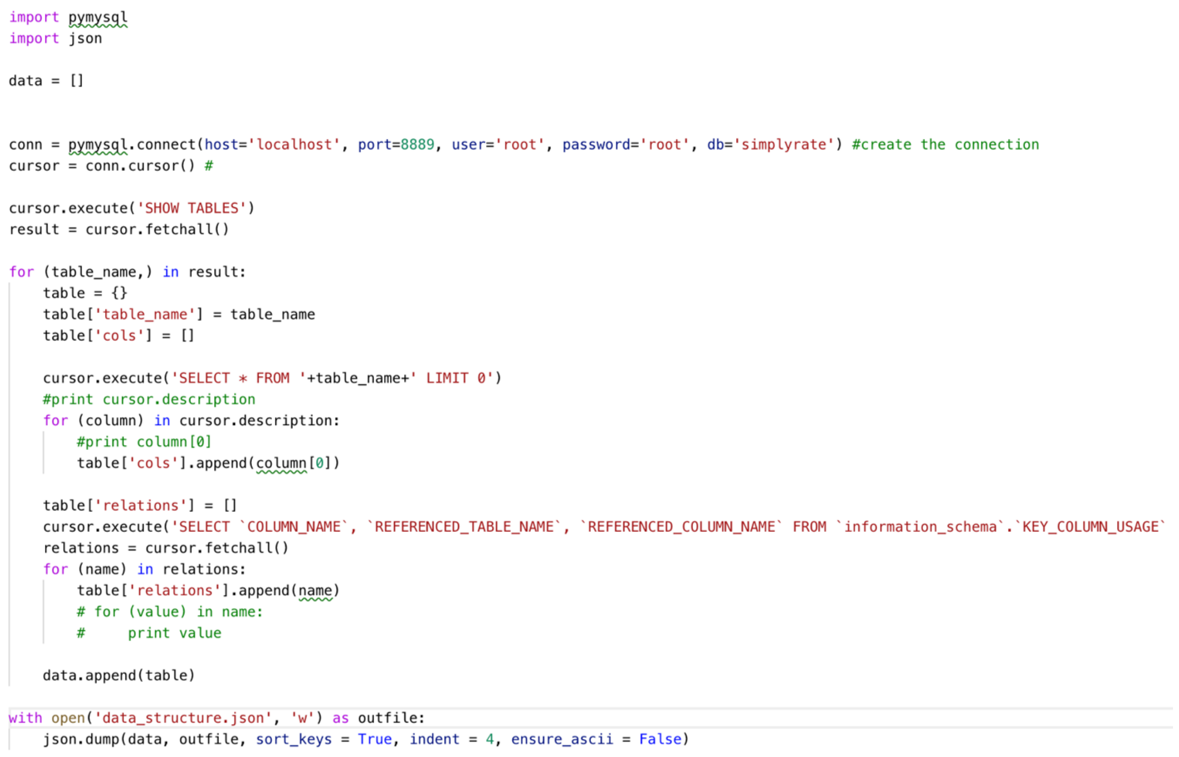
6.3.2 Database structure JSON Implementation

As shown in the following code snippet, it shows sample rules which have been generated from the database to make it easier to understand table structure and relationship between tables. These schema files are written in JavaScript Object Notation (JSON) file because it is a more lightweight format that is used to data interchange. The advantage of using this format is connected databases can be many capacities using this kind of lightweight data interchanging format will not affect performance wise. These written rules are dynamic because of that it will get change according to database changes and able to work any connected databases since these are not static rules.



```
[
    {
        "cols": [
            "id",
            "name",
            "url",
            "BusinessId"
        ],
        "relations": [
            [
                "BusinessId",
                "Businesses",
                "id"
            ]
        ],
        "table_name": "BusinessPlatforms"
    },
    {
        "cols": [
            "id",
            "name",
            "street_address",
            "city",
            "zipcode",
            "country",
            "phone_number",
            "registration_number",
            "image_path",
            "description",
            "facebook_id",
            "instagram_id",
            "youtube_id",
            "yelp_id",
            "google_plus_id",
            "createdAt",
            "updatedAt",
            "deletedAt",
            "UserId"
        ],
        "relations": [
            [
                "UserId",
                "Users",
                "_id"
            ]
        ],
        "table_name": "Businesses"
    },
    {
        "cols": [
            "id",
            "content",
            "status",
            "status_response",
            "createdAt",
```

Figure 6-1: JSON table structure

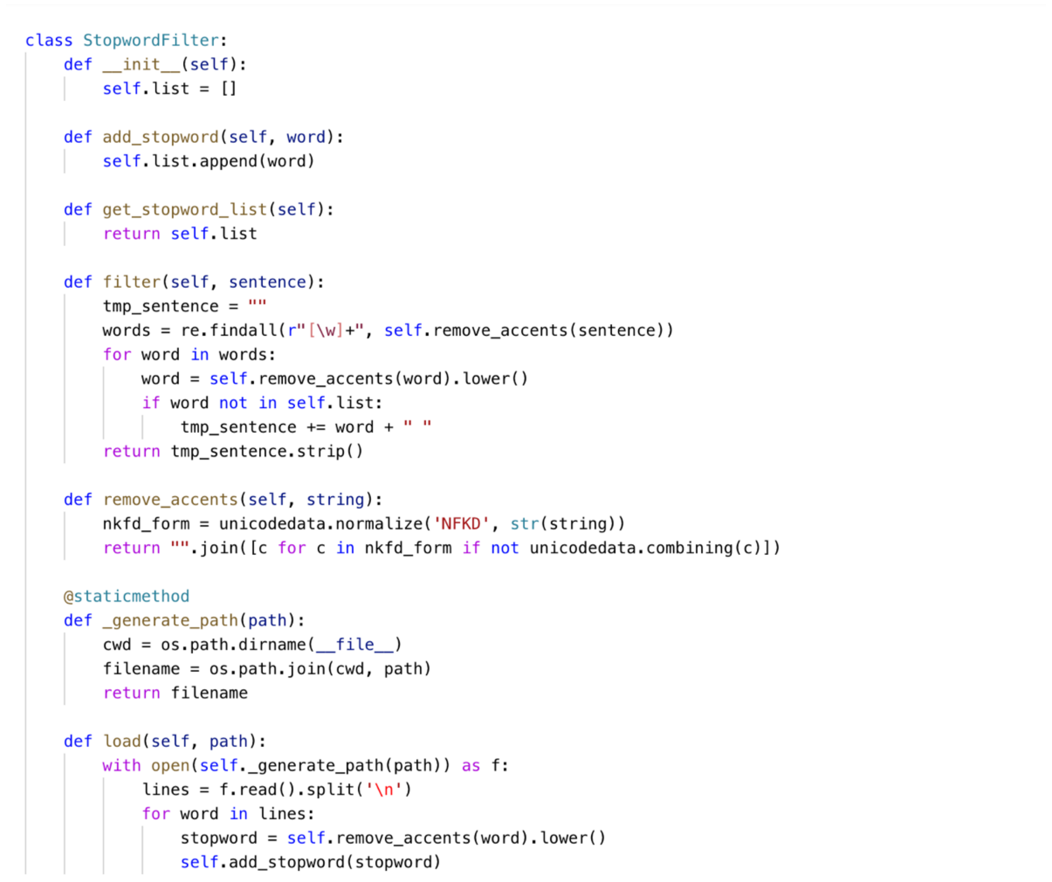


```python
import pymysql
import json

data = []


conn = pymysql.connect(host='localhost', port=8889, user='root', password='root', db='simplyrate') #create the connection
cursor = conn.cursor() #

cursor.execute('SHOW TABLES')
result = cursor.fetchall()

for (table_name,) in result:
    table = {}
    table['table_name'] = table_name
    table['cols'] = []

    cursor.execute('SELECT * FROM '+table_name+' LIMIT 0')
    #print cursor.description
    for (column) in cursor.description:
        #print column[0]
        table['cols'].append(column[0])

    table['relations'] = []
    cursor.execute('SELECT `COLUMN_NAME`, `REFERENCED_TABLE_NAME`, `REFERENCED_COLUMN_NAME` FROM `information_schema`.`KEY_COLUMN_USAGE`
    relations = cursor.fetchall()
    for (name) in relations:
        table['relations'].append(name)
        # for (value) in name:
        #     print value

    data.append(table)

with open('data_structure.json', 'w') as outfile:
    json.dump(data, outfile, sort_keys = True, indent = 4, ensure_ascii = False)
```

Figure 6-2: Database connection

6.4 Stopword filter

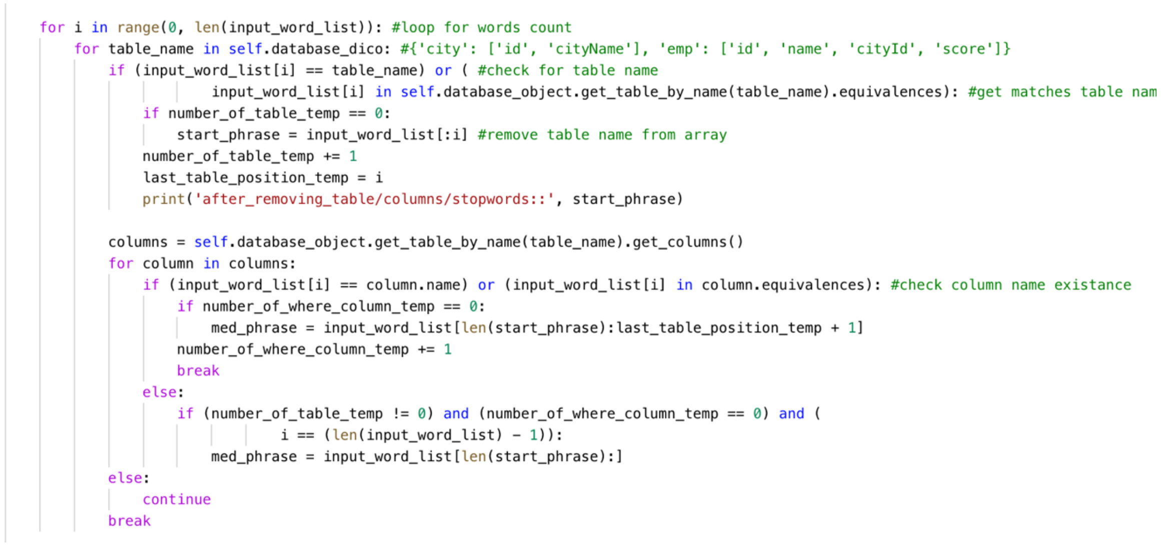
From below code snippet identified unwanted words in the sentence which entered by the user. These words also called as a noise in the text. Such as is, am, are, this, a, an, the. By removing those words it makes easier to tool more focus on keywords such as table names, column names, and conditions.



```python
class StopwordFilter:
    def __init__(self):
        self.list = []

    def add_stopword(self, word):
        self.list.append(word)

    def get_stopword_list(self):
        return self.list

    def filter(self, sentence):
        tmp_sentence = ""
        words = re.findall(r"[\w]+", self.remove_accents(sentence))
        for word in words:
            word = self.remove_accents(word).lower()
            if word not in self.list:
                tmp_sentence += word + " "
        return tmp_sentence.strip()

    def remove_accents(self, string):
        nkfd_form = unicodedata.normalize('NFKD', str(string))
        return "".join([c for c in nkfd_form if not unicodedata.combining(c)])

    @staticmethod
    def _generate_path(path):
        cwd = os.path.dirname(__file__)
        filename = os.path.join(cwd, path)
        return filename

    def load(self, path):
        with open(self._generate_path(path)) as f:
            lines = f.read().split('\n')
            for word in lines:
                stopword = self.remove_accents(word).lower()
                self.add_stopword(stopword)
```

Figure 6-3: Stopword remove algorithm

Below code snippet shows how to identify table names and column names from matching with JSON database rules.



```python
for i in range(0, len(input_word_list)): #loop for words count
    for table_name in self.database_dico: #{'city': ['id', 'cityName'], 'emp': ['id', 'name', 'cityId', 'score']}
        if (input_word_list[i] == table_name) or ( #check for table name
                input_word_list[i] in self.database_object.get_table_by_name(table_name).equivalences): #get matches table nam
            if number_of_table_temp == 0:
                start_phrase = input_word_list[:i] #remove table name from array
            number_of_table_temp += 1
            last_table_position_temp = i
            print('after_removing_table/columns/stopwords::', start_phrase)

        columns = self.database_object.get_table_by_name(table_name).get_columns()
        for column in columns:
            if (input_word_list[i] == column.name) or (input_word_list[i] in column.equivalences): #check column name existance
                if number_of_where_column_temp == 0:
                    med_phrase = input_word_list[len(start_phrase):last_table_position_temp + 1]
                number_of_where_column_temp += 1
                break
            else:
                if (number_of_table_temp != 0) and (number_of_where_column_temp == 0) and (
                        i == (len(input_word_list) - 1)):
                    med_phrase = input_word_list[len(start_phrase):]
        else:
            continue
        break
```

Figure 6-4: Table identification

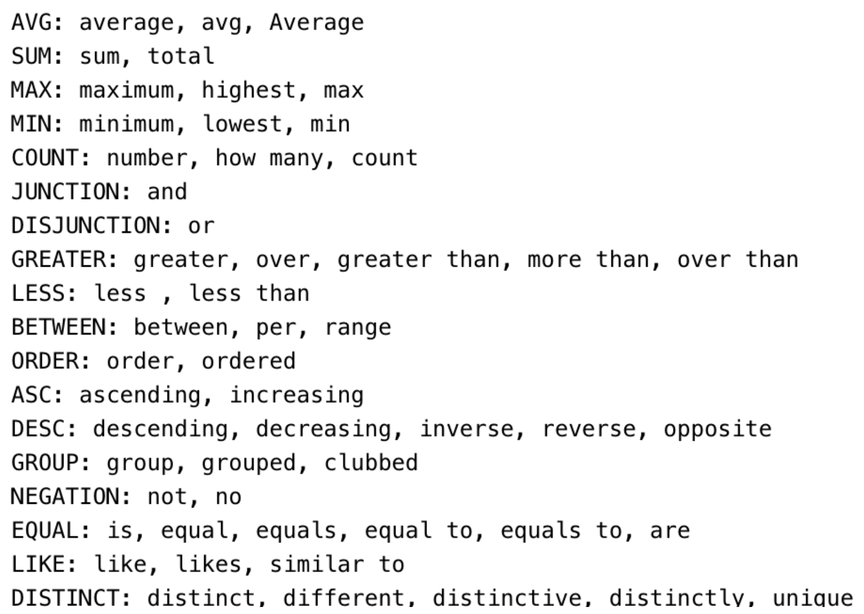
6.5 Extraction of words.

Once stopword removal and sentence split into word (tokenizing) is done, Application will start to identify meaningful sentence by matching between certain concepts entered by the user and the element of the database. To do this we first need to identify empty words and remove them as in below sentence

Display all **emp** from the **cityName** is **Ratnapura**

Once filter identifies meaningful words it only should return the elements "display, emp, cityName, Rathnapura."

As the second step is to match with database structure JSON file the meaningful words which we are going to perform the queries and this is where the order to know the entities table names, columns, primary and foreign keys to identify the relationship between tables



```
AVG: average, avg, Average
SUM: sum, total
MAX: maximum, highest, max
MIN: minimum, lowest, min
COUNT: number, how many, count
JUNCTION: and
DISJUNCTION: or
GREATER: greater, over, greater than, more than, over than
LESS: less , less than
BETWEEN: between, per, range
ORDER: order, ordered
ASC: ascending, increasing
DESC: descending, decreasing, inverse, reverse, opposite
GROUP: group, grouped, clubbed
NEGATION: not, no
EQUAL: is, equal, equals, equal to, equals to, are
LIKE: like, likes, similar to
DISTINCT: distinct, different, distinctive, distinctly, unique
```

Figure 6-5: Word bag

At this stage of the process, each keyword of the request entered by the user is extracted. The application also contains a list of synonyms for each of these keywords. The idea is now to find a match between the keywords of the application and the entities of the database in order to realize a segmentation of the request according to the correspondences found and to know the structure better and the request to generate. At the time of the correspondence, all the words are written in lowercase letters and all the diacritical characters are normalized. list of synonyms for each of keywords can be improved by editing English.csv file.

6.6 Join Query Handling

In joining we treat only the inner joins (INNER JOIN). Two types of inner joins exist, implicit and explicit. When there is a selection or a constraint on a column that is not part of the FROM table, when the table to which the targeted column belongs is not mentioned in the sentencing entry is an implicit join.

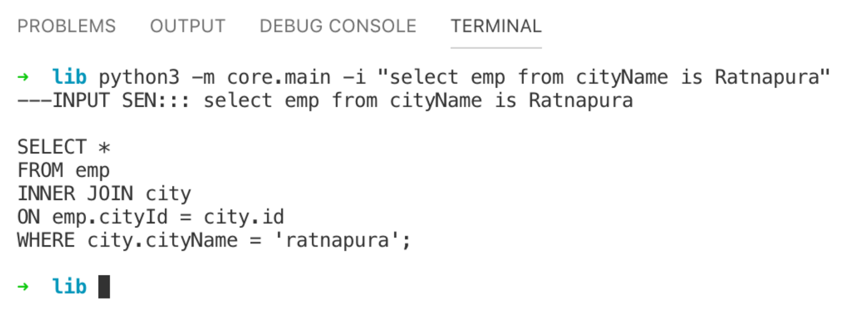
In this case, make a join between the table of the target column and the table FROM which is specified in the sentence.

In the case of an explicit join, the table on which we must do join is specified directly in the sentence,

as in the example: "Which students have a teacher who the first name is Ravi? "

Here we must make a join between the table high and prof the teacher, in order to be able to select students by making a constraint on the first names of the teachers. If the column of the selection or constraint is not found in the FROM table, or in a joinable table from the FROM table, then the query is impossible to build. Because of knowing the primary and foreign keys of each table can implicitly deduce the actual links between the tables and therefore knows if one table can be connected to another and if so, by which tables pass. Indeed, the Proposed tool can create joins through more than one table if required.

6.7 Generated sample user output



```
→  lib python3 -m core.main -i "select emp from cityName is Ratnapura"
---INPUT SEN::: select emp from cityName is Ratnapura

SELECT *
FROM emp
INNER JOIN city
ON emp.cityId = city.id
WHERE city.cityName = 'ratnapura';

→  lib
```

Figure 6-6: System output

This figure shows once user input natural language question to the proposed tool how it generates output as a structured query

6.8 Summary

This chapter discussed the implementation of the proposed solution tool. It described the implementation of the major modules that it has and about the functionalities which each module is equipped with. Next chapter will describe how the evaluation was done of the implemented system and the details of whether the objectives have been achieved using test cases. Further, it'll discuss on drawbacks and limitations of the implemented system.

Chapter 7

7. Evaluation of the solution

7.1 Introduction

In this chapter, we will discuss the correctness of the proposed tool by evaluating it through test cases. Doing test cases we expect to identify whether it satisfied given requirements and mentioned aspects such as functionalities, reliability, usability, efficiency, and maintainability. For better identification, we hope to run human testing since this tool more focusing on non-technical users. Running on several test cases we hope to identify any gaps, error or any missing requirements.

7.2 Participants

Venuerific Pvt. Ltd

Venuerific is a market that connects people who want to organize an event (seminar, meeting, birthday, wedding, etc.) with people with appropriate spaces. Spaces may vary from commercial businesses such as bar, restaurant, cafes, hotels, function rooms to non-commercial activities such as private estates, lofts, warehouses, islands, and others. Apart from that, Venuerific has also evolved into a one-stop shop for people who want to grow their business through events. We do this by helping them create engaging and engaging content that unlock the real value of their assets. Our vision is to create unique and seamless experiences for all those who wish to organize a special occasion or celebration.

The main participant from this company for the evolution of the proposed tool was a team of 5 non-technical users. We have asked them to retrieve the information which required for decision making for their day to day tasks by connecting this tool to their commercial database/production database.

Test Data	Expected Results	Actual Results	Pass / Fail
Number of venues there are ordered by venue_name in descending and ordered by type?	SELECT COUNT FROM venues ORDER BY venue_name DESC, type ASC	SELECT COUNT FROM venues ORDER BY venue_name DESC, type ASC	Pass
count how many venues there are ordered by name	SELECT COUNT(*) FROM venues ORDER BYvenue_name ASC	SELECT COUNT(*) FROM venues ORDER BY name ASC	Fail
Count the number of users	SELECT COUNT(*) FROM users	SELECT COUNT(*) FROM users	Pass
Counts the names of the users whose names are Adam and the age is 25	SELECT COUNT(users.names) FROM users WHERE users.name = 'Adam' AND users.age = '25'	SELECT COUNT(users.names) FROM users WHERE users.name = 'Adam' AND users.age = '25'	Pass
List all users	SELECT * FROM USERS	SELECT * FROM USERS	Pass
Show customers from city is boogies	SELECT * FROM customers WHERE city = 'boogies';	SELECT * FROM customers WHERE city = 'boogies';	Pass
Display customer from city is rocher	SELECT * FROM customers WHERE city = 'rocher';	Error no table name found	Fail
Display maximum budget, the name of all package and package details	SELECT MAX(budget), name FROM package INNER JOIN package_details ON package.id = package-details.pid;	SELECT MAX(budget), name FROM package INNER JOIN package_details ON package.id = package details.pid;	Fail
Display name, booking Name of all bookings and customer	SELECT customer.name, city.cityName FROM city INNER JOIN emp ON city.id = emp.cityId;	SELECT emp.name FROM city INNER JOIN emp ON city.id = emp.cityId;	Fail
Display bookingName	SELECT booking. bookingName	SELECT booking. bookingName	Pass

of all customers and booking	FROM customers INNER JOIN booking ON booking.id = customers.bid;	FROM customers INNER JOIN booking ON booking.id = customers.bid;	
distince name from customers	SELECT DISTINCT customers.name FROM customers;	SELECT customers.name FROM customers;	Fail
distinct name from customers	SELECT DISTINCT customers.name FROM customers;	SELECT DISTINCT customers.name FROM customers;	Pass
List all promotions	SELECT * FROM promotions	SELECT * FROM promotions	Pass
List all events	SELECT * FROM events	Error no table name found	Fail
Show me all event_messages where read is true	SELECT * FROM event_messages WHERE read = ‘t’	SELECT * FROM event_messages WHERE read = ‘true’	Fail
View all review with venue name	SELECT reviews.comment, venues.name FROM reviews INNER JOIN reviews.venue_id = venues.id	SELECT reviews.comment, venues.name FROM reviews INNER JOIN reviews.venue_id = venues.id	Pass
What is the number of venues in this database?	SELECT COUNT(*) FROM venues	SELECT COUNT(*) FROM venues	Pass

Table 7-1: Test cases

DEKACH Pvt. Ltd

Dekach is medium scale import and export company founded in early 1998, They are using a custom build ERP solution application to maintain inventory by adding purchase orders, issuing quotations and invoices to customers.

A group of five marketing coordinators was used to check the validity of the result generated from the proposed tool.

Test Data	Expected Results	Actual Results	Pass / Fail
List all invoice	SELECT * FROM invoices	Error no table name found	Fail
Display employee from city is Kohuwala	SELECT * FROM emp INNER JOIN city ON emp.cityId = city.id WHERE city.city = 'Kohuwala';	Error no table name found	Fail
Show me all quotations	SELECT * FROM quotations	SELECT * FROM quotations	pass
Display customer from invoices where invoice_number is A001	SELECT * FROM invoices INNER JOIN customer ON customer.Iid = invoices.id WHERE invoices.invoice_name = 'A001';	SELECT * FROM invoices INNER JOIN customer ON customer.Iid = invoices.id WHERE invoices.invoice_name = 'A001';	Pass
Show data for a city where the name is not nugegoda and id like 2	SELECT * FROM city WHERE city.name != 'nugegoda' AND city.id LIKE '%2%';	SELECT * FROM city WHERE city.name != 'nugegoda' AND city.id LIKE '%2%';	Pass
List all payments	SELECT * FROM payments	SELECT * FROM payments	Pass
Display all accounts	SELECT * FROM accounts	SELECT * FROM accounts	Pass
View all logs	SELECT * FROM logs	SELECT * FROM logs	Pass
What is the number of payments in this database?	SELECT COUNT(*) FROM payments	SELECT COUNT(*) FROM payments	Pass
count distinctly how many invoices there are ordered by customer_name in descending and ordered by total?	SELECT COUNT(*) FROM customers INNER JOIN invoices ON customer.id = invoices.cid ORDER BY customer_name DESC, total ASC	SELECT COUNT(*) FROM customers INNER JOIN invoices ON customer.id = invoices.cid ORDER BY customer_name DESC, total ASC	Pass
count how many emails there are	SELECT COUNT(*) FROM emails INNER JOIN customers ON	SELECT COUNT(*) FROM emails INNER JOIN customers ON	Pass

ordered by the customer	emails.id = customers.eid ORDER BY customer.name ASC	emails.id = customers.eid ORDER BY customer.name ASC	
Count the number of purchase orders	SELECT COUNT(*) FROM pOrders	SELECT COUNT(*) FROM purchase orders	Fail
Counts the names of the users whose names are Thushara and the status is active	SELECT COUNT(names) FROM student WHERE users.name = 'Thushara' AND student.status = ‘active’	SELECT COUNT(names) FROM student WHERE users.name = 'Thushara' AND student.status = ‘active’	Pass
Display name, log time of all users and log	SELECT emp.name, log.logTime FROM users INNER JOIN log ON users.id = log.uid;	SELECT emp.name, log.logTime FROM users INNER JOIN log ON users.id = log.uid;	Pass
distince name from emp	SELECT DISTINCT emp.name FROM emp;	SELECT emp.name FROM emp;	Fail
Display maximum number, name, invoice of all customer and number	SELECT MAX(number), customers.name, invoice.number FROM customers INNER JOIN invoice ON invoice.id = customer.cid;	SELECT MAX(number), customer.name, invoice.number FROM customers INNER JOIN invoice ON invoice.id = customer.cid;	Fail

Table 7-2: Test cases

Redot Pvt. Ltd

Redot Pvt Ltd was founded in 2012 in Sri Lanka and is an IT company that provides services to its customers in the areas of web and mobile applications. It serves SMEs around the world and satisfies customers around the world. Since most customers come from Singapore, it has already been registered in Singapore. The services provided include the development, design, integration, and maintenance of web services, mobile applications, search engine optimization, security and networking, and graphics.

The main participant in the project evaluation was a team of 10 QA engineers from Redot Pvt Ltd, All participants have between 2 and 8 years of experience.

Test Data	**Expected Results**	**Actual Results**	**Pass / Fail**
List all client	SELECT * FROM CLIENT	SELECT * FROM CLIENT	Pass
Display all client	SELECT * FROM CLIENT	SELECT * FROM CLIENT	Pass
View all client	SELECT * FROM CLIENT	SELECT * FROM CLIENT	Pass
Show me all client	SELECT * FROM CLIENT	SELECT * FROM CLIENT	pass
List all clients	SELECT * FROM CLIENT	Error no table name found	Fail
Display employee from cityName is Ratnapura	SELECT * FROM emp INNER JOIN city ON emp.cityId = city.id WHERE city.cityName = 'ratnapura';	Error no table name found	Fail
Display emp from cityName is Ratnapura	SELECT * FROM emp INNER JOIN city ON emp.cityId = city.id WHERE city.cityName = 'ratnapura';	SELECT * FROM emp INNER JOIN city ON emp.cityId = city.id WHERE city.cityName = 'ratnapura';	Pass
Display name, cityName of all emp and city	SELECT emp.name, city.cityName FROM city INNER JOIN emp ON city.id = emp.cityId;	SELECT emp.name, city.cityName FROM city INNER JOIN emp ON city.id = emp.cityId;	Pass
Display name, city Name of all emp and city	SELECT emp.name, city.cityName FROM city INNER JOIN emp ON city.id = emp.cityId;	SELECT emp.name FROM city INNER JOIN emp ON city.id = emp.cityId;	Fail
Display maximum score, name, cityName of	SELECT MAX(emp.score), emp.name, city.cityName FROM city	SELECT MAX(emp.score), emp.name, city.cityName FROM city	Pass

all emp and city	INNER JOIN emp ON city.id = emp.cityId;	INNER JOIN emp ON city.id = emp.cityId;	
Count the number of students	SELECT COUNT(*) FROM Students	SELECT COUNT(*) FROM Students	Pass
Counts the names of the students whose names are Kamal and the age is 25	SELECT COUNT(student.names) FROM student WHERE student.name = 'kamal' AND student.age = '25'	SELECT COUNT(student.names) FROM student WHERE student.name = 'kamal' AND student.age = '25'	Pass
What is the number of the city in this database?	SELECT COUNT(*) FROM city	SELECT COUNT(*) FROM city	Pass
Show data for the city where cityName is not Pune and id like 1	SELECT * FROM city WHERE city.cityName != 'pune' AND city.id LIKE '%1%';	SELECT * FROM city WHERE city.cityName != 'pune' AND city.id LIKE '%1%';	Pass
distince name from emp	SELECT DISTINCT emp.name FROM emp;	SELECT emp.name FROM emp;	Fail
distinct name from emp	SELECT DISTINCT emp.name FROM emp;	SELECT DISTINCT emp.name FROM emp;	Pass
count distinctly how many cities there are ordered by name in descending and ordered by score?	SELECT COUNT(*) FROM city INNER JOIN emp ON city.id = emp.cityId ORDER BY emp.name DESC, emp.score ASC	SELECT COUNT(*) FROM city INNER JOIN emp ON city.id = emp.cityId ORDER BY emp.name DESC, emp.score ASC	Pass
What are the address and phone number of the customer whose name is Mark and whose age is greater than 14 grouped by address?	SELECT client.adresse, client.telephone FROM client WHERE client.nom = 'marc' AND client.age > '14' GROUP BY client.adresse	SELECT client.adresse, client.telephone FROM client WHERE client.nom = 'marc' AND client.age > '14' GROUP BY client.adresse	Pass

count how many cities there are ordered by name	SELECT COUNT(*) FROM city INNER JOIN emp ON city.id = emp.cityId ORDER BY emp.name ASC	SELECT COUNT(*) FROM city INNER JOIN emp ON city.id = emp.cityId ORDER BY emp.name ASC	Pass

Table 7-3: Test cases

7.3 Summary

This chapter discussed the evolutions and the testing of the proposed system, Participants for the testing and some test cases done by participants.

Chapter 8

8. Conclusion

8.1 Introduction

In this chapter, we will discuss the further improvements can be done to tune-up this proposed solution tool based on user reviews and the limitations faced when developing this tool.

8.2 Conclusion

Building this proposed solution tool was able to successfully complete. By looking at the test cases we able to understand non-technical users able to meet the expected results.

When gone through this development process we able to master a new language like python and learn some new technologies.

8.3 Limitations

When we were developing this proposed tool, able to identify some limitations like the user has to enter the exact table when asking the questions. Because when system development process developer can use any name for tables like. Let’s think developer creating employee table he can rename it as emp or employee because of that when processing natural language question it impossible to identify the table or column name without having the exact name. Another example in WordPress when table creating it use prefix wp_ for every table start.

8.4 Future works

Currently, this proposed tool able generate queries according to user questions, in future developments we are planning to extend it to execute generated query and display results as well and give support to export generated results as in PDF format or excel formats.

Another improvement planned on this is that the connect this tool to frontend UI interface which helps the user to enter questions with auto-suggestion help.

8.5 Summary

This chapter provides the conclusion of the overall solution achieved by the Research project named "Natural Language Based Report Generator" and this described the faced limitations and future works planned on this proposed solution tool.

9. Reference

[1] “The Stanford Natural Language Processing Group.” [Online]. Available: https://nlp.stanford.edu/software/pos-tagger-faq.html. [Accessed: 19-Nov-2017].

[2] “JSON.” [Online]. Available: https://www.json.org/. [Accessed: 16-Feb-2019].

[3] “Welcome to Python.org,” *Python.org*. [Online]. Available: https://www.python.org/doc/. [Accessed: 16-Feb-2019].

[4] *NLIDB: Natural Language Interface to DataBases*. DukeNLIDB, 2017.

[5] “Natural Language Interface for Databases,” *KUERI.ME*. [Online]. Available: http://kueri.me/. [Accessed: 13-Jul-2018].

[6] "query: A python framework to transform natural language questions to queries in a database query language," 14-Nov-2017. [Online]. Available: https://github.com/machinalis/quepy. [Accessed: 15-Nov-2017].

[7] “What is CRUD? | Codecademy.” [Online]. Available: https://www.codecademy.com/articles/what-is-crud. [Accessed: 16-Feb-2019].

[8] “What is SQL (Structured Query Language)? - Definition from WhatIs.com,” *SearchSQLServer*. [Online]. Available: http://searchsqlserver.techtarget.com/definition/SQL. [Accessed: 19-Nov-2017].

[9] W. A. Woods, R. Kaplan, and B. Webber, “The Lunar Sciences Natural Language Information System,” Jul. 1972.

[10] G. Rao, C. Agarwal, S. Chaudhry, N. Kulkarni, and D. S. Patil, “Natural language query processing using semantic grammar,” *International Journal on computer science and engineering*, vol. 2, no. 2, pp. 219–223, 2010.

[11] N. Stratica, L. Kosseim, and B. C. Desai, “Using semantic templates for a natural language interface to the CINDI virtual library,” *Data & Knowledge Engineering*, vol. 55, no. 1, pp. 4–19, Oct. 2005.

[12] R. Alexander, P. Rukshan, and S. Mahesan, “Natural Language Web Interface for Database (NLWIDB),” *arXiv preprint arXiv:1308.3830*, 2013.

[13] “Natural Language Toolkit — NLTK 3.3 documentation.” [Online]. Available: https://www.nltk.org/. [Accessed: 13-Jul-2018].

[14] “The Web framework for perfectionists with deadlines | Django.” [Online]. Available: https://www.djangoproject.com/. [Accessed: 16-Feb-2019].

[15] “Welcome | Flask (A Python Microframework).” [Online]. Available: http://flask.pocoo.org/. [Accessed: 16-Feb-2019].

[16] “Bitbucket: What is Bitbucket? - Atlassian Documentation.” [Online]. Available: https://confluence.atlassian.com/confeval/development-tools-evaluator-resources/bitbucket/bitbucket-what-is-bitbucket. [Accessed: 16-Feb-2019].

[17] “Visual Studio Code - Code Editing. Redefined.” [Online]. Available: https://code.visualstudio.com/. [Accessed: 16-Feb-2019].

[18] phpMyAdmin contributors, “phpMyAdmin,” *phpMyAdmin*. [Online]. Available: https://www.phpmyadmin.net/. [Accessed: 16-Feb-2019].

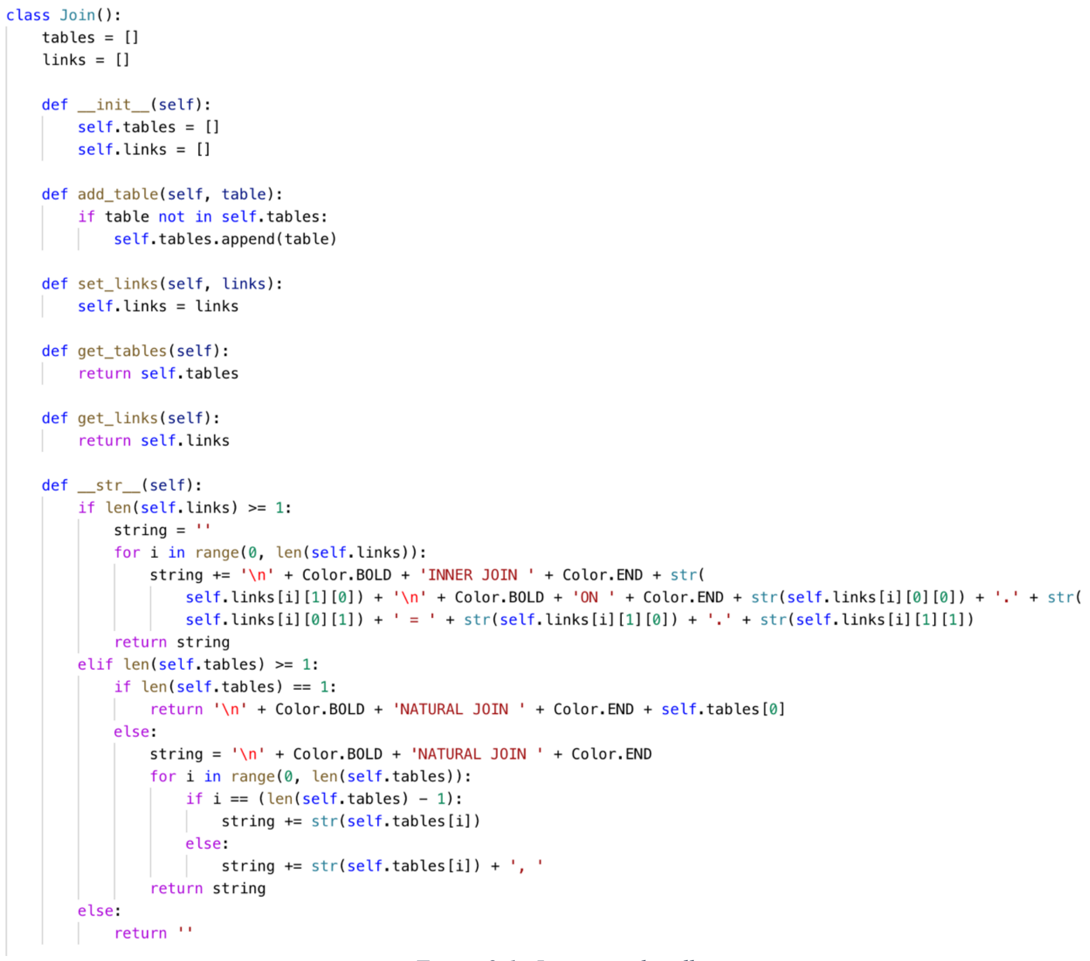
[19] “MAMP & MAMP PRO.” [Online]. Available: https://www.mamp.info/en/. [Accessed: 16-Feb-2019].

[20] “Amazon EC2.” [Online]. Available: https://aws.amazon.com/ec2/. [Accessed: 16-Feb-2019].

[21] “AWS Elastic Beanstalk – Deploy Web Applications.” [Online]. Available: https://aws.amazon.com/elasticbeanstalk/. [Accessed: 16-Feb-2019].

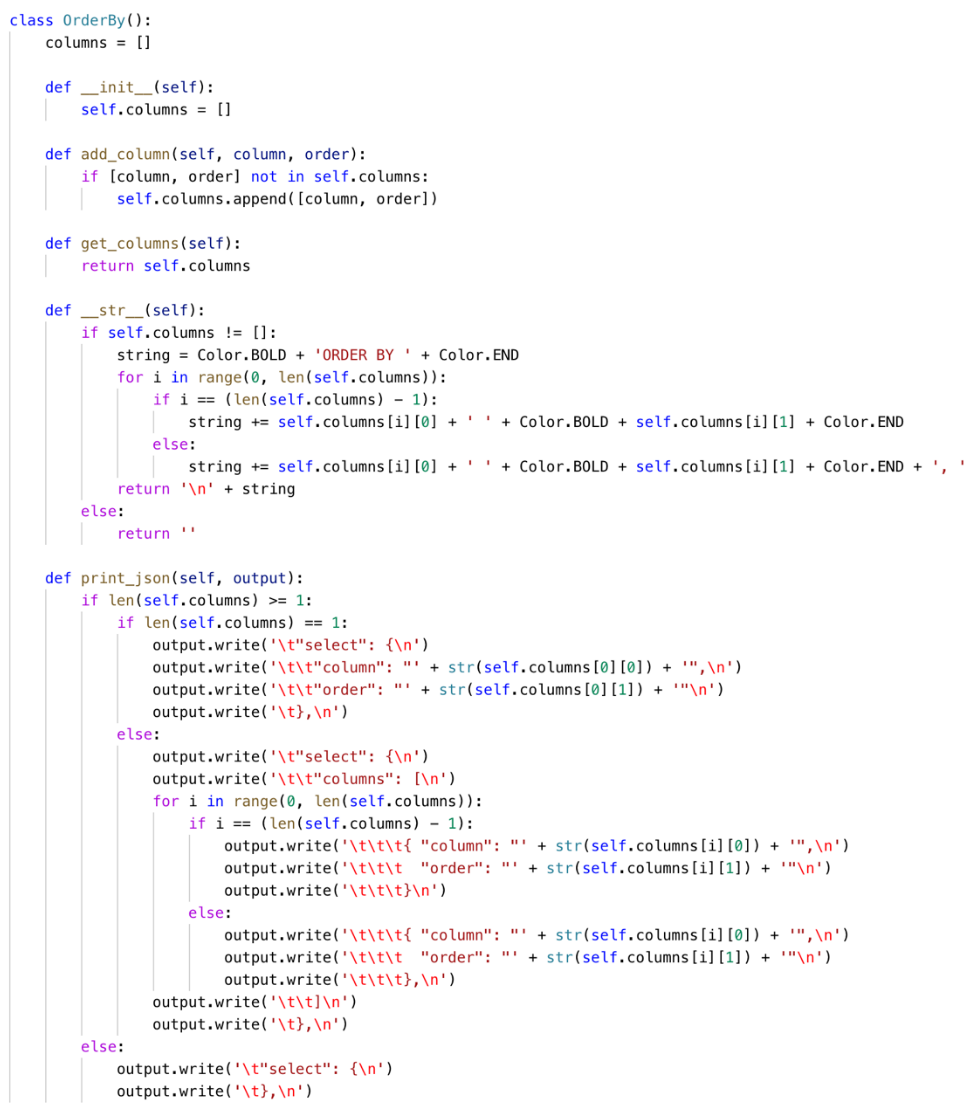
[22] “What is ERP - Enterprise Resource Planning? Webopedia.” [Online]. Available: https://www.webopedia.com/TERM/E/ERP.html. [Accessed: 15-Nov-2017].

Appendix A



```python
class Join():
    tables = []
    links = []

    def __init__(self):
        self.tables = []
        self.links = []

    def add_table(self, table):
        if table not in self.tables:
            self.tables.append(table)

    def set_links(self, links):
        self.links = links

    def get_tables(self):
        return self.tables

    def get_links(self):
        return self.links

    def __str__(self):
        if len(self.links) >= 1:
            string = ''
            for i in range(0, len(self.links)):
                string += '\n' + Color.BOLD + 'INNER JOIN ' + Color.END + str(
                    self.links[i][1][0]) + '\n' + Color.BOLD + 'ON ' + Color.END + str(self.links[i][0][0]) + '.' + str(
                    self.links[i][0][1]) + ' = ' + str(self.links[i][1][0]) + '.' + str(self.links[i][1][1])
            return string
        elif len(self.tables) >= 1:
            if len(self.tables) == 1:
                return '\n' + Color.BOLD + 'NATURAL JOIN ' + Color.END + self.tables[0]
            else:
                string = '\n' + Color.BOLD + 'NATURAL JOIN ' + Color.END
                for i in range(0, len(self.tables)):
                    if i == (len(self.tables) - 1):
                        string += str(self.tables[i])
                    else:
                        string += str(self.tables[i]) + ', '
                return string
        else:
            return ''
```

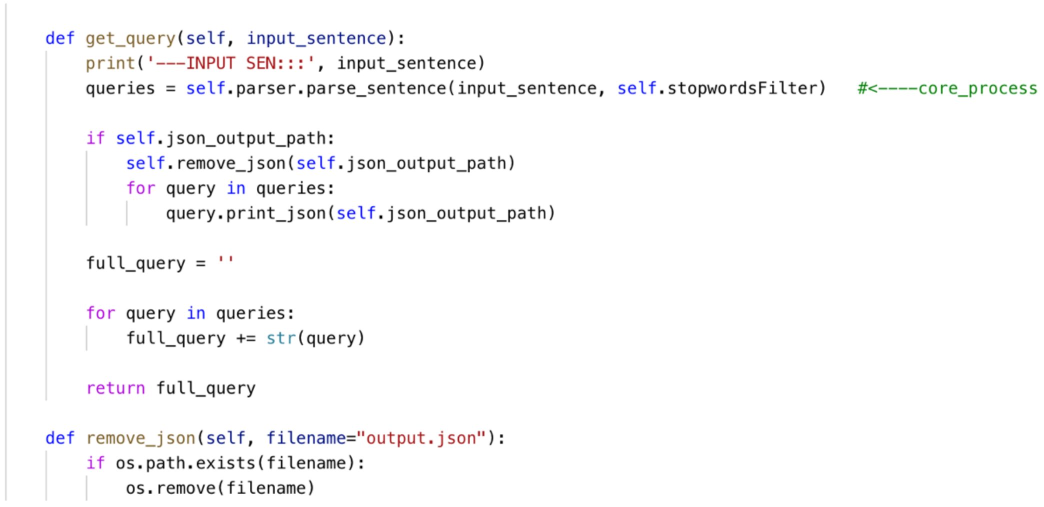
Figure 9-1: Join query handler



```python
class OrderBy():
    columns = []

    def __init__(self):
        self.columns = []

    def add_column(self, column, order):
        if [column, order] not in self.columns:
            self.columns.append([column, order])

    def get_columns(self):
        return self.columns

    def __str__(self):
        if self.columns != []:
            string = Color.BOLD + 'ORDER BY ' + Color.END
            for i in range(0, len(self.columns)):
                if i == (len(self.columns) - 1):
                    string += self.columns[i][0] + ' ' + Color.BOLD + self.columns[i][1] + Color.END
                else:
                    string += self.columns[i][0] + ' ' + Color.BOLD + self.columns[i][1] + Color.END + ', '
            return '\n' + string
        else:
            return ''

    def print_json(self, output):
        if len(self.columns) >= 1:
            if len(self.columns) == 1:
                output.write('\t"select": {\n')
                output.write('\t\t"column": "' + str(self.columns[0][0]) + '",\n')
                output.write('\t\t"order": "' + str(self.columns[0][1]) + '"\n')
                output.write('\t},\n')
            else:
                output.write('\t"select": {\n')
                output.write('\t\t"columns": [\n')
                for i in range(0, len(self.columns)):
                    if i == (len(self.columns) - 1):
                        output.write('\t\t\t{ "column": "' + str(self.columns[i][0]) + '",\n')
                        output.write('\t\t\t  "order": "' + str(self.columns[i][1]) + '"\n')
                        output.write('\t\t\t}\n')
                    else:
                        output.write('\t\t\t{ "column": "' + str(self.columns[i][0]) + '",\n')
                        output.write('\t\t\t  "order": "' + str(self.columns[i][1]) + '"\n')
                        output.write('\t\t\t},\n')
                output.write('\t\t]\n')
                output.write('\t},\n')
        else:
            output.write('\t"select": {\n')
            output.write('\t},\n')
```

Figure 9-2: Order by query handler

Appendix B



```
def get_query(self, input_sentence):
    print('---INPUT SEN:::', input_sentence)
    queries = self.parser.parse_sentence(input_sentence, self.stopwordsFilter)   #<----core_process

    if self.json_output_path:
        self.remove_json(self.json_output_path)
        for query in queries:
            query.print_json(self.json_output_path)

    full_query = ''

    for query in queries:
        full_query += str(query)

    return full_query

def remove_json(self, filename="output.json"):
    if os.path.exists(filename):
        os.remove(filename)
```

Figure 9-1: User input capture code