Natural Language Query to SQL: Dynamically Adaptive System

BY

MEHEDI IMAM SHAFI
ID: 151-15-5248

This Report Presented in Partial Fulfillment of the Requirements for the Degree of Bachelor of Science in Computer Science and Engineering

Supervised By

Mr. Mohammad Mahmudur Rahman
Associate Professor
Department of CSE
Daffodil International University

Co-Supervised By

Mr. Saiful Islam
Lecturer
Department of CSE
Daffodil International University

DAFFODIL INTERNATIONAL UNIVERSITY

DHAKA, BANGLADESH

NOVEMBER, 2018

APPROVAL

This research titled **Natural Language Query to SQL: Dynamically Adaptive System**, submitted by Mehedi Imam Shafi, ID No: 151-15-5248 to the Department of Computer Science and Engineering, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Engineering and approved as to its style and contents. The presentation has been held on 12-12-2018.

BOARD OF EXAMINERS

_____

Dr. Syed Akhter Hossain — **Chairman**
Professor and Head
Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University

_____

Narayan Ranjan Chakraborty — **Internal Examiner**
Assistant Professor
Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University

_____

Md. Tarek Habib — **Internal Examiner**
Assistant Professor
Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University

_____

Dr. Mohammad Shorif Uddin — **External Examiner**
Professor
Department of Computer Science and Engineering
Jahangirnagar University

DECLARATION

I hereby declare that, this project has been done by me under the supervision of **Mohammad Mahmudur Rahman, Associate Professor, Department of CSE** Daffodil International University. I also declare that neither this project nor any part of this project has been submitted elsewhere for award of any degree or diploma.

Supervised by:

_____

Mr. Mohammad Mahmudur Rahman
Associate Professor
Department of CSE
Daffodil International University

Co- Supervised by:

_____

Mr. Saiful Islam
Lecturer
Department of CSE
Daffodil International University

Submitted by:

_____

Mehedi Imam Shafi
ID: 151-15-5248
Department of CSE
Daffodil International University

ACKNOWLEDGEMENT

First I express my heartiest thanks and gratefulness to almighty Allah for His divine blessing makes it possible to complete the final year thesis successfully.

I am really grateful and wish my profound indebtedness to **Mr. Mohammad Mahmudur Rahman**, **Associate Professor**, Department of CSE Daffodil International University, Dhaka. Deep Knowledge & keen interest of my supervisor in the field of *Natural Language Processing* helped me a lot to carry out this research. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

I would like to express my heartiest gratitude to Almighty Allah and Head**,** Department of CSE, for his kind help to finish my project and also to other faculty members and the staffs of CSE department of Daffodil International University.

I would like to thank **Mr. Mahafuzur Rahman**, CTO, CodeMarshal, for his enormous help, support and direct guidance throughout the research to enable me reaching the stage that my work currently is in.

I would also like to thank **Mr. Saiful Islam**, Lecturer, Department of CSE, Daffodil International University, for his continuous help and support during the entire time.

I would like to thank my entire course mates in Daffodil International University, who took part in this discussion while completing the course work.

Finally, I must acknowledge with due respect the constant supports and patients of my parents.

ABSTRACT

In today's world data is the fundamental requirement for almost everything and accessing data is a required element in almost everyone's life. From student to billionaire tycoon, local shopkeeper to news reporter data is required in daily life to produce information and make decisions. Various sources generates various kind of data now. Devices, systems, offices, shops everyone and everything contributes to the regular increment of data. Data now is stored in more structured and proficient way in databases than ever before. The amount of data available right now is beyond the wild guess of a few years ago. But accessing this immense amount of data is not quite easy for the general audience. A complex programming language (SQL) is required to use retrieve data efficiently and as expected which makes the process bitter and harder. General audience is comfortable in asking questions in natural language, the language which we use in daily life to communicate with each other. But natural language is mostly unstructured and very complex in nature. Computer cannot understand this let alone act upon it, yet. Therefore an intermediate medium which is capable of accepting queries in natural language as input and retrieve data from database interpreting the input into SQL format can become a great solution this problem. The purpose of this study to represent a different view than what has already been done to solve this problem. This study has brought about 85 to 90% accuracy over various databases. This report has detailed insight on what can be done and what have already been done in this specific topic.

Table of Contents

List of Figures

FIGURES **PAGE NO**

List of Tables

CHAPTER 1
Introduction

1.1 Introduction

Storing and retrieving data has always been a priority work to us. From the time of ink and paper till now, days of super computers and data center, we demand to store data such a way that we can retrieve the data at any moment of our need. Relational databases now stores over 70 percent of data [1] in the world. Almost all of the **RDBMS** runs with a structured query language (SQL). SQL is a programming language which makes it very structured and complex to format when required.

On the other hand Natural Language is the language that we, human, have been using for over thousands of years. There are hundreds of thousands language alive or known to be existed throughout the history of humankind. All the language came to its development through massive changes, additions and editions. It is being developed for thousands of years which also made it complex in nature. We cannot bind natural language to a set of rules. Natural language is very easy for human to understand and use for the fact that human brain is so developed that it can parse any sentence given to it even without structures. And again we learn the language from variety of sources which makes our brain super adaptive to understanding what is the inner meaning of any given sentences. This all in the context of the user's speaking language.

To computer natural language is very hard to interpret. Because of the fact that computer is not yet enough evolved to interpret unstructured sentences. Computer needs to understand what is wanted as in what the instruction is given for it to execute. Natural Language Processing is the field of computer science which deals with problems related to natural language and computer's interpretation of them. Combining the requirement of accessing data from RDBMS using SQL and complexity of the language it would be very helpful to make such system they can take natural language string as input and let user access the data by generating SQL in system intermediately.

1.2 Motivation

Natural language interface to database is a fairly old area of research. It is almost as old as natural language processing. The reasons are many in number. Its easier to ask in natural

language than in a structured language. It gives more access to users. To add or retrieve data from a relational database management system one needs to know this programming language. Apart from system administrator or programmers or developers it is not common for general users to know that language. Which is a very big hindrance in the path of access to data. That is why it has been always a challenge to the researcher to make such a system which will be able to accept queries in natural language and execute upon.

There are many systems exists now that let users access data from RDBMS. Most of them nowadays offer graphical user interface. Making a GUI is often complex and needs a lot of works to be done behind the UI for users to interact. Often graphical interface of a database does not provide enough options to explore the data more freely. There are many ways data can or may need to be retrieved from database. All those combinations are often not possible to implement in GUI or sometimes the necessity comes at sudden point. Furthermore GUI also limits the boundary of data types. In those situation running a SQL query could solve the problem of not having an option to that graphical interface.

Therefore an interface which can accept natural language queries and act upon will make it easier for both users and system engineers to develop systems.

1.3 Rationale of the Study

Making a natural language interface for database if a fairly old area of research. This problem is almost as old as natural language processing field. The reason is very natural. Human's interaction with computer contains basically making the computer work as commanded. Computer uses memory to store data, instructions to present when asked. Therefore our most commands to computer require accessing its memory. Now to elaborate it to computers which are used to store data are often gets command to display the stored data. In this sense accessing data plays a major role in computer science. And to access data more easily and comfortably computer needs to understand what is asked precisely. Many researchers in past have worked on this problem and proposed many viable solution for the time being. The complexity of natural language is yet to be learnt by computers at fullest extent. Mapping an unstructured string into a very structured one (SQL) is generally a tough task. Moreover databases are not limited to a certain number of entities or features. Both of these make the problem more appealing yet very hard one to crack. Every work done in the past (except a few) have been done against a specific database to show that it can be done but many of them failed to actually provide any solid adaptive solution. Therefore the problem still exists as to work from the very

fundamental. Researches is still going on to solve this problem more precisely and more efficiently.

1.4 Research Questions

Every complex system is basically made up of many small simple parts. As such a research problem often becomes understandable once every part of the problem is defined properly. The research this report is made upon also comes with many sub problems that needs to be understood before understanding the actual problem. This section of the report deals with all the basic questions that comes in with the topic itself.

1.4.1 Natural Language Query

A natural language query (nlq) is a string produced by an human being AKA user using user's native language. This includes all structured and unstructured sentences. The sentence may or may not follow rules of grammar. The sentence even may or may not a complete sentence. For human being this is very normal to converse in natural language. The language (any one language) is being developed for thousands of years and thus now very complex.

1.4.2 Structured Query Language

Relational database management systems (RDBMS) use their very own programming language to execute operations (create, retrieve, update, eliminate) on the data stored in the database. This is called **Structured Query Language** every RDBMS supports SQL. This language is quite complex in nature and requires experience to use efficiently.

1.4.3 NLQ to SQL

The goal of this research is to propose, implement and test such a system which will be able to accept NLQ from user and interpret it into a SQL and later retrieve data from the database. Inserting or removing data into or from database is very crucial and requires expert observation. Little mistakes there can result in major problems. Therefore we have confined our work within retrieving queries. Every database is different from every other databases. There has been many attempts to create such systems before. All those works were limited to work on the target database and hence does not work in other database. This requires lot of manual works to make

such system works. This is the most crucial reason to build a system that can adapt to any given database.

In this research we have addressed this specific problem, to make a true dynamically adaptive system, thoroughly. In the later chapter we have disclosed more details on the problem and our proposed works. Another problem of such system is that database may contain various data points and so does a natural query. Therefore the system needs to understand the relation between a value and its corresponding location. We have also attempted to solve this problem of matching data from database to map given query string(s).

This research is limited to work with the natural language English for the time being. The reason for choosing English is that English has the most amount of resource for parsing and gathering enough information easily.

Every query contains an objective of search, in the sense of data retrieval, which needs to be identified to get specific data rather than unnecessary fields. Our system can detect what is required, if match found on database metadata, in most cases.

1.5 Expected Outcome

This research includes both a proposal to solve the stated problems and a basic implementation to test the hypothesis. The primary objective of this project is to propose a system that can extract SQL from a given input string in natural language for any connected database. That is to propose a true dynamically adaptive system for the problem. This problem includes many sub objectives. Among the sub objectives it is expected to solve the following ones -

- Ability to work on any RDMS that uses SQL to execute queries.
- Ability to identify condition from existing data in database.
- Ability to detect true dependency between sub string in given query to fine tune conditions.
- Ability to detect exactly what is wanted.

1.6 Report Layout

This report is designed in such a way that going through the report will give a complete understanding of the system, how it works, how to reproduce and outcome of the research. The report follows the standard thesis reporting template provided by DIU.

This chapter gives the insight of the problem, objective of the research, what motivated us and the introduction to the report itself.

The next chapter discusses background, related works and challenges in the problem. To understand the state of the art this chapter includes detailed citation and work descriptions of what has already been done. Their working procedures and a brief outcome of the works. Shortcomings of the existing systems are also mentioned in brief.

The following chapter contains the methodology and techniques used in the research in details. The system's in and out description can be found here. This can also be used to reproduce the system. The chapter contains all information that one needs to carry on the research.

Chapter 4 contains the details on implemented system. The proposed methodology has been implemented into a web based system which is described with all functionalities in this chapter. Diagrams and methods for reproducing the system is also explained there.

Chapter 5 deals with the outcome of the research, implementations, and other analysis of the research. Data sheets of testing and stats can be found on this chapter. This chapter shreds light on the success of the research. The metrics mentioned in the chapter can be used to evaluate the methodology.

The final chapter contains limitations, conclusions, future works, and a summary of the work.

CHAPTER 2
Background

2.1 Introduction

As mentioned earlier the problem is almost as old as NLP itself. Retrieval of data more efficiently and easily has always been a priority field of research. The more efficiently users can access data the more possibilities open up. Storing data has been made easier with the development of SQL [Structured Query Language]. But for common users SQL is not an easy knowledge to use. In some cases admins of different domains also need to achieve data from such databases. Which may be hindered by the lack of knowledge of SQL. Therefore the problem has its own kind of appeal to everyone.

2.2 Related Works

There has been many researches on this specific problem. Both proposals and systems have been made to solve the problem efficiently. For the last few decades a lot of new approaches and ideas have been introduced to solve this issue. Every work has its pros and cons. All proposals, made, have been helpful to the pathway for a better solution later on.

Lunar (Woods, 1973) [2] is the first published system of such kind. Lunar took user's query in natural language and answered about rocks that were brought from moon. The system made was confined to its usage within the database containing information about the brought rocks. Although the work was very limited but it certainly did show what is possible and what help could it be if it's perfected.

Allen J, 1983 has described the importance about such kind of system in his book [3]. In his book he described how the linguistic in computer science has been changing over the period of time. The necessity to understand small insights from languages are very clearly discussed here. Various kind of mathematical models are introduced and how parsing natural language can bring more flexibility is addressed. This also includes a fair explanation on how wider and more realistic range of features found in human language and to limit the dimensions of program goals.

Lifer/Ladder designed by Hendrix, 1978 [4] is another implementation of such system which supported single table from a database. This system was for Navy ships which could answer simple questions about them. This represents how an interface to provide access to large body of data distributed over a computer network via natural language can be introduced. The work limited to one specific table of data that is worked previously upon.

Agrawal, 2013 is not a very old system proposal, which used domain ontology to process the input query and analyze the semantics [5] understand the meaning of query [6]. In the work authors proposed a deeper method of analyzing detailed knowledge about the given query. This domain ontology based system works very good for a given ontology. The paper described in details how domain ontology works along with a database containing information about train schedule of Delhi. The implementation shows an appealing side of domain ontology. Although the limitation of the system is that it is necessary to analyze the domain of the system accurately which requires lot of manual works. Defining domain and fine tuning its ontology is what prevents it from becoming adaptive system.

An expert system proposed by Siaser et al., May 2008 [7] which used semantic analysis to translate queries in natural language into SQL. In ontology or semantic analysis based systems the system requires to have lot of information about that particular domain. Which is used to setup the system. Whereas our system does not require any prior knowledge about the system.

System by Rao et al, 2010 [8] have established a system compatible with simple queries along with basic implicit queries. Whereas Chaudhuri et al, 2013 [9] covers some aggregation functionalities as well. Aggregation functions are more complex than just retrieving data from database. It requires more mapping and deeper understanding of the input queries.

Another approach done by **Filbert Reinaldha** [10] in 2014 which is based on dependency parser and able to distinguish question from general directive sentence in input query. This approach tried to solve the dependency on specific database by processing the query alone.

A much newer work **nQuery**, 2017 [11] has attempted to make a much more efficient and independent system. Their system focuses on mainly table, attribute mapping as well as clause tagging which allows to generate more accurate output. Our work depends on both query analyzing and mapping.

A language MASQUE/SQL was proposed, an update to the previous MASQUE, by I. Androutsopoulos, G. Ritchie, P. Thanisch [12] to accept natural language queries from users. The system maps the language to a secondary conditional base named LQL which then is converted to SQL query. Our system maps directly to SQL without any new language.

Quepy is an working system [13] that can translate natural language into SQL queries. The system works with very basic level of sub-string matching. Our system works with any kind of given query as long as the database has some reference to work with.

All of the previous works shows us that it is indeed necessary to build such a system and that it is actually possible to solve the problem with enough initiatives and working. The fundamental problem lies in analyzing the input query and mapping properly into the respective database. All the works has shown their unique way of representing the problem and attempt to solve the same problem in their own ways.

All of the mentioned works have helped us mapping our own way of solving the problem.

2.3 Research Summary

We have investigated the problem and all (what could be found) related works to make a very clear understanding of the problem and what has been done so far. The research included reproducing some of other authors' works partially to get better pictures, mathematical representation of others' works. The research shows that the problem still exists and needs to be solved to enable access to data more easily. Our research and proposed work has been tested and proved to be a working solution to the problem. Our system can generate better SQL for any given queries for any attached databases. Although our works also have its own limitations and future scopes of work which is described in details in the later sections of the report.

2.4 Scope of the Problem

The necessity of a system that enables access to data through natural language is very appealing. The implementation of such system at fullest extents will help everyone.

Everything around us is now data-driven, dependent or produce data. Access to mass amount of data is being blocked because of no or lacking of knowledge in relational database

management system. Therefore making a system that can accept queries in natural language will enable access to this data.

Every users will be benefited from the system because they will not need to learn complex syntax of SQL and understand database structure properly. They can acquire data as they please. This will, in turn, enable them to use more time/focus on interpreting the meaning of the data and applying it to solve their problems rather than the mechanical process of accessing it. This will both save time of the users to get data from the database and make users more efficient and productive.

System admins and designers or developer will also be benefited massively from such system. Designing a system often takes long time to work and implement everything in order. Such system will make it easier developers to leave the complex part of options to the system for working on-demand rather than implementing everything.

If a system as such can be perfected the field of artificial intelligence will hit another level in Human Computer Interaction. If computers can be made to interpret queries as complex as querying a database a more better understanding of natural language can be derived.

2.5 Challenges

The problem is very complex in nature because of many aspects mixed with it. There are many sub problems that needs to be solved to actually achieve a promising solution.

Natural language is very complex itself. We cannot bind user to any set of rules for querying. If we confine users to many rules for getting proper data it will become a structured language close to SQL which will fail the motive. The complexity of natural language needs to be parsed very delicately. To handle this there has been many researches and systems are being developed day by day. Properly parsing a natural query is more important than anything in this problem. Same motive can be derived in many forms. I.E. Students of CSE can also be represented as Students from CSE or students of cse department. In this example using the phrase department has no actual specific meaning rather than amplifying the CSE phrase.

Another problem that is significant is identifying what is actually presented as a condition to match. Now there are many criteria that may be used to define one condition. A data that is

present in the database can actually be referred in the database to narrow down the query. For example in a student database a data of email can be very specific which is required to be matched i.e. Student who has email xxx@domain.com. Another requirement is that keyword may actually be present in the query that defines a kind of condition.

One word can be in many forms in natural language. Handling every form is not easy or a feasible solution. Therefore it needs to be normalized into a single form. Which is called stemming or lemmatization. This can also produce some other sub problems like not being able to match stemmed with data in database. For example let's assume there is a data in database "working", now if the input query also includes "working" as to be identified as a condition stemming will produce "work" from the query. Now this will not be matched with "working" in the database. Therefore while stemming is necessary to generalize forms it also may produce further works required. A solution to this may be stemming the data from database as well. But that will make the system computationally unusable for larger databases.

A more complex problem in the field of natural language processing is to parse multi-worded phrase as a single entity. For example a data in database can be of multiple words. We are defining words as sub-string of multiple characters. Now while parsing parser may not identify a phrase as a single entity parse it as multiple tokens which will result in producing wrong conditions. A viable solution to this problem is still undergoing by many researchers. Computational costs for large databases is also a concern of the problem. Although optimizing comes after the implementation of such system and can also be addressed as a very separate problem.

CHAPTER 3

Research Methodology

3.1 Introduction

After researching all current approaches we have learned all the benefits of approaches and their limitations and scopes as well. Therefore our proposed methodology includes an overview of what is done to prevent what or to extend what capability. This chapter of the report contains every steps, data, algorithms and procedures explained.

3.2 Research Subject and Instrumentation

Domain: The problem lives the domain of natural language processing. Although the complete problems in spread over multiple domains such as data analysis, NLP, RDBMS etc. The extensive nature of the problem is what makes it more unique to its kind. The problem can be extended to solve in every language available and used in the world right now. But our research work is limited to one language, English, only. The reason for choosing this language is multiple. English has the highest number of available tools for parsing and algorithms. Picking a language which has the best probability of deeper understanding will have better chance of showing positive outcome. Therefore we picked 'English' as the chosen language for this research work.

Instrumentation: To solve the underlying sub-problems of the main problems it is necessary to use the state of art instruments to get better outcome. Our methodology contains the usage of various packages from very well developed software as well as algorithms.

Techniques of NLP has been being improved on a regular basis. Parsing sentences stemming, lemmatize, POS tagging, dependency parsing has been made very easy due to massive improvements to the package built for NLP. Some necessary algorithms are briefly mentioned here -

1. **Tokenize:** In the field of NLP Tokenization is a very useful technique which is used in almost every NLP application and algorithms. Tokenize is the system of making chunk from any given string and divides the chunks into individual sub-strings. Tokenization algorithms return list of sub-strings made out from the given string.

2. **POS Tagging** Parts of speech tagging or identifying which parts of speech is a given token is very important for our purpose. For example a verb or a determiner can be safely ignored from our query string. Because in querying they play a very little or insignificant role as they play on a real conversation.

3. **Lemma/Stemming:** Stemming a token or finding lemma out of it is the algorithm to generate a generalized form of a token to be used in more universal form throughout an application. For example go, goes, going, gone every one of the words has the same root go. Therefore writing rules for go and stemming all other forms to go is much easier and helpful then writing rules for all of them.

4. **Mapping:** Mapping is the technique of returning a corresponding value. In our system we need to map any values accordance with the database it is attached to. Therefore mapping is also a very important instrument to us.
5. **Thesaurus:** Same word can be represented in many other ways. There we cannot definitely say user will be querying using only one, specific, one. Therefore it is necessary to consider thesaurus for the system.

Many other instruments have been mentioned throughout the detailed methodology of the system in later parts of the chapter.

3.3 Data Collection

We needed various kind of data from various kind of sources the collected and used data can be categorized into three categories.

1. **Development Data:** We needed multiple kinds of data for this stage.

   (a) For stemming/ getting lemma we have used the English language library of stanford-corenlp. This data is massive in size [˜1 GB]. Used in the core-nlp pipeline to analyze input strings. This also helps the core-nlp server for POS tagging, tokenization.

   (b) For getting thesaurus we used a different source of data. The wordnet dictionary dump is used to get thesaurus of a given word.

2. **Test Data:** After implementation the system we proposed needed additional data to be tested. Test data includes various databases from different sources. We have tested our system across mainly databases. The databases' descriptions are as follows -

    - **Music.db:** This is an sqlite3 standalone database containing information of over 3000 tracks.

There is a single table 'tracks' containing all the information. The columns are described below -

- *id* - Each instance contains an unique id for the songs.
- *track* - Name of the track of that instance.
- *album* - Album that contains the track.
- *composer* - Composer of the track.
- *artist* - Artist/singer of the track.
- *duration* - Duration of the track in milliseconds.

- **Govt-data.db:** This is a database downloaded from government website. This database contains over 300,000 instances of institutions in Bangladesh. The institutions are of education in kind. There are over 20 attributes in the database.
- **TOPC.db:** This is a database converted from a comma separated file (csv). This database includes registration data of 300 students for a contest. There are over 10 attributes in the database.

3. **Train Data:** For CoreNLP and NLTK to run properly they need language models and corpuses. Those data comes with the library themselves therefore no additional description is added to this report.

Data Source: All the data used throughout the research are from open sources and were made sure no copyright was violated. Libraries that are used are also under open source domain therefore data came with them are also subject to their copyright and under the open source domain.

3.4 Detailed Methodology

This section and sub-sequent subsections contain detailed step by step details of the proposed methodology and works that has been done.

3.4.1 Overview of the System

The system can be provided with a database to generalize the use-case. It can also be made system specific with a very little work once. Upon receiving the connection to a database, the system will analyze the database to generate metadata about the database and saved. Details on metadata is discussed on later section. This metadata will be used to map and hash query in later portion of the system. Then the system is ready to take query. The query is first parsed to produce lemmas, tokens and pos (Parts of Speech) tags using Stanford CoreNLP [14]. Afterward the query is hashed to chunk out sub-strings that are database entry. This results in generating a primary SQL query following a SQL template library. The hashed query then goes through dependency parser to check dependency between words. Combining and comparing the parsed result with primary query a secondary SQL query is generated. Which is then shown to user and used to retrieve data from database.

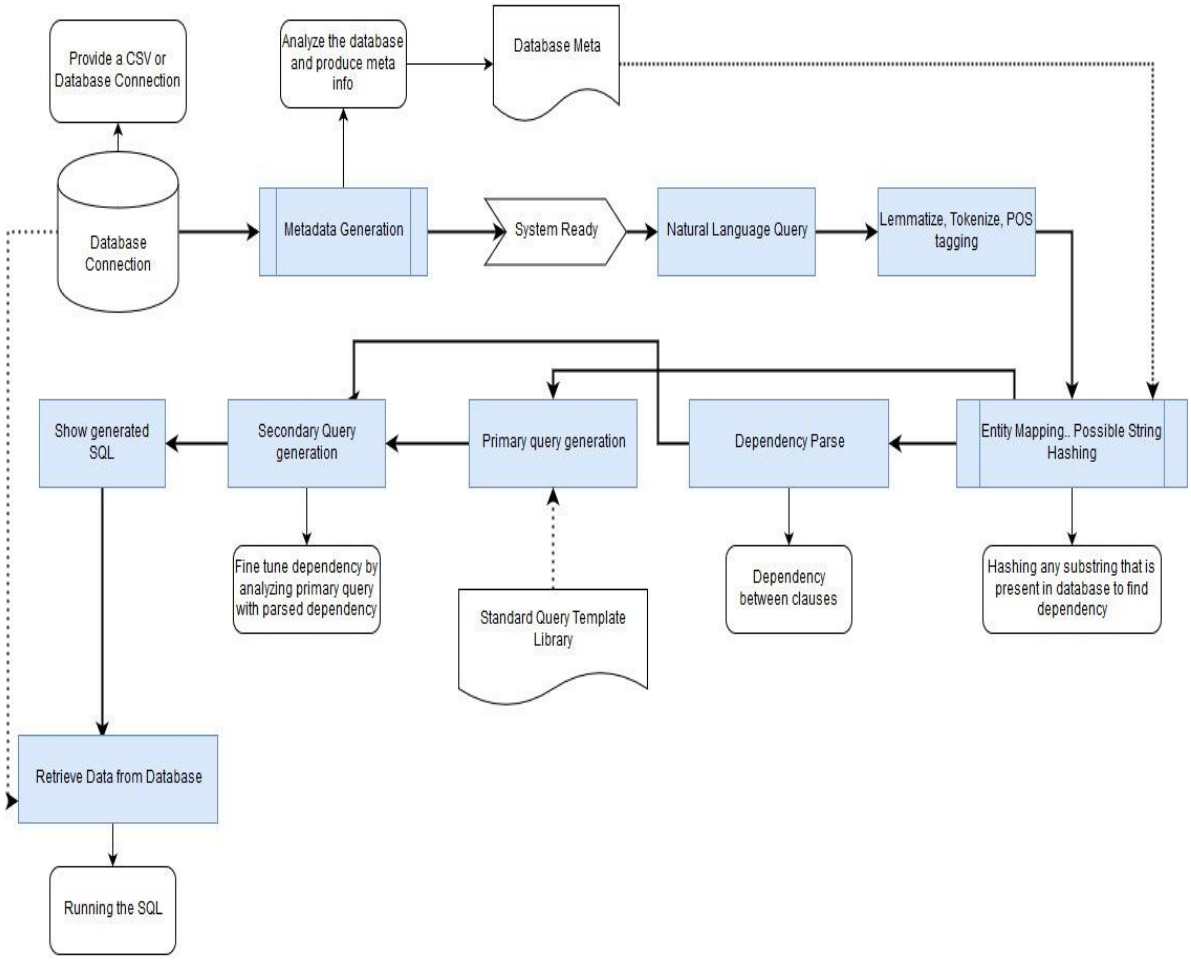

Figure 3.4.1.1: System Diagram

3.4.2 Database Connection

The primary goal of this work is to design such a system that can or will be used across any given relational database management system that runs on SQL. To be a database independent system the system needs a way so that any database can be connected to it with minimum effort. For the ease of our work and portability we tested our built system with SQLite [15]. With a few tweaks it can be used in any RDBMS available out there. The database connection is very straightforward we just need to define the path of the offline database file of SQLite database which goes by the extension of *.db*. The system stores the path to the database and runs queries to retrieve data after generating SQL query.

3.4.3 Metadata

Creating meta data is a very important task after connecting with the database. This process runs **only once for one database**. The process can be shown as following:

Every SQLite database and any other RDMS contains one database or at least a table containing information about databases and tables stored in it. To prepare meta data we first need to get what is inside the database. Thus we fetch the table information which contains the columns' name, data type

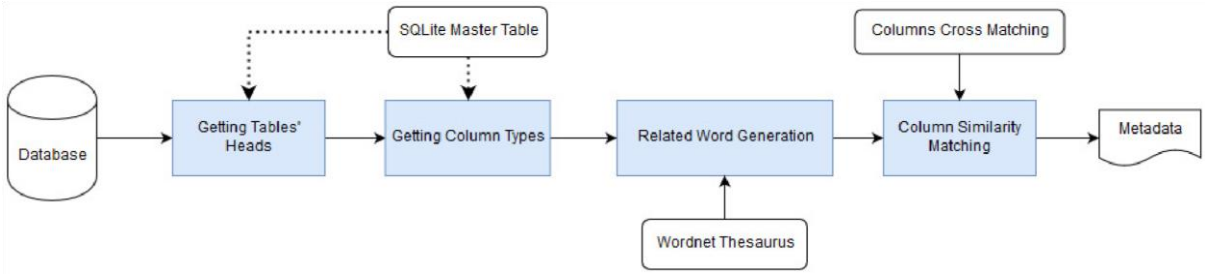

Figure 3.4.3.1: Metadata Generation Flow

and other information. We store column names and their data type in dictionary format in Python. Each analyzed attributes is stored in same way to its corresponding table and column.

Now we need to consider that natural query will not always consist exact column names as database. Because users cannot know that. Therefore it will be often synonyms that will appear in the query. To solve this issue our system will store possible synonyms for the column names as well. The system uses Python's NLTK library's WordNet [16] module. All thesaurus is then stored to its respective column's "thesaurus" attribute.

There will situations where two columns might contains similar string a number of times. If the number of appearance is less this might or might not raise any issue. But if there are many frequent same string it will handy to know the other possible column options. To address this problem our system cross matches every columns that has string type of data in it. For larger database it will take longer period of time which leaves us space of optimization. But since this will occur only once it can be defined acceptable. After cross matching the similar column names will be stored in metadata under an array named "similar". After the process a portion of a database's meta data may look like this -

```
{
    "tracks" : {
         "artist" : {
            "type"  :  "TEXT"  ,
            "thesaurus" :    [
                "artist" , "band" , "singer"
            ] ,
            "similar":[
```

```
                "album"
            ]
}
```

3.4.4 Lemmatize, Tokenize, POS Tagging

Once the database metadata preparation is complete the system is ready to accept query. Input query is basically a sentence containing many words, in NLP called tokens. To interpret the query string for system we need to split the string into tokens and then analyze the tokens precisely. Natural query generally consists of words in various forms other than their base form. Thus they are needed to be lemmatized i.e. **students** needs to be transformed into **student**. This is to make sure all words are in same form and can be understood by the system. Here is an example of the process with input query and its corresponding outputs –

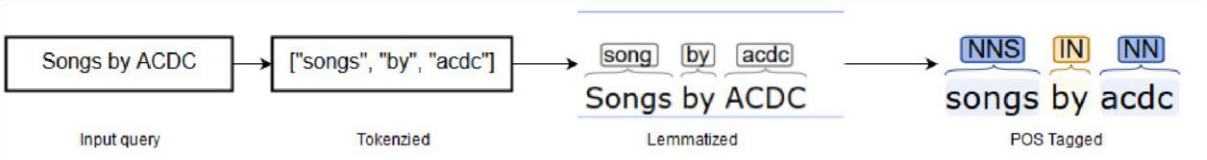

Figure 3.4.4.1: Parsing Input Query

3.4.5 Entity Mapping: Hashing

This is by far the lengthiest portion of our system. Before going into the technical details we should discuss what is the goal and why it is done? To build a system which can actually work on any database needs to look at the data that the database holds. Lets break it with an example. Suppose we have an database containing information about an hotel. Now when a user says something like "Room number of Harry Sheldon". Now because of the parsing in previous phase we got the tokens in the sentence. But how do we know what is given that might be in the database? We can easily understand that Harry Sheldon is a person who stayed in some room at the hotel. Therefore the string 'Harry Sheldon' is stored somewhere in the database. if we find out where it is stored, in case of database which column, we can make out one condition to put in SQL condition. For this case let's say 'Harry Sheldon' is in customer name column that makes up a condition as 'customer name = 'Harry Sheldon''. Searching through the database can be time consuming but it helps the system to be more accurate.

Found word(s) in the database is/are hashed as one i.e. Harry Sheldon to #harrysheldon. By doing so we are making sure when dependency is parsed multi worded phrases will be counted as one and they don't need to be mapped from dependency.

3.4.6 Dependency Parsing

After we get an hashed string the system runs it through a NLP Dependency parser. The purpose of this stage is to get dependency between tokens to get deeper understanding of what is required to search. Because of the hashing now multi worded phrases that are present in the database will be counted as one single word and thus the dependency can be analyzed more efficiently. Stanford CoreNLP's basic dependency parser provides an excellent result on most of the queries identifying dependency among the tokens and also analyzing the required clause. The subject of any given query is marked as 'root'. In general if we think subject is the item we normally ask to be shown. i.e. 'name of the student of class 4 ' here in the query 'name' is the root which is indicating the required selection for the query. Now the name is dependent on 'student' where as 4 in dependent on 'class'. The system understands this and make out as follows - 'SELECT name FROM student WHERE class = 4 '.

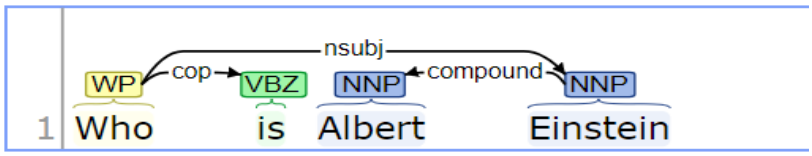

Figure: 3.4.6.1: Dependency Parsing

3.4.7 Primary Query Generation

While the dependency parser parses the hashed query, a primary query is generated using the hash information returned from entity mapping. This is primary one because it can only map the condition upon finding the sub-string(s) present in the database. To do this the system has it's standard SQL template library, for the starting it only contains a general query that is "SELECT * FROM table name WHERE conditions". From this template we then replace the "conditions" with conditions generated in entity mapping stage. We also get the table name from that stage and replace it in the template. In many cases this primary query may be the answer to the entire given queries already.

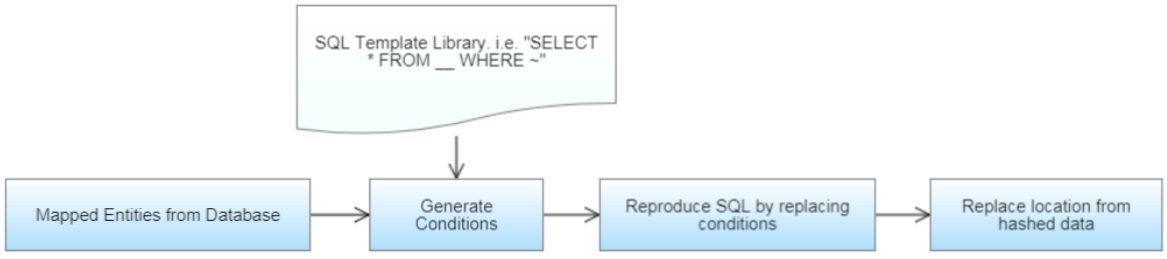

Figure 3.4.7.1: Primary SQL Generation

This entity mapping also does the similarity analysis. While searching for sub-string matching similar column names also gets attached with the result that will be returned. The target of this is solve the issue where one value might be in multiple different columns and letting the users independence on choosing which one to select. This entity mapping will also provide the related table names. That is if there are multiple tables it will return result with multiple table names as well. Although for this work our system does not support database with multiple tables, yet.

3.4.8 Secondary Query Generation

While the primary query gets generated the system also gets the parsed dependency. Now the system merges both (parsed dependency and primary query) to fine tune the generated primary query. Most of the time system will get the requirement clause from the dependency parsing, what to select. Parsing will also provide with enough information to generate conditions from dependency among words. For example if a string gives a numerical value that is dependent on age (i.e. customer under age 30 years old) the system will get condition for age value. Now here other metrics also influence the condition generation like if there is any column with thesaurus or name with that dependent clause. Say for previous example if there is no such column this condition will be discarded.

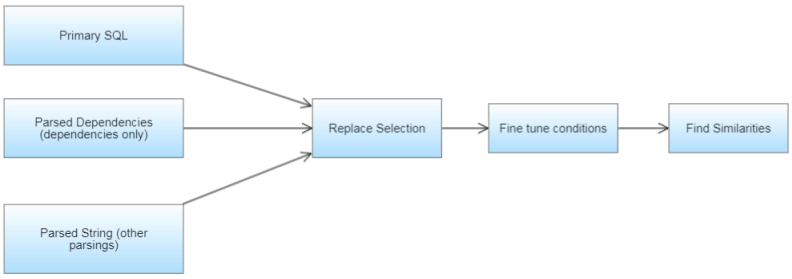

Figure 3.4.8.1: Secondary SQL Generation

By merging primary query and parsed dependency we get a more narrowed down SQL query with more conditions and short selection list. This SQL query is what we are calling the Secondary Query. This secondary query is the SQL interpretation of given natural language query. This is the last step of our natural language to SQL translation system.

3.4.9 Show Generated SQL Query

The secondary SQL query is now shown to user for any correction (if possible). Although the Primary SQL query is also shown to user after creation. The generated secondary SQL query

is shown to users in such way that users can choose between similar column names. This works to provide the users flexibility to ask queries without wondering which column they fits, that is without any actual knowledge of the database schema.

3.4.10 Retrieve Data

After generating the Secondary SQL query in the previous step, the query is run against the database to retrieve data. This is by default automated process to retrieve data and show to user. Collected data then can be manipulated for better user friendly view. After changing to any options from secondary SQL Query preview the changed SQL query is again run against the database to retrieve data.

3.5 Implementation Requirement

The details system information is given below:

Hardware Requirement (minimum):

Processor:

Over 3.0 Ghz and 3 core CPU and multithreading enabled.

Memory:

At least 16 8GB of physical RAM.

Storage:

At least 20 GB of HDD space.

Software Requirement:

Operating System

- Linux - Ubuntu 18.04
- Windows - Windows 10 (professional)

Required Environments

- Python 3.6
- Java - JDK 8.1
- Anaconda
- WSGI GUNICORN

Packages

- Pandas
- Stanford Core-nlp
- NLTK
- Messy Tables
- Wordnet

CHAPTER 4

System Implementation

4.1 Introduction

Every methodology or hypothesis requires to be tested thoroughly. What could be better than implementing the idea and put it to test. Therefore we have implemented our proposed methodology as a working system [development/rough build] to test our hypothesis. This chapter of the report deals with the in and out description of our implemented system. This chapter also describes input parameters and output parameters of the system.

4.2 Overview

We have implemented our proposed system to test against various databases. The system implementation is done in Python. An web based interface is developed to input queries in natural language and see output. The detailed system specification is discussed in the last chapter in details.

The system is capable of taking input query in natural string and then parse in the background where a engine is designed to do the parsing and other works to generate the SQL. The system contains 2 part.

1. Front End (UI)
2. Back End (Engine)

Both of them are described in the following subsections.

4.3 Front End

A user interface is built for using the system. The following figure shows the web based user interface for the system.

The front end interface consists of several parts -

- Upload file
- Select file

- Metadata viewer
- Development console
- Input Query in Natural Language
- Primary SQL
- Secondary SQL
- Fetched result

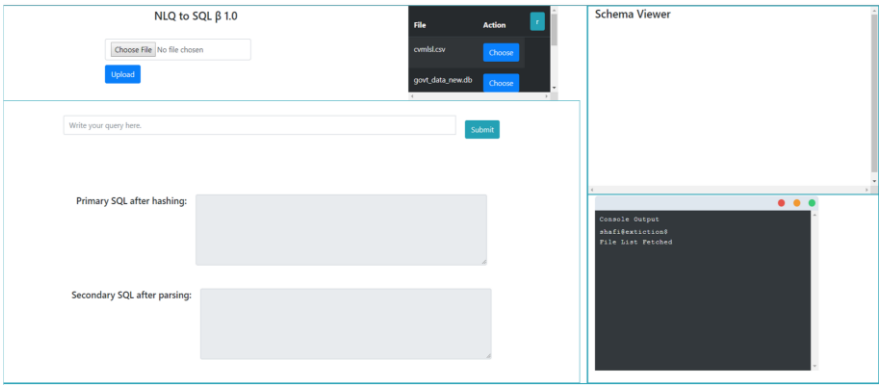

Figure 4.3.1: Front End of the System

4.3.1 Upload File

Our system accepts uploading database in form of csv (comma separated values) or db (standalone database file) format for the time being. But it can easily be connected to any SQL supported RDBMS using respective driver. When a full featured system is available that can be integrated very easily.

Uploading is supported once per session. That is after uploading the file the upload option is disabled and changed into showing which file is currently selected. This goes for selection as well. For csv file type it is first converted into a standalone database by backend engine. The process is described in depth in later part of the chapter. After uploading a file a meta data is generated for the database and shown in the schema viewer portion of the UI.

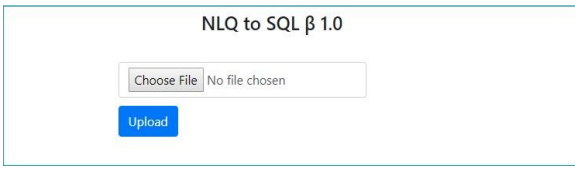

Figure 4.3.1.1: File upload system

4.3.2 Select File

Any file that has been uploaded before in the system is stored in a directory titled ‘upload‘. All the uploaded files are shown in a window to be selected by users. This enables users an hassle free database swapping.

Selecting an existing database will also hide the upload section and display the selected database in its stead. There is also an refresh option in the section to refresh the list of file if user wants to put any

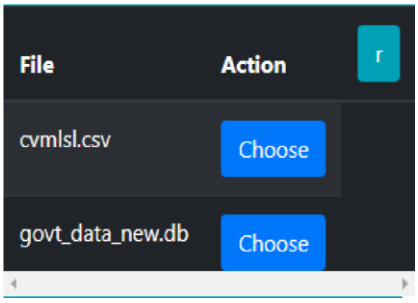

Figure 4.3.2.1: Selecting Existing Databases

database in the upload directory without uploading via interface. This can work better in case of putting multiple databases at once. And in case of development it may come in very handy

4.3.3 Meta Data Viewer

After uploading or selecting a database the system will generate meta data for the database. This meta data can be viewed in this window of the UI.

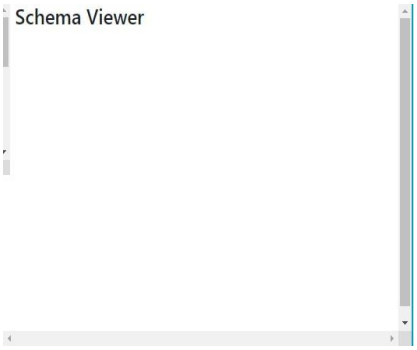

Figure 4.3.3.1: Metadata viewing window before uploading or selecting anything

The window is configured such a way that the meta data which is stored in json (javascript object notation) format is shown in a indented view. This makes it easier for user to know what is available in the database. Although it is not necessary for users to use any of the terms from

the metadata, it can come in handy if the input is not producing outputs as expected for the user. This window is auto updated when any change is occurred in the database or any update occurs in the meta-data.

Figure 4.3.3.2: Metadata viewing window after uploading or selecting database

4.3.4 Development Console

A console like portion is integrated in the system for the development run. That is while testing the system it may be required to test what is the backend producing. When something is passed to frontend from the

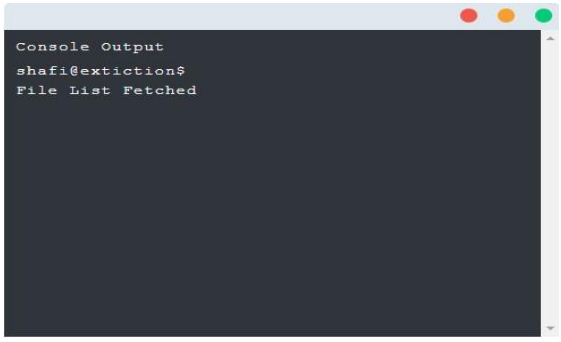

Figure 4.3.4.1: Development Console in the beginning

backend the console will show everything without processing. This is to make sure everything is getting passed between layers.

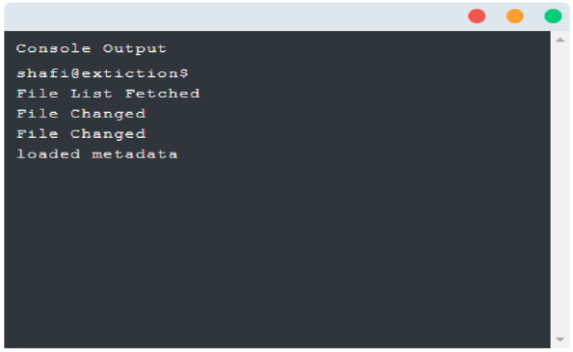

Figure 4.3.4.2: Development console output

4.3.5 Input Query in Natural Language

After uploading or selecting a database when the produced meta data is produced user can start querying the system. For querying a input field is added. Where user can write any queries in natural language to run against the selected database.

Figure 4.3.5.1: Input box for query in natural language

In the input box user can put any kind of query string without having any structural limitation. i.e.

The following chapters of UI will proceed to use this sample query and its output generated by the system. **Query:** *composer of carol*.

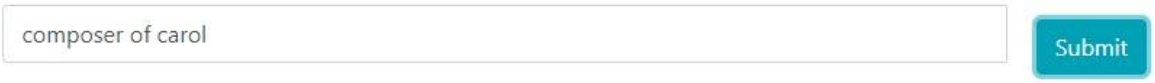

Figure 4.3.5.2: Input box including an example query

4.3.6 Primary SQL

After user inputs a query in natural language the backend engine is invoked to process the natural query string. The steps in the engine is described on the later parts of the chapter. After primary processing is complete a primary SQL is generated. This query includes the primary outcome of matching string against the selected database. This generates some of the condition by analyzing the metadata and query's substrings. The output is shown in a text area. This area

is made read-only for the fact that user do not need to edit this query anyways. And in the complete system it may be deprecated.

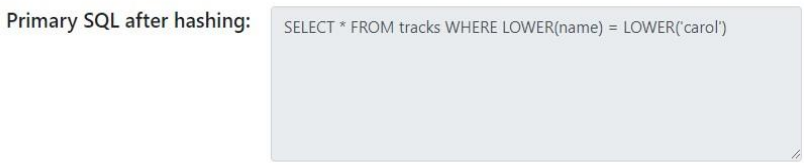

Figure 4.3.6.1: Generated primary SQL field

This primary SQL is then used for a deeper analysis and produce a better understanding of what is wanted. A point to be noted is that in the figure the SQL includes a selection clause of by default [* which is used to fetch all attributes of a table]. Take a look in the secondary SQL where it is replaced with a specific attribute field.

4.3.7 Secondary SQL

After the primary SQL is generated the engine proceeds to make a deeper understanding of the query itself. In the steps here a more detailed and accurate SQL is generated which usually contains a specific selection clause and more accurate conditional clause.

Another text area is there to preview this secondary SQL. This secondary SQL is then used as query parameter to fetch result from the database. In this SQL the selection clause is more specific that user

Figure 4.3.7.1: Generated secondary SQL viewer

requested only the value of composer. Therefore a more accurate SQL is generated.

4.3.8 Fetched Result

When secondary query is run against the selected database result is fetched and shown in the following section of the UI. Although this UI is not very user friendly again, this is just a development build to test the hypothesis.

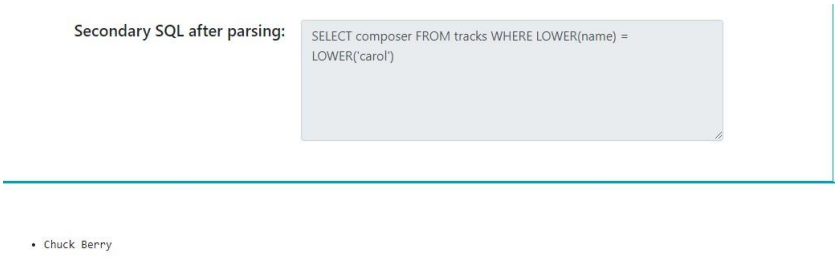

Figure 4.3.8.1: Fetched Result

4.4 Back End

While frontend is developed for ease of usage for testing and test users, the backend is made rather complex for handling the majority of processing. Our system's backend is developed on python's flask framework for handling http requests. The main engine for processing the input query is passed from the framework and parsed accordingly. Like the frontend backend is also composed of several parts. All of these parts and how they works are already described in the Methodology chapter. Therefore in this section we will be just going over them as a review and not in thorough details.

- **Request handler** All the request coming from the frontend is handled according by the backend. The probable requests are replied with html or json format as requested.
- **File uploader** A newly uploaded database file (csv, db) is send to engine for processing and stored in proper place.
- **Directory manager** Previously uploaded files are stored and processed as requested.
- **Metadata generator** When a new database is uploaded the engine reads it and prepare metadata accordingly. Produced metadata is stored as a json format. Metadata is also send to front end as a JSON object for using in schema viewer or any other purposes as may require.
- **Input query hashing** Entered query string is then processed to generate query hashes according to the database.
- **Query parsing** Input string is parsed used CoreNLP library for chucking, tokenizing, POS tagging, lemmatizing/stemming, dependency parsing.

- **Primary SQL generation** A primary SQL is generated and send to front end within a package for using in the frontend.
- **Secondary SQL generation** After processing the primary SQL a secondary one with more details and more accuracy is produced and also sent to frontend.
- **Querying the database** The secondary SQL is then used to run a query against the selected database.

The backend is the core of the system. This is where our hypothesis lies to prove that our approach is actually working. This not yet optimized engine requires about **36 seconds** to start up the first time. And then for each query it takes about 7-8 seconds to parse and produce an output. The backend is still very computationally costly and not thoroughly numbered for more average metrics.

4.5 Connection

To make frontend interact with the backend we used ajax as the medium. Ajax framework lets us make request between applications and process incoming data. When something is commanded in the frontend an ajax request is made to the server. And server process the request accordingly and when processing is done it returns a result to the requested end.

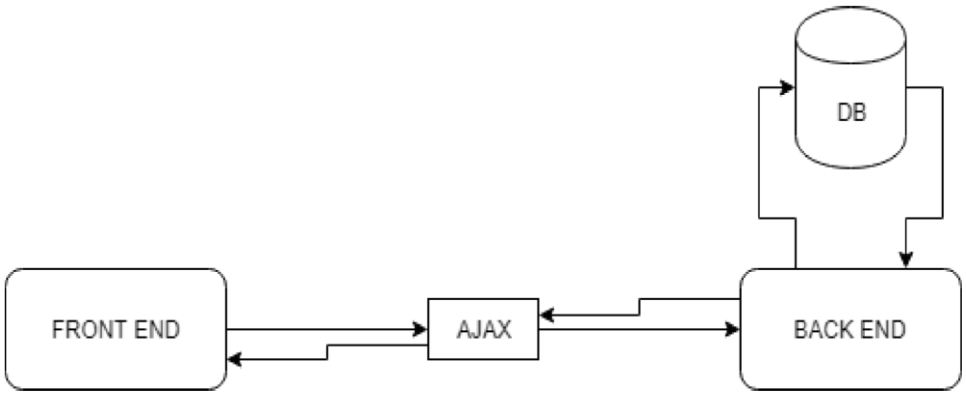

Figure 4.5.1: Frontend and backend connection

4.6 Steps to Reproduce

Any researcher who has read the report till now should have a very clear idea about how the system operates. This section of the chapter completes the knowledge so far. Anyone who

wants can reproduce our entire system from scratch and get a working solution for the system that has been done so far.

The system's BPM (business process model) describes a single request and its flow along the entire system.

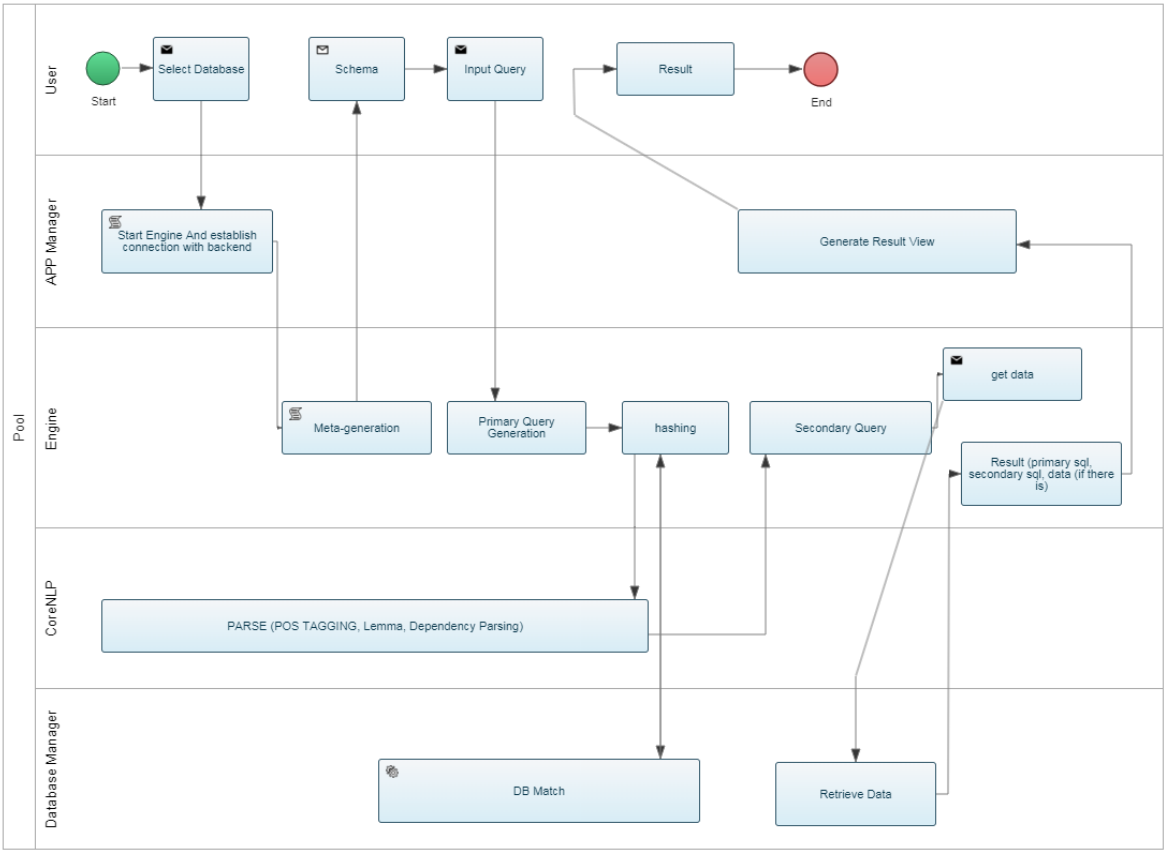

Figure 4.6.1: Business Process Model

There are 5 participants in the system where user is the only human entity. Human user starts the entire system by selecting the database. The system take care of the automated metadata generation task and all that is necessary to be done under the hood for a newly selected database. This also includes storing the metadata properly and sending it back to user as schema view.

After that user can input query to system for interpret and system, again, automatically take care of the rest of the tasks under the hood as shown. Finally a result package is thrown back to user as a view.

CoreNLP and Database Manager is considered independent entities because both are to be run as their own services and we or our system do not interfere with their processing. Therefore making them individual independent agent of the system. The system's interaction with them is regulated by the system engine nonetheless.

A detailed class diagram of our implementation is shown in the next figure. The class diagram contains all classes and operations that our system is capable of doing. Most of the methods described in the class diagram is not for the public usage and they are mostly called from the driver class.

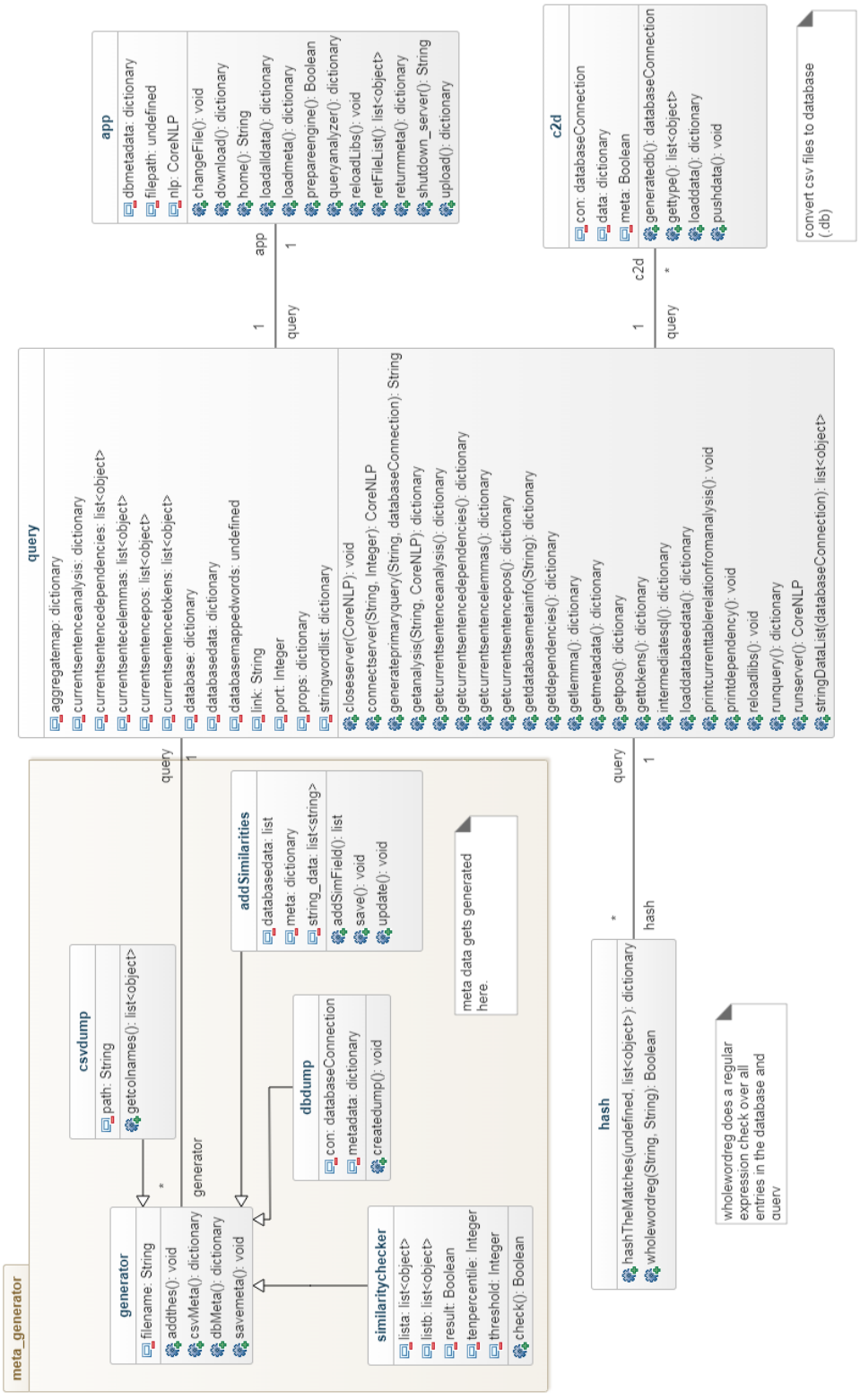

Figure 4.6.2: Class Diagram

Here **app** class is the handler class that joins the main backend to the frontend of the system. **Query** class is the main class that we were referring as engine in many places before. This class has the parsers and other analyzer to do the actual interpretation. A package **meta_generator** comes with various handy classes that helps to generate, store, reproduce, and convert metadata of a given database. Our system is capable of converting a csv file into standalone database for the ease of data accessing. **C2D** class is capable of doing the conversion. Hash class do the entity mapping from database and produces substring hashes for Primary SQL generation. The class diagram shows the parameter for each method as well. Most of the returned values are in Python `dict` (marked as dictionary) format.

Repository url for source code is available at appendix section.

CHAPTER 5

Experimental Result and Discussion

5.1 Introduction

Our implemented system has been tested across different databases. And the result metrics are in the favor of success. This chapter of the report contains inputs and respective outputs for various databases. This chapter also contains descriptive analysis of the outputs.

5.2 Experimental Results

This section of the chapter shows the actual inputs in natural language and their respective output generated by the system. As we have mentioned in the data section of the report we have tested the system against various databases, we will show outputs for multiple databases. Three of the many databases we tested are going to be shown in this section.

music.db

Table 5.1: Data sheet for music.db

#	Query in natural language	System output
1	who sang you shook me	SELECT * FROM tracks WHERE LOWER(name) = LOWER('you shook me')
2	songs by backbeat	SELECT * FROM tracks WHERE LOWER(artist) = LOWER('backbeat')
3	who composed carol	SELECT * FROM tracks WHERE LOWER(name) = LOWER('carol')
4	songs in big ones	SELECT * FROM tracks WHERE LOWER(album) = LOWER('big ones')
5	song in backbeat soundtrack composed by larry williams	SELECT * FROM tracks WHERE LOWER(album) = LOWER('backbeat soundtrack') AND LOWER(composer) = LOWER('larry williams')
6	number of song in out of exile	SELECT * FROM tracks WHERE LOWER(name) = LOWER('out of exile')
7	distinct composer in the album out of exile	SELECT composer FROM tracks WHERE LOWER(name) = LOWER('out of exile')

8	album of ac/dc	SELECT album FROM tracks WHERE LOWER(composer) = LOWER('ac/dc')
9	songs of 25000 ms	SELECT * FROM tracks
10	artist of twist and shout	SELECT artist FROM tracks WHERE LOWER(name) = LOWER('twist and shout')

As it can be seen above only 1 out of 10 cases it failed to detect any condition from the input query. Number 9 indicates an random integer value which may or may not be in the database. This creates an scope of future works. The limitations will be put off for the analaysis and later chapters and sections of the report. Even if it failed to detect any condition here it still creates the SQL of fetching all data. Which indicates the fail proofness of the system.

Therefore from the datasheet shown above it can be said the system performed with an accuracy of around 90%. A more detailed graph can be seen in the chart below.

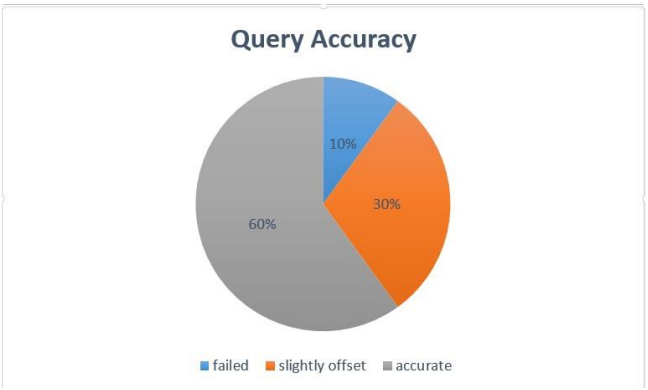

Figure 5.2.1: music.db accuracy measurement chart

topc 18 .csv

Table 5.2: Data sheet for topc 18 .csv

#	**Query in natural language**	**System output**
1	name of students from uttara campus	SELECT name FROM topc s18 WHERE LOWER(campus) = LOWER('uttara')
2	students from section i	SELECT * FROM topc WHERE s18 LOWER(section) = LOWER('i')
3	students who have tshirt size xxl	SELECT * FROM topc WHERE s18 LOWER(tshirt) = LOWER('xxl')

4	name of students from swe department	SELECT name FROM topc WHERE s18 LOWER(department) = LOWER('swe')
5	name of students whose payment is done	SELECT name FROM topc s18
6	name of students whose payment is ok	SELECT name FROM topc s18 WHERE LOWER(payment) = LOWER('ok')

As it can be seen from the datasheet above all the query has been identified properly and the respective SQL is generated properly for every query. That means system worked 100% accurately for this particular database.

gov data.db

In the datasheet above one thing may seem odd which is the way in 4th query institute name is used (INSTITUTE NAME). The reason is database may contain column names which are not proper words. Therefore system may become confused with anything other than that like using only 'name' in 5th query returned everything with the selection clause (SELECT *). Detailed analysis is on the next section. An accuracy map is built upon the data –

Table 5.3: Data sheet for govt data.db

#	**Query in natural language**	**System output**
1	madrasha in barishal	SELECT * FROM govt data new WHERE LOWER(INSTITUTE TYPE) = LOWER('madrasha') AND LOWER(POST) = LOWER('barishal')
2	high school in bogra	SELECT * FROM govt data new WHERE LOWER(INSTITUTE TYPE) = LOWER('school') AND LOWER(POST) = LOWER('bogra') AND LOWER(ADDRESS) = LOWER('high school')
3	ADDRESS of alim madrasha which is recognize	SELECT ADDRESS FROM govt data new WHERE LOWER(INSTITUTE TYPE) = LOWER('madrasha') AND LOWER(EDUCATION LEVEL) =

		LOWER(’alim’) AND LOWER(AFFILIATION) = LOWER(’recognize’)
4	INSTITUTE NAME of institute which is alim madrasha	SELECT INSTITUTE NAME FROM govt data new WHERE LOWER(INSTITUTE TYPE) = LOWER(’madrasha’) AND LOWER(EDUCATION LEVEL) = LOWER(’alim’)
5	name of institute which is alim madrasha	SELECT * FROM govt data new WHERE LOWER(INSTITUTE TYPE) = LOWER(’madrasha’) AND LOWER(EDUCATION LEVEL) = LOWER(’alim’)

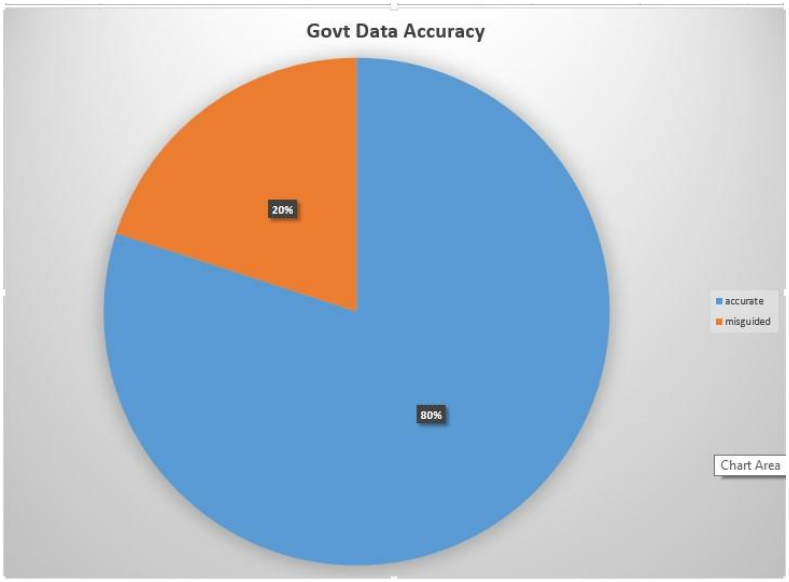

Figure 5.2.2: govt data.db accuracy measurement chart

5.3 Descriptive Analysis

Data sheets in the previous section are pretty self-explanatory about the outcome of the research. Our approach proved to be of working after testing in many databases. Although not all metrics are shown in the results section. To explain the outcome necessary ones are presented there.

The system proved to be working with various databases without working for individual one. That proves our first hypothesis of the problem. A dynamic adaptive system which compatible with any SQL driven RDBMS.

Generating conditions can be achieved by hashing the database. Although the size of the database and number of instances in it will be a matter of concern while working on real life system. System can be confused with random ranged number as happened with the music length in query of music database. Therefore it shows a limitation of the system.

By parsing the query using techniques of NLP can bring up deeper understanding of the input query. With the parsed result generated SQL contains more accurate structures. This parsing requires a lot more works to become more precise. For example in the govt database it failed to identify that INSTITUTE NAME is simply a modified version of 'name'.

A little carefulness on the users' end Although we don't expect it but if the user place the query a little bit carefully watching the schema or having what's actually in the database the SQL generation performance can be increased by a good amount.

The average accuracy for the system is about 85% for the state it is currently in. The system can work with the basic template. The system cannot get aggregation functions and other complex queries just yet. But with a continuous development that it is going through now it can achieve results in those shortcomings in very short period of time. The system requires a boot time of about 36 seconds where it loads modules for corenlp, nltk (wordnet, corpuses) and other modules. After the first run is done the system requires a 6 - 8 seconds window depending on the database size to hash the primary SQL and another 4 to 5 seconds window to generate secondary SQL. The total time to complete one query is about 12 to 15 seconds as of now. This time can be reduced drastically with some optimization steps that are not yet implemented.

5.4 Comparison

This section of the report contains a brief comparison between our proposed system and other already existing systems or proposed methodologies.

No/Zero Tailoring Unlike other systems our system is completely plug and play. Although ln2sql supports using database dumps directly, it doesn't actually work with a live database. It only reads the sql file of a database dumps and generates SQL according to that file. Whereas our system directly use a db connection to analyze and produce result. And quepy requires the

db to accept structured request and reply as such. Although these two promise to work on any given system, after some modification, others are pretty much limited to one specific database.

At the very least something is shown As shown in data sheets in cases of failure to identify conditions of selection clause the system will automatically return a very basic SQL. The outcome of template library is handful in such way. Although this may produce a huge problem where there are lots of data and system returns all of it.

True natural language query input Our system takes any kind of query input. That is the input query is not limited to interrogative sentences or any kind W/H questions.

No direct mapping Input query does not require to contain any keywords from the database directly. This increases the scope of the system in many folds. Other existing systems require direct keyword mapping. That is if there is no mapping entity name present in the query it will not generate any kind of SQL output.

Deeper understanding Unlike other systems our system depends on both schema of the selected database and the input query. The analysis of database schema and hashing from the data in it makes the conditions more accurate while analyzing the input query provides more detailed dependency information.

For deeper comparison please see the table below.

Table 5.4: Feature comparison

Feature	Quepy	LN2SQL	nQuery	Our System
Basic SQL	Yes	Yes	Yes	Yes
Adaptive	No	Yes	No	Yes
Dependency analysis	No	No	Yes	Yes
Mapping	Table Name	Direct	Parsed	Parsed
Multi Table	-	Yes	Yes	No
Manual Tailoring	Yes	No	Yes (little)	No
Date Retrieve	No	No	No	Yes
CSV	No	No	No	Yes

5.5 Summary

Our system implementation and methodology proved to be of success regardless the data types in the databases. This system has the potential to bring about more to the field of study and becoming a complete solution to Natural Language Query interface for SQL based RDBMS. The metrics can drift a little off given the results are based on the test run by us the researchers not general users. Therefore a chance remains intact that the system may still fail to some of the queries that may or may not be complete against the selected database.

To improve the result accuracy, runtime of the system, more functionality lot more works needed to be done. A more to what can be done is discussed on the next chapter of the report.

CHAPTER 6
Summary, Conclusion, Recommendation and Implication for Future Research

6.1 Summary of the Study

Fetching stored data should be easy not complex. Having to learn SQL makes it hard. But having a system that can understand users' requirement from the query in natural language makes the things as easy as having conversation on a daily life. This makes this research appealing to all researchers in Natural Language Processing. If a system can be developed which can interpret into a complex structure like SQL many other fields of research will be opened for the researchers.

This will enable researchers and users and every other people to access data more freely. Therefore making the system a globally helping hand. Although the outcome of our research in the limited time is very limited, the outcome is still very promising. A system that can identify condition from the database makes the outcome dependable rather than only parsing the query itself. Having the NLP parser to finding dependency between phrases makes it easier to identify a better relationships between them. A better dependency understanding means a better understanding of what is wanted based on what. Which is the system is already doing.

Making multiple tables merged into one single table makes the work very easy to run across for finding conditions.

6.2 Conclusions

This undergrad research, although in a very short time, has made the problem perfectly clear and what has been done. We have focused on making the problem scope clear so it serves as a platform for structural extension to this system. This report can be identified as the state of the art literature review as well. The work that has been proposed and done so far shreds a new angle of light with which the problem can be solved more efficiently. The outcome so far is very convincing and can be developed more into a better outcome producing system.

The main objective that has been mentioned for the thesis is achieved and thus that completes the purpose of the research. A true dynamically adaptive system is what the final outcome of

this project. Our system being capable of working in any given database (standalone or csv) can produce output.

6.3 Recommendation

Perfection is always a work in progress, there our proposed system is only at its early stages. Therefore a lot of works can be done to it. This section of the report contains the limitations of current system and a few possible future works. Current system can serve as the base system that can be extended with more features to solve various other related problems or sub-problems to this massive problem.

6.3.1 Limitations

- Current system can only produce very simple SQL queries. That is structured as "SELECT <attribute(s)> FROM <table_name> WHERE <condition(s)>". This is the first limitation that will come into mind while working with the system.
- Thesaurus may often not contain every possible word that may replace the used word in database.
- Database attributes' names need to be parsed in such a way that the query can find a way to be mapped there. This can be considered as a limitation because users will not use what is in the database. Therefore having to naming the attributes properly is, a minor, limitation of the system.
- The system is still very slow in run time. No optimization work has been done so far.
- For larger size of database the hashing process will take a lot of time.
- Database containing data type of `byte` i.e. blob may fail to retrieve data from the database. This is a problem that is arose from the parsing system.

6.3.2 Future works

- **Aggregation Functions** A major feature of SQL based RDBMS is that it can associate a few functionalities in the selection clause that generates more accurate arithmetic or other kinds of result production (i.e. sum, average, count). To find this kind of relational aggregation functions from the given query is a big challenge waiting for our system. If solved the system will definitely cover more of the ground of database systems.

- **Multiple Tables** Real world databases contain multiple table for ease of data representation and storing whereas our system merge them to work. We cannot keep merging every databases for our purpose. Although it may result in better outcome, we have to find solution that will work with both single and multiple table databases live.

- **Better Relationships** A few arithmetic relations like greater than, less than, not more than are not yet implemented. Thus the error in the data sheet of music.db. Implementation of handling random integers and relationships with other phrases on the query with it will make the SQL more accurate.

- **Bigger Template Library** The SQL has a various combinations of SQL which we have represented as template. Increasing the template library to cover many other kinds of SQL is necessary to perfecting the system.

The mentioned are the next steps of improving performance and after implementation of these the system will become a more suitable of open world applications. Optimization of the system is challenge that comes after the system is capable of solving problems at hand. When it is capable of solving then we can think of making it fast and more capable of handling errors.

6.4 Implication for Further Study

This research can be extend to many extents for a complete system that can understand any query given to it in natural language for any given database. The limitations and possible future works mentioned in the report can be a very good starting points. We are still working on the system and will continue to work on the system furthermore for a better and more accurate system. For any researchers that want to follow up the methodology we have proposed can start from where the system currently is. This report is the first step of reproducing current state of the system.

References

[1] "RDBMS Usage Statistics rdbms usage break down." available at <<https://db-engines.com/en/ranking_ categories>> last accessed: 2018/09/17 at 10:25pm

[2] W. WOODS, "The lunar sciences natural language information system : Final report," *BBN Report*, 1972.

[3] M. Brady and R. C. Berwick, eds., *Computational Models of Discourse*. Cambridge, MA, USA: MIT Press, 1983.

[4] G. G. Hendrix, E. D. Sacerdoti, D. Sagalowicz, and J. Slocum, "Developing a natural language interface to complex data," *ACM Transactions on Database Systems (TODS)*, vol. 3, no. 2, pp. 105–147, 1978.

[5] Y. W. Wong, *Learning for semantic parsing using statistical machine translation techniques*. Computer Science Department, University of Texas at Austin, 2005.

[6] A. J. Agrawal and O. Kakde, "Semantic analysis of natural language queries using domain ontology for information access from database," *International Journal of Intelligent Systems and Applications*, vol. 5, no. 12, p. 81, 2013.

[7] F. S. djahantighi, M. Norouzifard, S. H. Davarpanah, and M. H. Shenassa, "Using natural language processing in order to create sql queries," in *2008 International Conference on Computer and Communication Engineering*, pp. 600–604, May 2008.

[8] 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.

[9] P. P Chaudhari, "Natural language statement to sql query translator," *International Journal of Computer Applications*, vol. 82, no. 5, pp. 18–22, 2013.

[10] F. Reinaldha and T. E. Widagdo, "Natural language interfaces to database (nlidb): Question handling and unit conversion," in *2014 International Conference on Data and Software Engineering (ICODSE)*, pp. 1–6, Nov 2014.

[11] N. Sukthankar, S. Maharnawar, P. Deshmukh, Y. Haribhakta, and V. Kamble, "nquery-a natural language statement to sql query generator," in *Proceedings of ACL 2017, Student Research Workshop*, pp. 17–23, 2017.

[12] I. Androutsopoulos, G. Ritchie, and P. Thanisch, "Masque/sql an e cient and portable natural language query interface for relational databases," *Database technical paper, Department of AI, University of Edinburgh*, 1993.

[13] Quepy, “Quepy: A Python framework to transform natural language questions to queries.” available at <<http: //quepy.machinalis.com/>> last accessed 26- September -2018 at 10:25 pm.

[14] Stanford, “Stanford Core NLP.” Available at <<https://stanfordnlp.github.io/CoreNLP/>> last accessed 21- September -2018 at 10:25 pm.

[15] SQLite, “SQLite.” Available at <<https://www.sqlite.org/index.html>> last accessed 21 September 2018 at 10:25 pm

[16] WordNet, “WordNet Interface for NLTK.” Available at <<http://www.nltk.org/howto/wordnet.html>> last accessed 21- September -2018 at 10:25 pm.

Appendix: A

CoreNLP by Stanford

CoreNLP is a natural language processing toolkit developed by Stanford university researchers. This is one of the most vastly used toolkit in the domain. This tool contains all known NLP tools and algorithms. The pipeline can be used in any application that requires the help of the library. It is developed in JAVA but can be used via wrapper in many other languages. In our case we used it with a Python module called StanfordCoreNLP. The library requires a corpus of the used language, which is English in our case. The language packages contains, words, synonyms, grammatical structures for the library to learn from.

Appendix: B

Costly System

The system we proposed is in nature very computationally costly. The libraries used in the system requires a very high end work station. Therefore it is not recommended yet to run on a low end system for every user. Rather a server to provide services to many people may be more viable.

Appendix: C

Reason of web based interface

Nowadays all systems are becoming more cloud based service rather than on locally running system. It makes the system more available to people easily.

Appendix: D

Open Source Repository

This repository is still under working therefore is to remain private until a usable stable version is released which is expected to be soon.
Link: https://github.com/mehedi-shafi/NLQ2SQL