Natural Language Query to SQL: Dissecting the semantic word dependency graph

BY

MAHMUD SAJJAD ABEER
ID: 161-15-6764

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

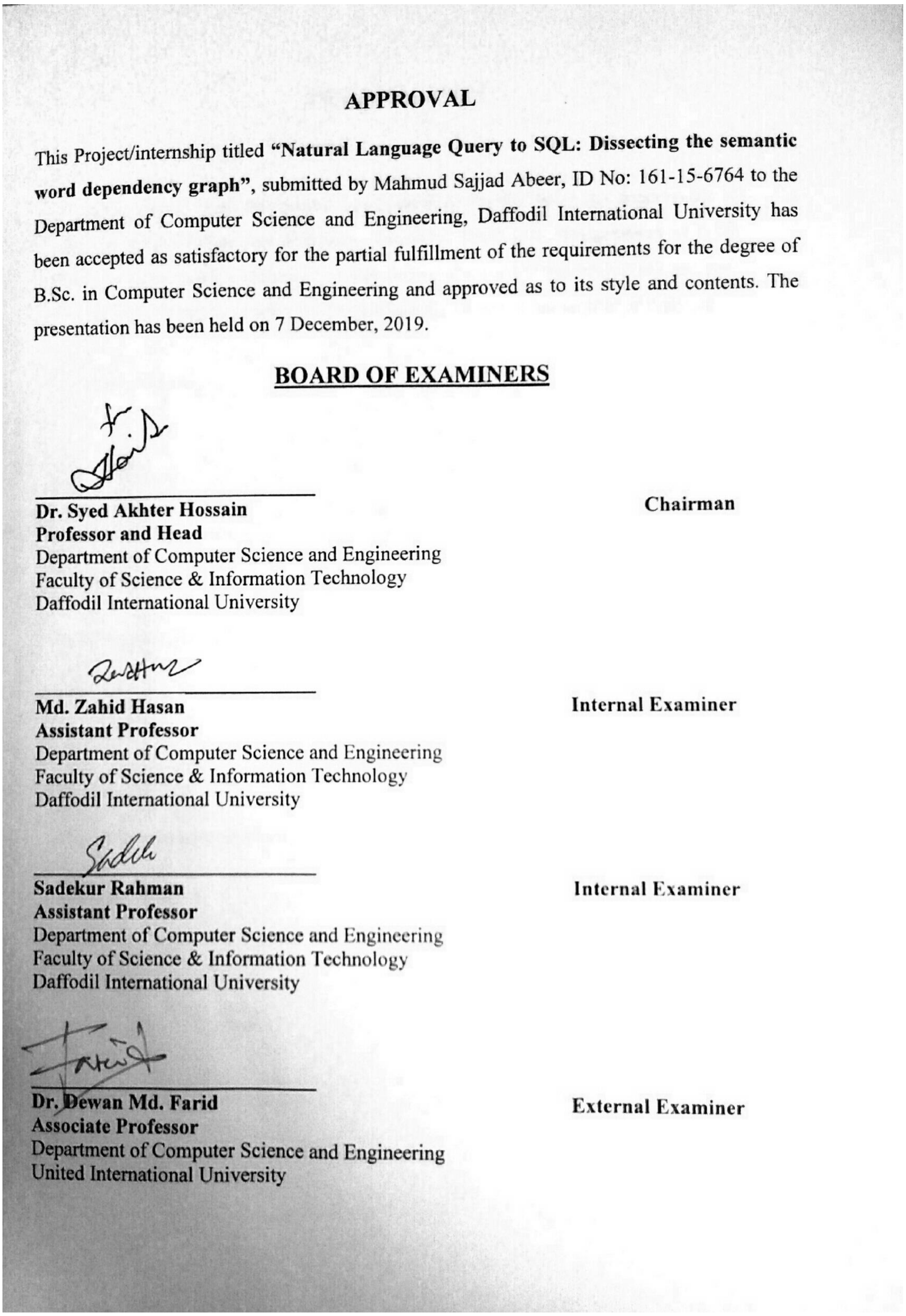
Co-Supervised By

Mr. Saiful Islam
Senior Lecturer
Department of CSE
Daffodil International University

DAFFODIL INTERNATIONAL UNIVERSITY

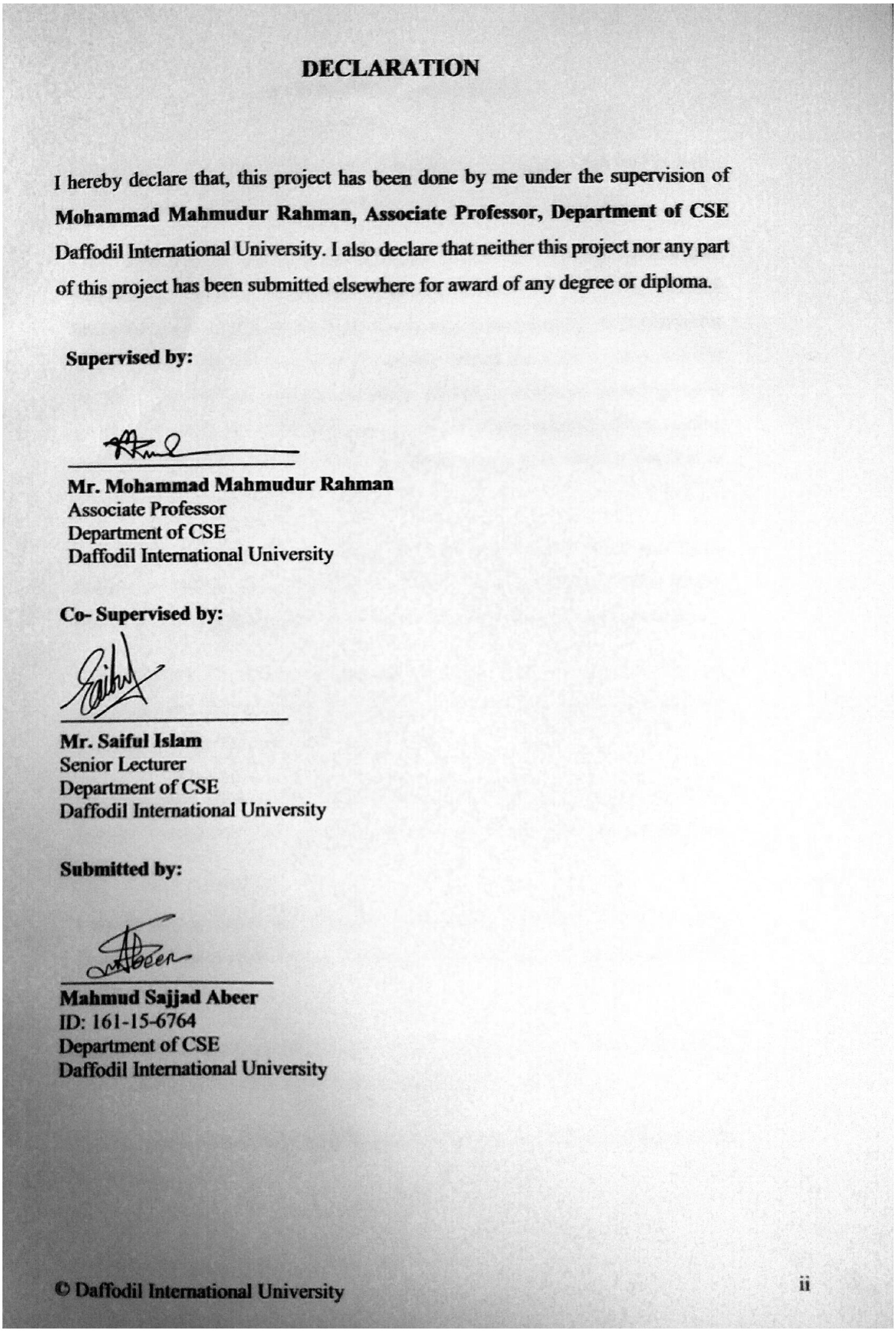
DHAKA, BANGLADESH

DECEMBER, 2019



APPROVAL

This Project/internship titled **"Natural Language Query to SQL: Dissecting the semantic word dependency graph"**, submitted by Mahmud Sajjad Abeer, ID No: 161-15-6764 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 7 December, 2019.



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:

[illegible]

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

Co- Supervised by:

[illegible]

Mr. Saiful Islam
Senior Lecturer
Department of CSE
Daffodil International University

Submitted by:

[illegible]

Mahmud Sajjad Abeer
ID: 161-15-6764
Department of CSE
Daffodil International University

© Daffodil International University ii

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**, Senior Lecturer, Department of CSE, Daffodil International University, for his continuous help and support during the entire time.

I would like to thank **Mr. Mehedi Imam Shafi,** Research and Development Engineer, CodeMarshal, for his continuous help, support and guidance from the beginning to end.

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

Data and Information is the core of Science and Technology. The more we are progressing the more data we are contributing to this vault which will be accessed by millions of people for further learning and uses. Now, the problem is, they are not as versatile as we expect them to be. The amount of information accessible currently is way past the wild estimate of a couple of years prior. No wonder that the digital data is stored in Computational Databases. These data are stored, accessed, updated and retrieved using a complex programming language which is in most of the situation SQL(Structured Query Language) for efficiency and it's applications. The problem is that we need to learn specific SQL languages to access those data even if it's for searching or researching purposes. This is the hardest part. We're creating a huge gap between the data and regular human. What if we don't need to learn any database query languages anymore to retrieve data from databases? We could create an intermediate medium that would capture the natural language query from human, generate the relevant SQL query and retrieve the expected data. This is a continuation of another research where this report has itemized understanding on a different kind of approach where we'd breakdown the words, classify it's internal characteristics and relationships with other sub contexts and iterate over it's dependency graph to determine the selection clauses and column matchings. Also, we will have a good understanding of what have just been done in this particular theme but focusing on the aggregation queries.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION

1.1 Introduction

Current world is all about data and information. The progress we are making right now is because we're reusing the past learnings which are knowledge based on information. We take all that information for granted but we didn't stop right there. We're still working on and making new findings for future generation. This is a matter of realization that knowledge, information they all are basically made of data. Most of them were and are kept in books for centuries but what about our current world? We won't be able to cope with the speed of progress if we had to store, process and retrieve data from hard written books.

Thus, we use computational methods to do all these with best possible efficiency where the DBMS(Database Management System), RDBMS(Relational Database Management System) comes in. This is fantastic and mind blowing how fast we can work with them to store, process or retrieve the data from digital databases nowadays. We just need to make the queries using SQL(Structured Query Language).

The problem is, this is not like human language. The queries are highly structured and complex from the perspective of a regular human. So, even if the data retrieval is efficient, we can't communicate with the database in natural language. However, the Natural Language Processing field of computer science deals with such problem but it only works to process the language. If we can combine the methods of Natural Language Processing this seems very much possible to build an intermediate system that would be able to extract the SQL query from a natural language query.

1.2 Motivation

The idea of Natural Language Processing is by itself very interesting to work on. When we're using the term "Natural Language", we're referring to "Human Language". Making a computer understand a natural language is like the dream of human flying in the sky. Although it's been an old area of research, there are many scopes to explore.

However, as computer enthusiast we look for unresolved problems, dig into it and look for new and improved solutions. While learning SQL queries it's not very uncertain to wonder if there exist any system available where we can alter the SQL queries with Natural Language Query.

Looking into past few works I noticed that one of my university graduate Mehedi Imam Shafi has proposed a system naming "Natural Language Query to SQL: Dynamically Adaptive System" regarding this very same topic. I went through the paper and observed his approach to solving the problem and I was very happy to see that there's already an available GUI based system developed. It handled the very basic queries analyzing the query, column names and data. He's proposed a complete system and implemented a segment of his approach which leaves many future scopes to work on regarding this topic. Keeping that in mind I decided to research on some of the future scopes he mentioned and find out a solution that could be used as an extension the previous approach.

1.3 Rationale of the Study

A mentionable number of researchers have and are still working for finding a solution to adapt a system that can convert human language query to structured query language. We'll check out some of them in "related works" section. Most of them targeting a fixed database to improve accuracy but it doesn't solve the actual problem existing in general terms. As we know, it may work good for that specific database it's scope of service is very small. Even though you train up a system, it may work differently affecting the accuracy with a different database. The "Dynamically adaptive system" proposes a nice approach but some segments are proposed nicely but still underdeveloped and the one we're going to focus is aggregation query.

1.4 Research Questions

As we go on with this research we’re going to face some basic questions which must be defined properly before reading any further.

1.4.1 Natural Language Query:

In this context we’re defining the Natural Language Query as a query made by a regular human in his/her native language. For now let’s just suppose it’s English but unfortunately when making queries no matter what the language is, a single query can be written in many human readable forms even in the same language. All of which are considered as unstructured sentences from computational perspective in this scenario.

1.4.2 SQL(Structured Query Language):

Structured Query Language is a programming language developed to manage Relational Database Management Systems. There are available variations regarding different RDBMS but SQL is conventionally considered as the basic. This requires experience and expertise to create highly structured and efficient database queries.

1.4.3 Aggregate Query:

An aggregate query in database management system is used when we need to derive some specific information such as sum, max, min, average etc over a set of entries.

Some common aggregate queries in natural language are mentioned below:-

- What’s the maximum marks in today’s exam?
- Find the average marks.
- Sum of costs

1.5 Expected Outcome

The primary goal of this project is to propose an improved solution specifically for aggregate queries and implementing it with a graphical interface to test the hypothesis and it's results which will be added to the pervious version developed by Mehedi Imam Shafi. However, I'm pointing out the objectives I'll be focusing on to reach an expected state where we'll see: -

- Ability to detect and mark aggregate query with flexibility in choice of words in natural language query.
- Ability to mark general selection clauses like before.
- Ability to find relation between columns and relevant data without checking data entries in database.
- Ability to generate SQL query more time efficiently with/without checking hash of words.

1.6 Report Layout

The report is designed in a way such that readers will get complete understanding of the aggregation query generation approach, how it's designed and implemented currently, the results and what can be done to improve from current state.

Chapter 1 gives the basic understanding of the problem, motivation behind it and the objectives of this research.

In **Chapter 2** we're going to discuss it's background, past related works and challenges regarding this problem.

Chapter 3 will contain the back to back explanation of the techniques and methodology used in this research.

Chapter 4 is all about the implemented system and all of it's functionalities with proper manual and description.

Chapter 5 will mostly contain the analysis and comparison of results.

Chapter 6 will contain the summary, conclusion, constraints and future scopes of the work.

CHAPTER 2

BACKGROUND

2.1 Introduction

Background studies are like ABCs of the research works. Designing a solution that already exists is useless. Many people around the world are researching about new and old stuffs, giving new observations, improvements and results. As any topic Natural Language to Queries is an interesting topic to many. And ideas from other sources are almost always helpful. Even the failures are part of the research. In this chapter we'll look out into some related works.

2.2 Related Works

There have been numerous inquiries about on this particular issue. The two recommendations and frameworks have been made to take care of the issue proficiently. Throughout the previous not many decades a great deal of new approaches and thoughts have been acquainted with illuminate this issue. Each work has its advantages and disadvantages. All proposition, made, have been useful to the pathway for a superior arrangement later on.

Lunar (Woods, 1973) [1] is the main distributed arrangement of such kind. Lunar took client's question in normal language and replied about rocks that were brought from moon. The framework made was kept to its utilization to the database containing data about the brought rocks. In spite of the fact that the work was constrained however it absolutely showed what is conceivable and what help might it be able to be if it's culminated.

Allen J, 1983 [2] has depicted the significance about such sort of framework in his book. In his book he portrayed how the phonetic in software engineering has been changing over the timeframe. The need to see little bits of knowledge from dialects are unmistakably examined here. Different sort of numerical models are presented and how parsing normal language can bring greater adaptability is tended to. This likewise remembers a reasonable clarification for how more extensive and progressively sensible scope of highlights found in human language and to confine the components of program objectives.

Lifer/Ladder structured by Hendrix, 1978 [3] is another usage of such framework which upheld single table from a database. This framework was for Navy ships which could address straightforward inquiries regarding them. This speaks to how an interface to give access to huge group of information circulated over a PC arrange by means of characteristic language can be presented. The work restricted to one explicit table of information that is worked beforehand upon.

System by Rao et al, 2010 [4] have built up a framework good with basic questions alongside fundamental understood inquiries. Though Chaudhuri et al, 2013 covers some accumulation functionalities also. Conglomeration capacities are more intricate than simply recovering information from database. It requires all the more mapping and more profound comprehension of the information inquiries.

A much more up to date work **nQuery**, 2017 [5] has endeavored to make a substantially more proficient and autonomous framework. Their framework centers around fundamentally table, attribute mapping just as clause tagging which permits to produce progressively accurate yield. Our work relies upon both query analyzing and mapping.

A language MASQUE/SQL [6] was proposed, an update to the past MASQUE, by I. Androutsopoulos, G. Ritchie, P. Thanisch to acknowledge normal language queries from clients. The framework maps the language to an auxiliary contingent base named LQL which at that point is changed over to SQL query. Our framework maps straightforwardly to SQL with no new dialect.

Natural Language Query to SQL, **Dynamically Adaptive System(**Mehedi Imam Shafi**,** 2018) [7], a system that's proposed and developed to a basic stage to take user queries, do some nlp operations directly on the query and finally mapping the columns and data that fits the users query to a reasonable stage to convert it to a SQL query. However it didn't handle the aggregation queries which we're going to work on in this research.

Rule-SQL(Tong Guo, Huilin Gao, 2019) [8] is one of such research works done recently. Offered the query and the results from the database table without the SQL rationale structure, RuleSQL utilizes the principles dependent on table segment names and question string for the SQL investigation first and afterward utilizes the investigated SQL for supervised training.

2.3 Research Summary

Researching above mentioned related works this is clear that the problem still exists in some manner. Some works are actually good and helped this research to open up some ideas. As we can see there are only few works on aggregation methods. We've also found some solutions working nice with neural network but in terms of query analyzing it's not pretty much clearing up the natural language processing. In our system we're analyzing the query and through deep understanding of the query finding a solution that most probably will open up more options to improve but will also give a reasonable approach to solve aggregate query related problems.

2.4 Scope of the problem

This research is going to act as an extension or continuation of Natural Language Query to SQL, Dynamically Adaptive System. The previous research gave an amazing approach with a simple implementation that can already work with simple queries but can't handle aggregation queries and join queries which requires a different kind of approach to solve. In this research we're only going to focus how aggregation queries can be analyzed to find a relevant SQL query.

2.5 Challenges

First of all, I'd like to acknowledge that the previous solution does most of the web-app implementation part but the research regarding aggregation queries is going to be very much different than that version.

Natural Language by itself is already very complex. Let's say for example we need to know the number of students with highest marks, one can ask "Give me highest marks?" one might ask "What's the maximum marks a student got?" both the queries will produce the same result but if you look closer you'll notice that the structure of the queries is in no way same. So, we're going to analyze the query properly so that both of them produce the same results and that's a challenge to look out.

Some words in the query may not hold any significance in terms of the actual SQL query but you can't tell the users that you can't use them in your natural language queries. This one is a problem we can't get rid of and rather adapt to it otherwise we're limiting the natural language as well as creating new structure to follow which leaves us back into the same problem we are trying to solve.

Along the way we'll see few more challenges and how we approached to solve them.

CHAPTER 3

RESEARCH METHODOLOGY

3.1 Introduction

We've explored many of the recent approaches and after researching all those methodologies we have adapted every one of the advantages of approaches and their constraints and extensions also. Along these lines our proposed system incorporates a review of what is done to avert what or to broaden what ability. However, this part of the report holds all the steps used for classifying the problems, data collection and research methodologies used to find a solution for aggregate queries.

3.2 Research Subject and Instrumentation

Domain: This should already be very much clear that this research lies solely on the natural language processing domain. We're going to use many of the tools that are already available. As of the previous we're sticking to English language. We're going to take the basic SQL query structures as it's almost of the same figure as others and conventionally followed for starters.

Instrumentation: In our research we're going to need some of the tools and instruments that were already used on previous solution for similar purposes. However, the methods are mentioned below: -

1. Tokenization
2. POS Tagging
3. Lemmatizing/Stemming
4. Dependency Parsing

You can find the details online. Why we needed these tools will be explained on later subsections where needed.

3.3 Data Collection

In our research we mostly relied on some well-established sources. For our specific scope of research, we had to do some manual work on building some dataset. We're testing on the similar test data used on the previous solution. So, we're only sharing the new Development Data that we've used in this project.

3.3.1 Development Data

As we're working on aggregation function, we needed to understand how one might ask max, min etc in natural language. One might ask "What's the largest size of boxes?". As you can see, he's asking for the maximum or MAX size of the boxes but saying "largest" in his query. Same for "smallest" which maps to MIN. We had to collect all such words (superlative degrees) and map it to the related aggregation. This could be done with other learning procedures which can be taken as further improvements on current solution.

For that we've collected the data from **List of comparative superlative words** [9] and then manually map the aggregation relation which you'll find on the *agg_functions.txt* on the project files.

Rest of the test databases were collected from the open source project of NLQ2SQL: Dynamic Adaptive Approach [10].

Data Source: All the information utilized all throughout the research works are from open sources and were ensured no copyright was disregarded. Libraries that are utilized are likewise under open source area in this way information accompanied them are additionally dependent upon their copyright and under the open source space.

3.4 Detailed Methodology

In this section we're going to see the detailed methodology we followed in current approach and implementation.

3.4.1 Overview of the System

The system is running in linear configuration except for the part where we're looking for relationships between data and columns/selection clauses. The system can be divided into some smaller subtasks and are tried to explain through a flow chart given below: -

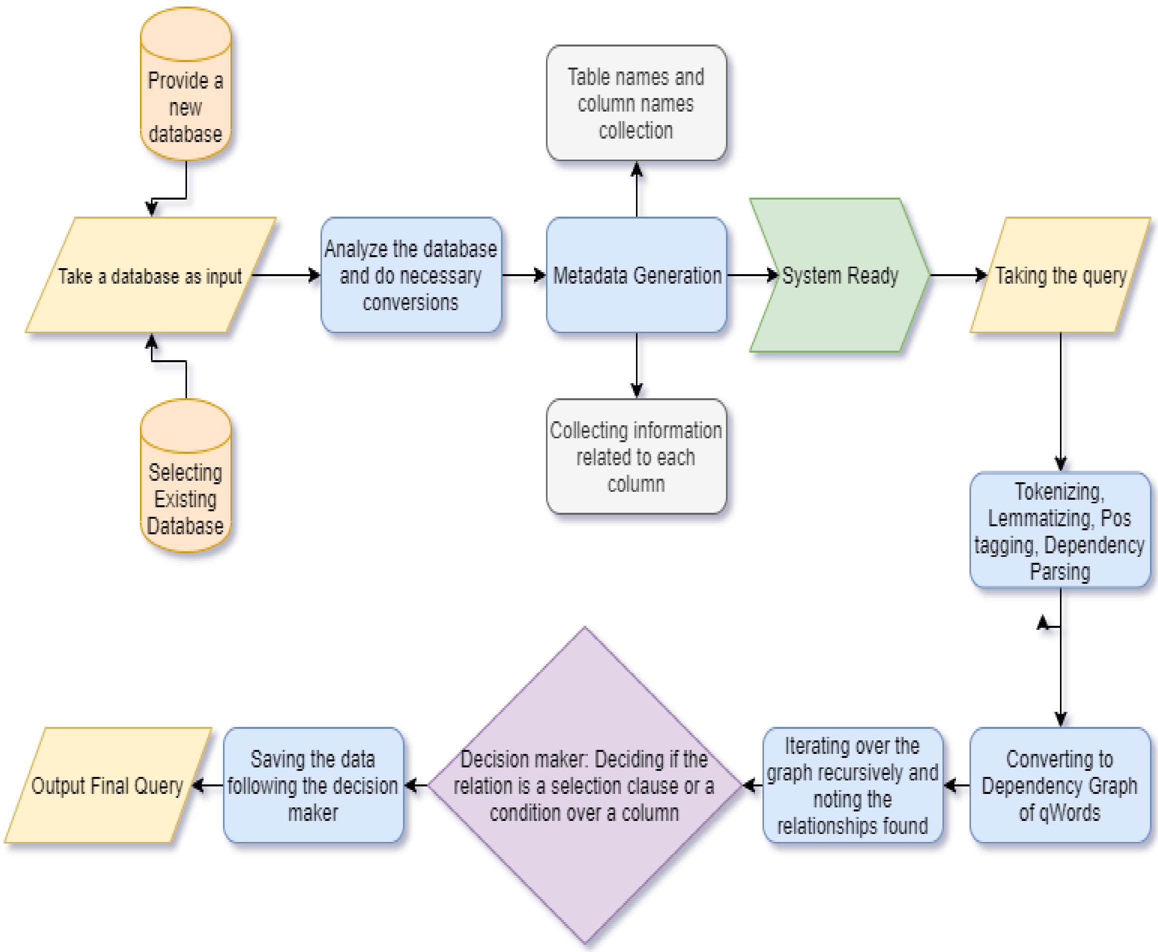

Provide a new database
Table names and column names collection
Take a database as input
Analyze the database and do necessary conversions
Metadata Generation
System Ready
Taking the query
Selecting Existing Database
Collecting information related to each column
Tokenizing, Lemmatizing, Pos tagging, Dependency Parsing
Output Final Query
Saving the data following the decision maker
Decision maker: Deciding if the relation is a selection clause or a condition over a column
Iterating over the graph recursively and noting the relationships found
Converting to Dependency Graph of qWords

Figure 3.1 System Figure

3.4.2 Detailed steps

1. **Table names and column names collection:** In our approach we're only going to use table name and column names from the given database. No relationship with database data is going to be mapped like the previous approach.
   This is surely going to improve the speed but may lessen some accuracy in some fields. However, the actual mode we've planned at initial phase of research had some analysis on the table data and how it can be used to improve the results. This can be taken as future improvements of current approach.
   In this research we're going to keep a version like the previous one where these data would be collected as meta-data of .db file after selecting/dumping a database.

2. **NLP annotation:** This phase is ran after taking the natural language query. In this phase we're using StanfordCoreNLP to collect the tokens, lemmas, pos tags and Dependency graph over the sentence and words.
   In our approach Dependency Parsing plays a significant role. We've tried other dependency parsers where spaCy(done in later) dependency parser did very well comparing to StanfordCoreNLP but in some cases spaCy was tokenizing it in such a way that we found it hard to implement. Other than that both of them had high accuracy but we chose to go with the StanfordCoreNLP for some advantages in our implementation.

   i.e: When we tried out the dependency parse for query "All details of 171-15-9400"

   The spaCy [11] found: -

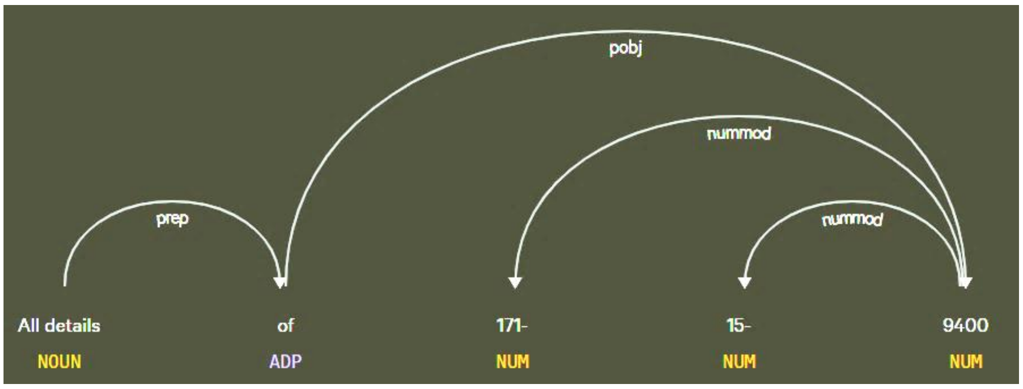


Figure 3.2 spaCy dependency parser

Whereas, StanfordCoreNLP [12] found: -

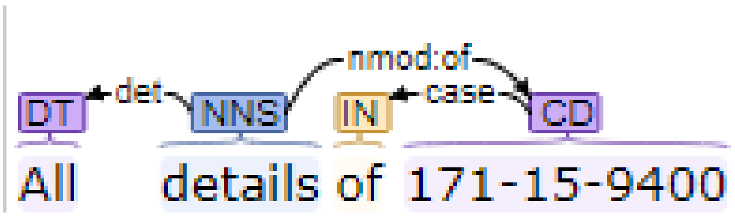


Figure 3.3 StanfordCoreNLP dependency parser

3. **Converting to qWords:** The words individually holds many characteristics. Combining all of them and putting them altogether was necessary. Which also helps to find relations with other words when we'll be looking for it in details. The object model will be discussed on Back-End subsection on 4th chapter[read the note there].

4. **Creating graph of qWords:** We're going to join all the qWords back again following the dependency parsing rules. Theoretically, this is no big deal but while implementing a solution to it, we had to create a new graph to include additional information for future iteration over it.

5. **Iterating on graph and finding condition and selection clauses:** We're iterating over the graph assuming it as a sentence. If a leaf node word is found we're trying to find a selection clause or a column name matching.

Here we're checking the parents of those words recursively as they hold some dependencies. We're also doing some checkups to detect if it's a column match, data match, stopword, selection clause or some aggregate query.
Here, by "stopword" we mean the type of words that holds no major significance when doing SQL queries. There exists a drawback when we're checking if it's a stopword or not. We may have selected a word that actually held significance. For example:-

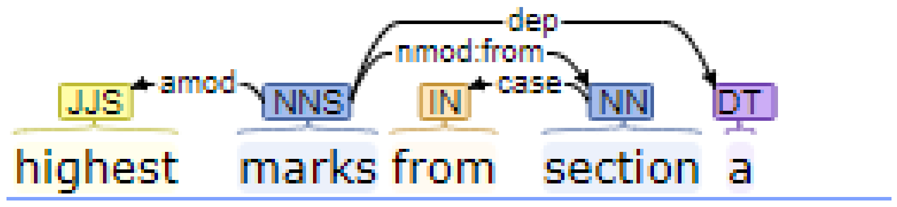


Figure 3.4 Lack of relative accuracy in stanfordCoreNLP

In above figure, you'll notice that "a" is marked as DT(Determiner) which it's actually not. However, in below example we'll see that changing the case of A changes it's pos tag:-

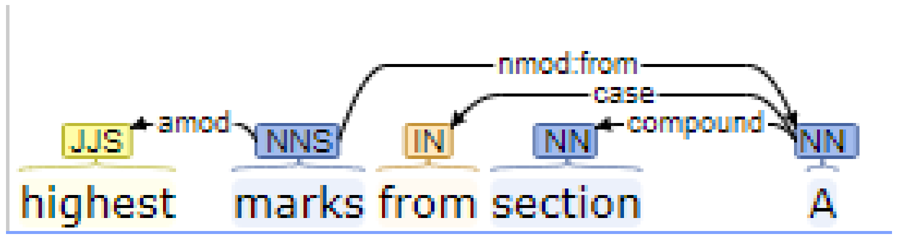


Figure 3.5 Change of pos tag after changing the casing

However, as we're working with StanfordCoreNLP, we must follow it's pos tags. Which although can be improved by using other learning methods and not covered in this research as we've reached a good level of accuracy following current method.
This is because of the linguistic significance of capitalization. And overcoming such problem is not yet proposed/implemented.

3.5 Implementation Requirements

We’re pretty much using the similar requirements as mentioned on the previous approach proposed by Mehedi Imam Shafi. The recommended system requirement based on the environment we tested our new approach in is given below.

Hardware:

Processor:

Core i3 CPU @ 3.30GHz or above

RAM:

6.00 GB or above

Storage:

20 GB of Hard Disk Drive or above

Software Requirement:

Operating Systems:

- Windows OS – Windows 10 (Pro)
- Linux – Ubuntu 18.04

Either of them should be fine

Environments:

- Anaconda
- Python 3.7 or above

Python Packages:

- NLTK
- Stanford Core-NLP
- WordNet
- Messy Tables
- Pandas
- Json

Also, it must have internet connectivity available to function properly.

CHAPTER 4

SYSTEM IMPLEMENTATION

4.1 Introduction

In real life the theoretical findings often works differently than expected. So, although we're basically working to develop a methodology to solve our mentioned problem we're going to develop and implement our idea through a work friendly system. This chapter will contain the detailed in and out report of the developed system. It will also contain the details of input parameters and methods required to run the system.

4.2 Overview

Implementing a basic solution for testing purposes was one of the goals mentioned before. As of the previous researcher of this project, python seemed easy to catch up with keeping the previous version alive. This version is going to be almost similar to the previous one as I've mentioned that it's only going to be an extension of it. For simplicity I'm still going to point out the basic functionalities and implementation for the upgraded version.

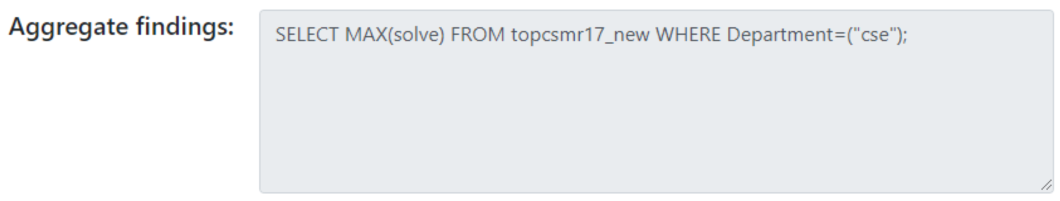
The system is implemented in basic two parts. Back end and front end.

The front end will interact with user and the back end will do all the processing. The detailed description is shared below.

4.3 Front End

The user interface is going to be almost exactly the same as the version we've mentioned earlier. Big thanks to Mr. Shafi for letting me create a git branch of his private project in github [7]. We're just going to add an extra field in that same interface. He already has the details about most of the parts and we're just going to explain the extension we've included.

Aggregation SQL findings:



```
SELECT MAX(solve) FROM topcsmr17_new WHERE Department=("cse");
```

Figure 4.1 Aggregate query findings

After generating the query, if it finds any aggregate query it will be shown on the grey field of "Aggregate findings" although this works for simple queries as well.

4.4 Back End

Language: We've used python and it's environment in it's back end. The reason why we chose this language is because python has a very rich set of libraries for data analysis and in this case Natural Language Processing. The libraries/packages we've used for this project is already mentioned in the Implementation Requirement subsection.

The python version we've used here is 3.7.0

Framework: Flask framework is used to maintain the webserver and connections. We actually had two mentionable options to implement it based on python. Django or Flask. Django enforces the developer to maintain MVC model and requires heavy coding. Which really is good for production level developments. Whereas, Flask offers a very lightweight development module, easy to learn, use, develop and deploy which as a researcher anyone would find interesting.

We've used flask version 1.1.1

Easy to use functional program: Any researcher would find the project very easy to use.

Any user having basic understanding of programming can follow the user guideline and use it for further research.

User guideline with cmd/terminal:

1. First, install all the dependencies including environment if required. Anaconda would lessen the work in case of basic environment. After that, just install the required packages mentioned on 3.5 subsection. Usually pip(Package manager for Python packages) is used to install the packages.
2. Open a terminal in your operating system
3. Change the directory to the targeted project folder
4. Open python IDLE in that directory. Make sure it's the correct version of python opened.
5. Import abeerQuery
6. Create an object of type aggQueryFinder.
7. Call ob.updateColumnNames(columns) where columns is the orderwise list of string of column names.
8. Call ob.query(qq) where qq is the query string from user which will return an object of type Sql from qClasses(For further use)
9. Call getSqlString(table_name) to get the final SQL string
10. After that if you write print(result), it will give you the Sql query found from this natural language query.

For example:

```
import abeerQuery as aq
ob = aq.aggQueryFinder()
ob.updateColumnNames(["name", "department", "section", "solve"])
ob.query("find name and solve from cse department")
result = ob.getSqlString("topcsmr17")
print(result)
```

Class attributes and methods: The class names, attributes and methods are mentioned below. The attribute names and method names are very much definitive and some of which are mentioned and explained in previous and next sections.

The class diagram of the new implementation is shared below:

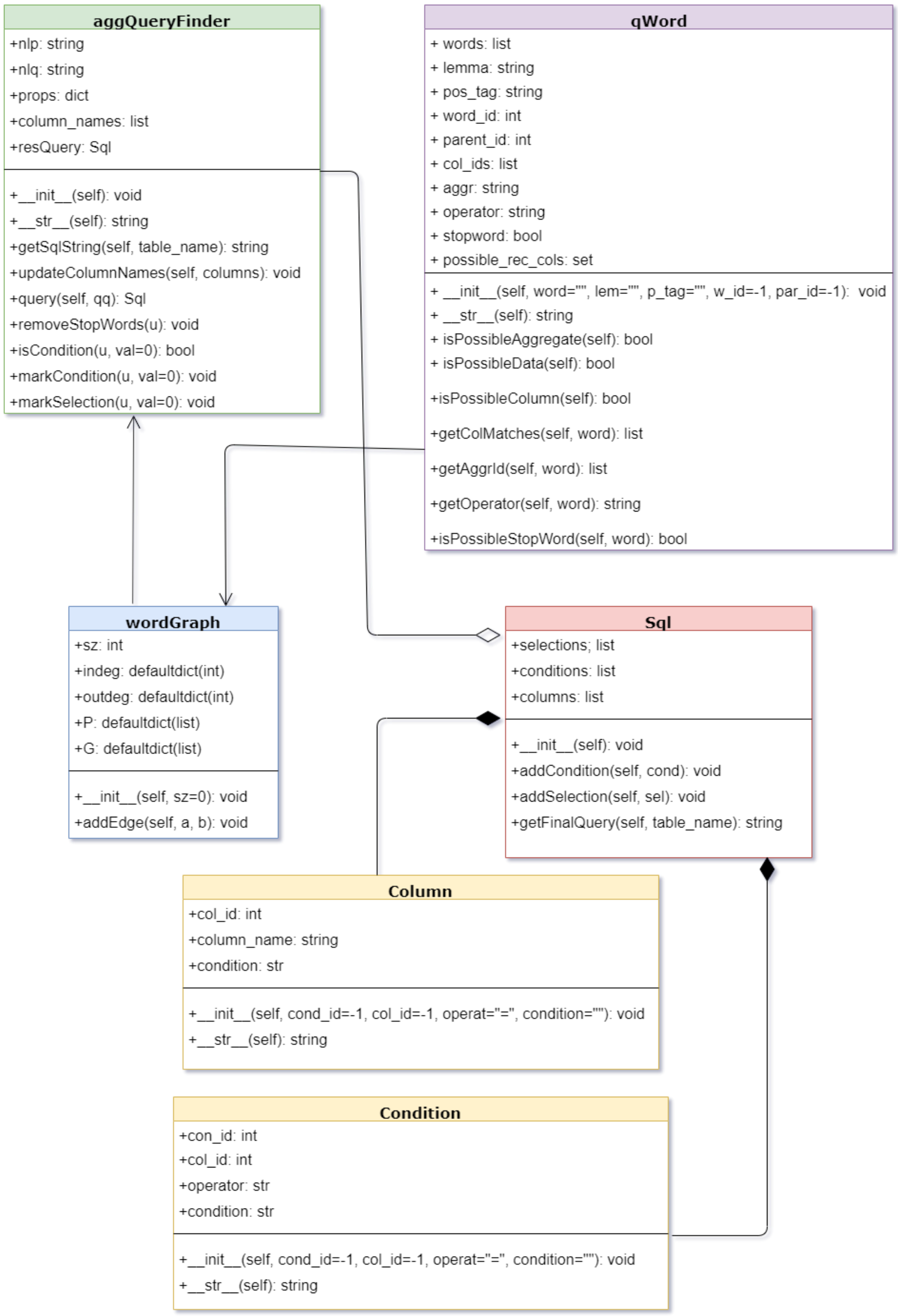

aggQueryFinder
+nlp: string
+nlq: string
+props: dict
+column_names: list
+resQuery: Sql
+__init__(self): void
+__str__(self): string
+getSqlString(self, table_name): string
+updateColumnNames(self, columns): void
+query(self, qq): Sql
+removeStopWords(u): void
+isCondition(u, val=0): bool
+markCondition(u, val=0): void
+markSelection(u, val=0): void
qWord
+ words: list
+ lemma: string
+ pos_tag: string
+ word_id: int
+ parent_id: int
+ col_ids: list
+ aggr: string
+ operator: string
+ stopword: bool
+ possible_rec_cols: set
+ __init__(self, word="", lem="", p_tag="", w_id=-1, par_id=-1): void
+ __str__(self): string
+ isPossibleAggregate(self): bool
+ isPossibleData(self): bool
+isPossibleColumn(self): bool
+getColMatches(self, word): list
+getAggrId(self, word): list
+getOperator(self, word): string
+isPossibleStopWord(self, word): bool
wordGraph
+sz: int
+indeg: defaultdict(int)
+outdeg: defaultdict(int)
+P: defaultdict(list)
+G: defaultdict(list)
+__init__(self, sz=0): void
+addEdge(self, a, b): void
Sql
+selections; list
+conditions: list
+columns: list
+__init__(self): void
+addCondition(self, cond): void
+addSelection(self, sel): void
+getFinalQuery(self, table_name): string
Column
+col_id: int
+column_name: string
+condition: str
+__init__(self, cond_id=-1, col_id=-1, operat="=", condition=""): void
+__str__(self): string
Condition
+con_id: int
+col_id: int
+operator: str
+condition: str
+__init__(self, cond_id=-1, col_id=-1, operat="=", condition=""): void
+__str__(self): string

Figure 4.2 Class diagram

Here, some of the mentionable classes and it's uses are given below:-

aggQueryFinder: This class is the driver class of the whole system.

qWord: qWord is the class used to store all the information related to the characteristics of the words.

wordGraph: This only holds the relationship between the words.

Sql: This class can be called as the resulting sql class. This holds all the information as column matches and selection clauses.

All of the mentioned class, attributes and methods are new and holds no direct dependency on the previous solution but for future improvements one may take it into account.

4.5 Connection

Although we've brought an extension into account, the connection remains the same as previous implementation and we didn't to change any of it's structure as it gets the job done. However, it's not much hard to get it working.

To make frontend cooperate with the backend we utilized ajax as the medium. Ajax system gives us a chance to make demand among applications and procedure approaching information. When something is told in the frontend an ajax demand is made to the server. Furthermore, the server processes the solicitation as needs be and when preparing is done it restores an outcome to the user end.

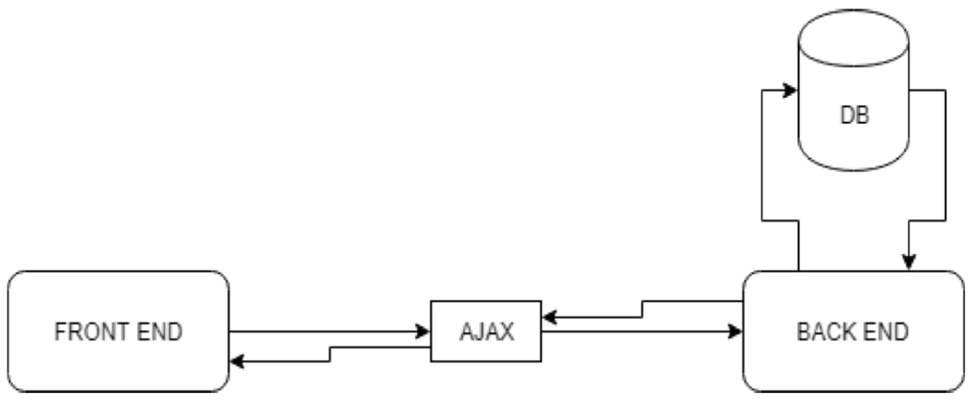


Figure 4.3 Connection Diagram

CHAPTER 5

EXPERIMENTAL RESULTS AND DISCUSSION

5.1 Introduction

So far, we’ve been through the theoretical idea and implementation details of this project. Now, it’s time to do some experiments with some hand-made queries. In this chapter we’ll checkout some hand-made queries on different databases, measure the expected accuracy and compare it with other available systems.

5.2 Experimental Results

In this section we’ll checkout some experimental queries and their results. As we’ve mentioned earlier that we’re doing no data-oriented matching and is out of the scope of this report, we’ll be providing the column name in some manner to test the current system.

The below experiments are done on **topcsmr17.csv** file:-

Table 5.1 Experimental results from topcsmr17.csv

#	Query in natural language	System output	Correct
1	find the max solve from section A	SELECT MAX(solve) FROM topcsmr17 WHERE Section=("C");	YES
2	find the max solve from a section	SELECT Section, MAX(solve) FROM topcsmr17_new;	NO
3	contact number of student with token 14	SELECT Contact FROM topcsmr17 WHERE Token=("14");	YES
4	highest solve of students from department of swe and section C	SELECT MAX(solve) FROM topcsmr17 WHERE Section=("C") AND Department=("swe");	YES
5	name of participant with id 161-15-6764	SELECT Participant FROM topcsmr17 WHERE id=("161-15-6764");	YES

6	lowest solve from day shift	SELECT MIN(solve) FROM topcsmr17 WHERE shift=("day");	YES
7	show me all the names and ids from cse department	SELECT * FROM topcsmr17_new WHERE Department=("cse");	NO
8	name and id from cse department in day shift	SELECT Varsity_ID, Participant_Name FROM topcsmr17 WHERE shift=("day") AND Department=("cse");	YES
9	everyone from cse department in evening shift	SELECT * FROM topcsmr17 WHERE shift=("evening") AND Department=("cse");	YES
10	count of participants having 0 solve	SELECT COUNT(*) FROM topcsmr17 WHERE solve=("0");	YES
11	average solve among all participants	SELECT AVG(solve) FROM topcsmr17;	YES

The table below holds the experimental results on **music.db**

Table 5.2 Experimental results from music.db

#	**Query in natural language**	**System output**	**Correct**
1	who sang you shook me	SELECT * FROM tracks;	NO
2	songs having 25000 length	SELECT * FROM tracks WHERE length=("25000");	YES
3	Show all artists from pop genre	SELECT artist FROM tracks WHERE genre=("pop");	YES
4	longest length of a song from rock genre	SELECT MAX(length) FROM tracks WHERE genre=("rock");	YES
5	count songs in nothingman album	SELECT COUNT(*) FROM tracks WHERE album=("nothingman");	YES
6	minimum length of song where artist is MJ	SELECT MIN(length) FROM tracks WHERE artist=("MJ");	YES
7	Show id and artist from pop genre	SELECT artist FROM tracks WHERE genre=("pop");	NO

5.3 Descriptive Analysis

After doing some experiments on different databases we can say that we're pretty satisfied about the results considering our proposed goal.

As we said earlier, we're going to experiment following the constraints we've clarified so that we can compare if we did good enough to implement a solution that meets the expected requirements.

We're going to analyze what we could achieve and what not through some reasonable questions in the context of our research.

5.3.1 Does it support basic aggregation queries correctly?

- **Yes**, it does with many variations. From above examples you'll see that MAX, MIN, AVG, COUNT is working perfectly fine. Also, all of them chose the correct column names which was much complex from General Point of View while working with natural language. The column name could be placed anywhere in different figure of speeches and may contain many unnecessary words in between. We could successfully ignore that in our approach. However, it's interesting how "largest", "smallest" words are converted to MAX or MIN according to the context. We have such a good library to handle such aggregation functions in our solution that resulted to this queries. This was the main goal of our research and we're happy to see that it's working.

5.3.2 Does it support multiple selection clauses?

- **Yes.** #8 from table 5.1 is a good example. It's very much clear that multiple selection clauses is working just the way it should be. You'd also notice that in this query it supported matching with segment of column name which is impressive although in our current solution there's a space of improvement in this area.

5.3.3 Does it support multiple column matching?

- **Yes.** In #4 and #9 from table 5.1 we can see that it's working perfectly. Column matching is comparatively easier to select in most of the contexts. In all of the mentioned queries you'll notice that only two of them has mistaken to find the column match among all the queries in table 5.1 and 5.2.

5.3.4 Is it ignoring unnecessary words properly?

- **Yes.** As you should realize, we use many words that holds no significance for SQL queries. "**Show me** all **the** ….", "**Find the** …. **for me**" etc. In our solution if you notice, we had an attribute in **qWord class** named **stopword**. We went through all the pos tags and analyzed their behavior and came up with the type of tags that in most of the cases are stopwords.

5.3.5 Does it support all figure of speech following same context?

- **Not always.** It's working in most of the cases but with some flaws. In #5 from table 5.1 the "a" in "a section" is marked as DT in pos tags whereas "A" is marks as NN. In that context, we couldn't manage to decide if it's a valuable word for the final query or not.

  In #8 from table 5.1 is working but #7 from table 5.2 is surely not working well even though the text is almost similar.

5.3.6 Would it work properly when column name doesn't match exactly?

- **No, not in current implementation.** It'd work for aggregation functions but we haven't enriched the thesaurus for each words in other contexts. So, this won't work now but we're planning to improve this problem very soon afterwards.

5.3.7 Does it support basic queries excluding aggregate queries as of previous?

- **Yes**, but in our current implementation, only following the selected constraints. Considering that we're not matching the query with data entries and expecting that users would usually mention the column names in some manner this would give very good results. In the #1 query in table 5.2 which I've collected from the pervious paper is a good example. The query was "who sang you shook me". Here as you can see only the name of the song is mentioned "you shook me" and we failed to solve it as current implementation is not looking for data match in databases.

  However, although data matching is important it's also important that if the user is asking a song that's not in the database the query is still created correctly.

5.4 Comparison

In this section of the report we will compare our proposed solution with other existing systems and methodologies.

Aggregation support: On the previous research regarding this topic, it had no aggregation function support. In most of the researches done before, very few focused particularly on aggregation query.

Selection clause determination: On the previous research the selection clause was highly dependent on the dependency root/head. As they analyzed and found that in most of the cases the root holds the selection clause but during our research we were convinced enough that through our approach it won't necessarily be depending on the root. It will try to consider and analyze other nodes in dependency graph which has given better results in our approach.

Multiple selection clause support: On the previous approach it could determine at best one selection clause and that being the root of the sentence. In our approach as we are considering to iterate over the whole graph and trying to determine selection clauses based on the relationships between words, we've been able to detect multiple selection clauses through the process.

Time efficiency: Not very important from current research perspective and if we could integrate the relevance of data with the natural language query as planned, the implementation would have been slower. However, as this is what we came up with now, it's fair to mention that we've already took it to a reasonable stage and it works super-fast comparing to the previous system as we are just collecting meta-data of the database.

5.5 Summary

In this research we have tried to improve the generation of selection clauses and column matchings and the experimented result implies that this is possible to reach a certain level of standard to determine the selection clauses wisely analyzing the dependency graph.

According to the study, this is a very satisfactory result under the given constraints. This is a clear indication that the way we approached is doable and we've already reached a level of success.

As we all know, there's always a room for improvement, integrating the relationship with database entries could improve the current performance and would require further research for maximum throughput.

CHAPTER 6

SUMMARY, CONCLUSION, RECOMMENDATION AND IMPLICATION FOR FUTURE RESEARCH

6.1 Summary of the Study

Before this research we had a solution that worked for simple queries pretty well but not all types of query. The previous researcher on this area mentioned some of the future scopes to work on and it’s a pleasure to work on one of the significant ones and we surely took it to another higher level.

It’s fascinating to think that now we’ve a solution that can answer aggregate queries as well. The way we approached this problem is new in this area of researched. There are many areas to improve even in current approach but we’ve taken it to a reasonable stage and it now can answer many basic queries and aggregate queries which we didn’t have earlier.

6.2 Conclusions

The scope of the actual research is very large and may take years to perfect. Taking that into consideration we had to select a subtask which we could work on. We had to clear our scope of research to concentrate and improve from current stage. After a long discussion we chose the most needed subtask after the previous version of work.

According to the analysis and results we’ve found we can now say that we’ve completed the main objectives that we’ve mentioned earlier and so it completes the objectives of this research.

6.3 Recommendation

No system is 100% perfect. It has to go through continuous research and development to overcome newer challenges having it’s own limitation, constraints and scope to improve. Here, we'll get to know some limitations and scope of future works regarding this research.

6.3.1 Limitations

- Current system can only work on simple aggregate queries. Complex queries will require more complex design and may not give a completely accurate result on this approach which as we mentioned wasn't in our research scope.
- Database column names sometimes can be irrelevant which is a great problem in this approach. We are relying highly on the database managers and their naming skills. The column names must be relevant to the type of data.
- The synonyms of column names can be very uncommon and in that case the expected query may not meet the requirements
- Count query can have group by subparts which aren't handled in this approach and as it requires a complex design it lies out of current scope of research.

6.3.2 Future Works

- **Better header name/word recognition**: In our approach we're not relying on the data and it's entity type. In future one can design a learning method where we can recognize the entity type of the data or words by training the database data. This will make the system slower but will give better results comparing to the current approach.
- **Complex aggregate queries:** So far, we've collected many of the superlative forms to map them with aggregation queries. In natural language one can explain such queries even without mentioning the exact word for it or some comparative explanations or maybe one can try using some count queries grouping by a followed rule.
- **Arithmetic calculations:** One may try to make a complex query where each subquery combined leads to find another mathematical calculation which is also not very rare in database queries.
- **Multiple tables:** This is a commonly used in database queries and also one of the significant scope to work on in future. Inner join, outer join, left join, right

join etc for example. One solution could be to merge all the tables altogether and solve it with simple queries but that leaves many more scopes to discuss and possibly some better approaches will come out in future.

- **Better relationships:** Relationship comparison like less than, greater than, greater than or equal, lesser than or equal etc are not handled yet in previous solution or this solution. This is also an interesting future scope to work on.

6.4 Implications for Further Study

This research can be reached out to numerous degrees for a total framework that can find any queries given to it in human language for some arbitrary databases whether they follow different formats or not. The restrictions and conceivable future works referenced in the report can be generally an excellent scope to continue this research and improve likewise or otherwise. We are as yet taking a shot at the framework and will keep on chipping away at the framework moreover for a superior and progressively precise system. For any analysts that need to catch up the strategy we have proposed can begin from where the system right now is. This report is the initial step of imitating current condition of the system.

REFERENCES

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

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

[3] 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, pp. 105-147, 1078.

[4] 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, pp. 219-223, 2010.

[5] 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, 2017, pp. 17-29.

[6] 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.

[7] M. I. Shafi, "Natural Language Query to SQL, Dynamically Adaptive System," Daffodil International University, Dhaka, 2018.

[8] Tong Guo, Huilin Gao, "Using Database Rule for Weak Supervised Text-to-SQL Generation," China Electronic Technology Group Corporation Information Science Academy, Beijing, China, 2019.

[9] "Easy pace learning," [Online]. Available: <<https://www.easypacelearning.com/all-lessons/grammar/1436-comparative-superlative-adjectives-list-from-a-to-z>>. [Accessed 20 May 2019].

[10] M. I. Shafi, "Github | mehedi-shafi," [Online]. Available: <<https://github.com/mehedi-shafi/NLQ2SQL>>. [Accessed 17 June 2019].

[11] "Explosion AI," [Online]. Available: <<https://explosion.ai/demos/displacy/>>. [Accessed 23 August 2019].

[12] "CoreNLP," Stanford, [Online]. Available: <<https://corenlp.run>>. [Accessed 23 August 2019].

APPENDIX

Appendix A:

Dependency Parsing

Phrase structure grammar is worried about how words and successions of words consolidate to shape constituents. A particular and corresponding methodology, dependency grammar, focuses rather around how words identify with different words. Dependency in this context is a relation that holds between a head and its dependents. The most tensed verb is usually considered as the head of a sentence and the other word is either dependent on the head or connects to it through a way of dependencies.

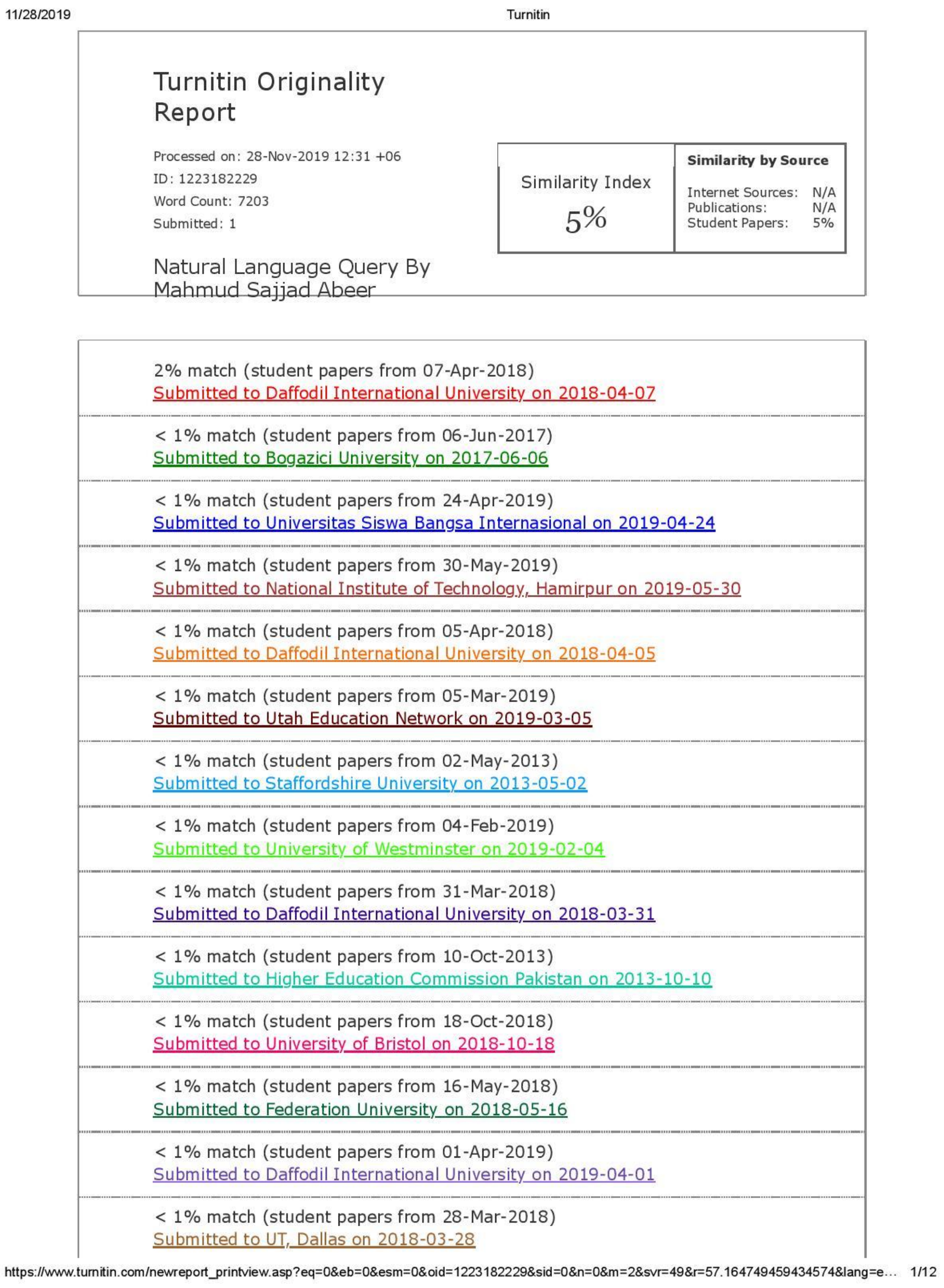
A dependency representation is usually represented as directed graph.

Appendix B:

Open Source Repository

The development is done and managed on github. The project is still unstable and under development. So, the repository is going to remain private to some selected researchers only until a stable version is ready to release. However, we're expecting it to make it public not very far.

Github repo: https://github.com/M-S-Abeer/Natural-Language-to-SQL



11/28/2019 Turnitin

Turnitin Originality Report

Processed on: 28-Nov-2019 12:31 +06
ID: 1223182229
Word Count: 7203
Submitted: 1

Natural Language Query By Mahmud Sajjad Abeer

Similarity Index	Similarity by Source	
5%	Internet Sources:	N/A
	Publications:	N/A
	Student Papers:	5%

2% match (student papers from 07-Apr-2018)
Submitted to Daffodil International University on 2018-04-07

< 1% match (student papers from 06-Jun-2017)
Submitted to Bogazici University on 2017-06-06

< 1% match (student papers from 24-Apr-2019)
Submitted to Universitas Siswa Bangsa Internasional on 2019-04-24

< 1% match (student papers from 30-May-2019)
Submitted to National Institute of Technology, Hamirpur on 2019-05-30

< 1% match (student papers from 05-Apr-2018)
Submitted to Daffodil International University on 2018-04-05

< 1% match (student papers from 05-Mar-2019)
Submitted to Utah Education Network on 2019-03-05

< 1% match (student papers from 02-May-2013)
Submitted to Staffordshire University on 2013-05-02

< 1% match (student papers from 04-Feb-2019)
Submitted to University of Westminster on 2019-02-04

< 1% match (student papers from 31-Mar-2018)
Submitted to Daffodil International University on 2018-03-31

< 1% match (student papers from 10-Oct-2013)
Submitted to Higher Education Commission Pakistan on 2013-10-10

< 1% match (student papers from 18-Oct-2018)
Submitted to University of Bristol on 2018-10-18

< 1% match (student papers from 16-May-2018)
Submitted to Federation University on 2018-05-16

< 1% match (student papers from 01-Apr-2019)
Submitted to Daffodil International University on 2019-04-01

< 1% match (student papers from 28-Mar-2018)
Submitted to UT, Dallas on 2018-03-28

https://www.turnitin.com/newreport_printview.asp?eq=0&eb=0&esm=0&oid=1223182229&sid=0&n=0&m=2&svr=49&r=57.164749459434574&lang=e… 1/12