Natural Language to Query Interface

CS492 Project

CSU16118 MDL16CS037 Asish Pulasery
CSU16130 MDL16CS059 Joel V Zachariah
CSU16147 MDL16CS092 Renjith R Kartha
CSU16166 LMDL16CS129 Sreekumar T H
B. Tech Computer Science & Engineering

Department of Computer Engineering
Model Engineering College
Thrikkakara, Kochi 682021
Phone: +91.484.2575370
http://www.mec.ac.in
hodcs@mec.ac.in

April 2020

Model Engineering College Thrikkakara
Department of Computer Engineering

C E R T I F I C A T E

This is to certify that, this report titled ***Natural Language to Query Interface*** is a bonafide record of the work done by

CSU16118 MDL16CS037 Asish Pulasery
CSU16130 MDL16CS059 Joel V Zachariah
CSU16147 MDL16CS092 Renjith R Kartha
CSU16166 LMDL16CS129 Sreekumar T H

Eighth Semester B. Tech. Computer Science & Engineering
students, for the course work in **CS492 Project**, which is the second part of the two semester project work, under our guidance and supervision, in partial fulfillment of the requirements for the award of the degree, B. Tech. Computer Science & Engineering of **A. P. J. Abdul Kalam Technological University**

Guide

Manilal D L
Assistant Professor
Computer Engineering

Coordinator

Manilal D L
Associate Professor
Computer Engineering

Head of the Department

Manilal D L
Associate Professor
Computer Engineering

June 14, 2020

Acknowledgements

This project would not have been possible without the kind support and help of many individuals. We would like to extend my sincere thanks to all of them.

First of all, We would like to thank our esteemed Principal, Prof. (Dr.) Vinu Thomas, for his guidance and support in maintaining a calm and refreshing environment to work in and also for providing the facilities that this work demanded.

We am highly indebted to our Project Coordinator and Head of the Department, Mr. Manilal D L, Associate Professor for their guidance, support and constant supervision throughout the duration of the work as well as for providing all the necessary information and facilities that this work demanded.

We would like to thank our Project Guide, Mr. Manilal D L for his support and valuable insights and also for helping me out in correcting any mistakes that were made during the course of the work.

We offer our sincere gratitude to all our friends and peers for their support and encouragement that helped me get through the tough phases during the course of this work.

Last but not the least, we thank the Almighty God for guiding me through and enabling us to complete the work within the specified time.

Asish Pulasery

Joel V Zachariah

Renjith R Kartha

Sreekumar T H

Abstract

Database systems have been employed from the 1960's to store data in a structured manner. To derive knowledge, specialized querying languages such as SQL have been developed. However, it is a quite taxing process for laymen users to gain familiarity with such systems. To over come such challenges, typically a system administrator is present as a translator between the manager seeking the response and the knowledge encoded in the system. As an attempt to overcome this communication overhead between the two entities, research in the field of natural language user interface occurred in parallel from the 1970's. From rule based parser systems, to techniques involving natural language processing (NLP), several alternatives have been proposed over the decades. Many existing natural language interfaces to databases (NLIDB) propose solutions that may not generalize well to multiple domains and may require excessive feature engineering, manual customization, or large amounts of annotated training data. We present a technique for constructing subgraph queries to represent activities, events, persons, behaviors, and relations - to search against a graph database containing information from variety of data sources. Our model interprets complex natural language queries by using a pipeline of named entity recognition and bi- nary relation extraction models to identify key entities and relations corresponding to graph components such as nodes, attributes, and edges. This information is combined in order to create structured graph queries, which may then be applied to graph databases. There are many applications for this interface - from educational establishments to banking information.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the onset of the new decade, research in the field of data exploration has expanded multi-folds, with new approaches being suggested to better abstract the objects and detect patterns of importance. From conventional query language based database systems to present day natural language dependent chat bots, the control over data has shifted from specialized developers to common man, thus empowering the masses.

Data is the oil of the 21st century, and along with this potential comes the challenging problem of effectively querying it. Research in the field of data mining has proposed several models of storing and accessing data at scale to derive conclusions about the system. Existing systems have used pattern matching, syntax-based, and semantic grammar methods to produce intermediate query representations.

Natural language interface to database systems produce database queries by translating natural language sentences into a structured format which in our case, is a subgraph query. They play an increasingly important role as people, not only those with specific domain expertise or knowledge of structured query languages, seek to obtain information from databases. These databases contain massive amounts of data and are expanding rapidly, especially since text and voice interfaces have exploded in popularity due to the accessibility and prevalence of web and mobile technologies.

A tool that offers quick access to information in a graph containing data possibly from multiple sources would benefit analysts from various backgrounds, allowing them to focus their resources on other tasks. Many current systems require that the user specify relationships between edges and nodes in order to query the graph. A natural language based system must therefore correctly interpret the sentence, remove extraneous information, map tokens and phrases to nodes and edges, and resolve any ambiguities.

We approach this problem by implementing a pipeline of traditional natural language processing and machine learning components for prediction and classification tasks. Our solution requires a small training set, in contrast to neural models which are expensive in terms of time and cost to train.

1.1 Proposed Project

The proposed project draws upon the shortcoming of the existing rigid syntax based systems that lack flexibility. It aims to develop a system that performs conversion of natural language input to its equivalent graph database query to execute and return the relevant tuples.

1.1.1 Problem Statement

To create a database querying interface input the question in English and return the required result after querying the system - thus providing a seamless interaction medium for layman users without the need of knowledge of sophisticated querying languages such as gremlin, SQL etc.

1.1.2 Proposed Solution

Our propose model translates natural language queries into structured language queries for graph databases. At a high level, the system extracts tokens from the source sentence corresponding to the nodes, attributes, and edges (aptly differentiating between the three). Using this information, the system rebuilds the overall subgraph structure and runs it by the graph database system. Apriori training is carried out with expected input cases - thus providing the scope for minute variations in input to return the suitable results.

Chapter 2

System Study Report

2.1 Literature Survey

1. Intelligent Querying system using natural language

   This paper proposes an intelligent NLIDB(Natural language interface for DataBases) for querying databases.The input will be given in natural language which is converted to sql and used to query classic relational databases. The query generation can be divided into three phases: Semantic Building,Meaning Representation Generation,Query Generation.Semantic building phase takes DB schema as input and builds a semantic map using Wordnet(lexical Db).In MR generation phase the input query is converted to MR of query it could be in function form eg address(student).The final phase is query generation where the SQL query is generated here inputs are outputs from semantic map and MR.The MR and words in semantic map are compared with synonyms in Wordnet to generate the best query.

2. An Intelligent System for Querying and Mining Databases using NLP

   DBIQS(Database Intelligent Querying System) is a natural language interface for databases that generates sql queries from english natural language and uses the result to generate graphs and charts to represent data in Db quantitatively.After generating the query and the system is capable of creating tailored views for user.Allows for pooling of information from multiple databases ,thus the DBIQS is capable of data mining from DBs.

3. Research on the natural language querying for remote sensing databases

   This paper proposes an NLIDB specifically for Remote Sensing databases. The language is chineese.It discusses different methods used for natural language querying.(pattern matching,dependency extraction,knowledge extension).The method used here can be described as ,the model parses the natural language queries according to fixed rules and algorithm, extracts the keywords that occur in the query and matches them with entities in the databases.When there is no exact match obtained for the query from the databases, the model could rewrite the query into another query to get the additional meaningful results with a RS-based knowledge base.he model has three modules: keywords extraction, keywords extension and SQL generation.Keyword extraction extracts keyword from natural language and labels them with attribute annotations.Keyword extension defines relationship between keyword and entities in DB.from this the semantics can be used to match keywords with attributes during SQL generation phase.This is done by predefined rules.

4. Impact of intellisense on the accuracy of Natural Language Interface to Database

   NLIDBs allow easy retrieval of data from databases using natural language but most of them have not been able to achieve 100 percent accuracy.This paper proposes a model with intellisense and error handling capabilities to achieve 100 percent accuracy. A GUI is used to describe the DB and schema to user so that he/she may know what kind of queries to input. An intellisense mechanism suggests queries based on history and schema the required query when the user starts typing the query.Finally after generation of query the query passes through an error handling module which ensures only correct queries are passed.

5. An Independant - Domain Natural Language Interface for Relational Database : Case Arabic Language

   Natural language interface to databases(NLIDB) is one of the classic problems in the field of NLP The objective of NLIDB is to extract data from databases using natural language. The using of NLIDBs can provide powerful improvements to the use of information stored in a database.

   This paper, propose a generic Arabic natural language interface databases. This interface allows user to access data in a database by answering questions in arabic natural language query(ANLQ). The system used NLP techniques to translate ANLQ into SQL query. The interface is based on machine learning and natural language processing.

6. A Flexible and Efficient Natural Language Query Interface to Database

   To retrieve information from database requires knowledge of database language such as SQL. Access to database systems by naive users require a logically database independent interface NDLIB(Natural language interface to database) suffices this need. NDLIB allows access to database through natural language queries making them logically independent from data. Research indicates existing NDLIB systems lack flexibililty of forming queries in user's own format. The proposed system consists of three major components,

   (a) Analyser
   (b) Mapper
   (c) Translator

   The function of analyser to interpret the queries entered by the next user. Mapper is used to correspond natural language to SQL query. Finally, Translator performs actual translation of a query.

7. Web-Based Bilingual Natural Language Interface to Database

   The paper proposes an NLIDB system namely WB-NLIDB (Web based Bilingual Natural Language Interface to Database) that uses two natural languages Hindi and Punjabi. User can create query in both of the languages and receive the response in same natural language.

   An NUDB system can be divided into two major modules: Linguistic Module Its main purpose is to convert the natural language input of user into a SQL query and then produce output in the same natural language based on the database search results. For the production of correct data, that query is then executed by the database management system in order. This results in data which acts as an output of the natural language query.

   Database Module Query input in natural language is converted into formal query language.The linguistic model consists of three phases :

   (a) identifying the language
   (b) Parser which is used to identify the grammatical structure
   (c) Karak Solver (karak is a form of noun and pronoun that forms a direct connection with the word)

8. ATHENA: An Ontology-Driven System for Natural Language Querying over Relational Database
   An ontology-driven system for natural language querying of complex relational databases. Natural language interfaces to databases enable users easy access to data, without the need to learn a complex query language, such as SQL. ATHENA uses domain specific ontologies, which describe the semantic entities, and their relationships in a domain. We propose a unique two-stage approach, where the input natural language query (NLQ) is first translated into an intermediate query language over the ontology, called OQL, and subsequently translated into SQL.

   ATHENA uses domain ontologies, which describe the semantic entities and the relationships between them.The ontology domain consists of real-world entities called individuals which are grouped into concepts based on similarity of characteristics.

9. Team : A Transportable Natural Language Interface System

   This paper focuses on the problem of constructing transportable natural-language interfaces, i.e., systems that can be adapted to provide access to databases for which they were not specifically hand tailored. It describes an initial version of a transportable system, called TEAM (for transportable Engllsh Access Data manager). The hypothesis underlying the research described in this paper is that the information required for the adaptation can be obtained through an interactive dialogue with database management personnel who are not familiar with natural-language processing techniques. The TEAM system has three major components:

   (a) An acquisition component
   (b) The DIALOGIC language system
   (c) A data-access component.

10. Optimizing Natural Language Interface For Relational Databases

    Input in the form of natural language is parsed using a semantic grammar. Abstract syntax tree thus generated is translated into SQL query by the use of NLP. RNN (Recurrent Neural Network) is used for the conversion by training it on large datasets from WikiSQL to ensure sufficient accuracy. DBMS is used to find the result set - database is organized in tier architecture to reduce query execution time.

11. A Natural Language Interface for Querying Graph Databases

    The translation of natural language queries into structured language queries for graph database ensures completion. Named Entity recognition (NER) modules classify entities of interest into two - for graphs, and for attributes. The Binary Relation Extraction (BRE) module is used to identify relationships between entities. We train NER to recognize portions of sentence, and BRE finds the inter-relationships intended within the system.

12. A Model of Generic Natural Language Interface for Querying Database

    A generic natural language query interface for relational database based on machine learning approach is used. Natural Language is parsed syntactically; then converted into intermediate XML logic query, and then finally, IXQL is converted to Database query language (eg: SQL) and evaluated against the database

2.2 Proposed System

Our proposed system intrinsically consists of two phases:

- In the first phase, we follow an approach of determining the key entities of the sentence (like 'course', 'faculty', etc) and represent the information in the form of a subgraph structure containing two nodes and a connecting edge using trained NLP models.
- In the second phase, we convert the intermediate subgraph structure into a graph database query which is then executed against our graph database to retrieve the relevant information.

Chapter 3

Software Requirement Specification

3.1 Introduction

A Software Requirements Specification (SRS) is a description of the requirements of the software system which is to be developed. It is a rigorous assessment of all kinds of requirements before the more specific system design stages. The Software Requirement Specification document lists sufficient and necessary requirements for the development of the project. It describes all the functional and non-functional requirements faced by the project.

3.1.1 Purpose

The purpose of this document is to provide a detailed elaboration of the requirements in carrying out our project - which is to create a natural language query interface for accessing the institutional databases. We envisage an effective model for translating English sentence into intermediate representation, before the final graph query format for processing. The proposed system needs minimal training data for expected types of queries based on which the conversion model is created.

3.1.2 Document Conventions

Below are the expansions of all the acronyms followed in this document.

- SRS : Software Requirements Specification Document
- NLIDB : Natural Language Interface for Database
- NLTK : Natural Language Toolkit
- NER : Named Entity Recognition
- BRE : Binary Relation Extraction
- MITIE: MIT Information Extraction

3.1.3 Intended Audience and Reading Suggestions

This document is intended for readers ranging across the profiles of developers, project managers, and documentation writer. This document might also prove to be of value to aspiring researchers in the field of natural language interface development for database system.

3.1.4 Project Scope

We propose to develop a natural language interface to access databases. This proves to be beneficial for laymen who are not well versed in database querying languages such as SQL, Gremlin etc. Therefore, the end user is able to make queries in natural language (such as English) and retrieve the results efficiently with notable accuracy. We foresee further developments on the project to build a fully fledged application which provides further flexibility in the variations of the input sentence from the user for the same intended query.

3.1.5 Overview of Developer's Responsibilities

It will be the responsibility of the developer to create a tool to parse the natural language inputs from the layman user and generate an equivalent parse tree, which is then converted to a graph database query statement. This query is then evaluated against the existing database and the results are generated. Apart from this, the developer will ensure that the results generated are the intended ones by prior testing with sample database and comparing the results during the development phase. Corrections, if any, will be made by consulting the user for feedback.

Some general points to take note:

- The developer must be able to review the existing systems and draw parallels with the proposed system.
- Ideas for system improvements, including cost proposals, must be discusses and presented.
- The developer must work closely with analysts, designers, and other staff.
- Detailed specifications regarding the project must be produced and apt program codes should be written.
- The developer should test the product in controlled, real situations before going live.
- It's the developer's responsibility to maintain the system once it is up and running.

3.2 Overall Description

This section provides an overview of the system including its user classes and their characteristics, the various constraints, assumptions and dependencies in the system, and abbreviations and terms used throughout this document.

3.2.1 Product Perspective

The product aims to provide a seamless interface to query database systems using natural language as the medium. Our system makes use of a training model that determines the critical entities of the sentence and represent it in an intermediate form before converting to its equivalent gremlin query to retrieve results from the graph database. The success of the current approach relies on large training dataset with sufficient variation of possible input queries.
For identifying the critical entities (key words), we make use of MITIE library and it's tools such as NER and BRE processing.

3.2.2 Product Functions

The project aims to develop a software that can quite accurately convert English sentences to its equivalent graph queries.
To outline the expected functionalities:

1. Retrieve data from a graph database using a Natural Language query..
2. An error handling module checks if the query is correct; if not, it is rejected.

3.2.3 User Classes and Characteristics

The primary user is an individual familiar with the use case upon which the system is developed - generally a manager seeking insights to make decisions in the environment. The general user can use the system to to carry out basic queries for examination.

- Users
  - The general user intends to query the database in the natural language of convenience.
  - The users are familiar with the type of knowledge present in the database, thus the range of possible queries to pose.

3.2.4 Operating Environment

The system will be locally deployed as a web application to run on the browser - thus easy access independent of the kind of device utilized.

3.2.5 Design and Implementation Constraints

- The training process can be time consuming for developing the entity extraction models.
- The system can risk falling short in accuracy with sufficiently complex queries.

- System is configured for graph databases.
- The system is restricted to the English language.
- The user is constraint to certain queries by error handling module.

3.2.6 User Documentation

The following abbreviations have been used in the document:

- NLP: Natural Language Processing
- NER: Named Entity Recognition
- BRE: Binary Relation Extraction

The following terms have been extensively used throughout the document:

- Named Entity Recognition: It is the process of identifying words that fall within a category (eg: 'apple' is a 'fruit').
- Binary Relation Extraction: It is the process of determining relations within the system that pertain with respect to the graph database.

3.2.7 General Constraints

The general constraints include:

- The training process can take several hours
- The input sentence should be of the kind that the trained model is familiar with
- Limitations due to the degree of variety in the dataset used for training.

3.2.8 Assumptions and Dependencies

The assumptions and dependencies of our product include:

- The input sentence is of the kind that the trained model is expecting
- For database, a graph based model will be used.
- The model is assumed to have been accurately trained.

3.3 External Interface Requirements

External interface requirements specify hardware, software, or database elements with which a system or component must interface. This section provides information to ensure that the system will communicate properly with external components. The following are the external interface requirements concerned with the project.

3.3.1 User Interfaces

The query to be tested is to entered into a text column and the submit button is to be clicked. The result will be displayed on page after retrieving from the database. The system will also display information about the database, making it easier for the user to frame queries.

3.3.2 Hardware Interfaces

No external hardware systems have been used in the project.

3.3.3 Software Interfaces

- Operating System: Linux, Windows
- Programming Language: Python, HTML, Javascript
- Library: Gremlin-Python Server
- Input Data: A string (English query)

3.3.4 Communication Interfaces

The communication interface is maintained through function callbacks from the front end of the product. When the user interacts with the system, the function associated with that interaction will be called. A callback is a function that we associate with a specific GUI component or with the GUI figure. It controls component behavior by performing an action in response to an event for that component.

3.4 Hardware and Software Requirements

The hardware and software requirements form the primary base for software development. These requirements may vary depending on the machine and the operating system. The following section describes the hardware and software specification required for the project.

3.4.1 Hardware Requirements

- Laptop with Linux based OS with networking capabilities.
- RAM of minimum 8 GB

3.4.2 Software Requirements

- Operating System: Windows or any Linux distribution.
- Python: Python is an easy to learn and powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python, with its elegant syntax and dynamic typing, together with its interpreted nature, makes it an ideal language for rapid application development on most platforms. Python also has library support for data manipulation and machine learning. Pythons approach to the object-oriented paradigm makes it easier to divide modules and collaborate in a project. Pythons inbuilt functions for reading and writing files makes it to work on multiple file formats and images. Functions used: open (to read file)
- Gremlin: Gremlin is a graph traversal language and virtual machine developed by Apache TinkerPop of the Apache Software Foundation. Gremlin works for both OLTP-based graph databases as well as OLAP-based graph processors. Gremlin's automata and functional language foundation enable Gremlin to naturally support imperative and declarative querying, host language agnosticism, user-defined domain specific languages, an extensible compiler/optimizer, single- and multi-machine execution models, hybrid depth- and breadth-first evaluation, as well as Turing Completeness.
- MITIE: MITIE is an NLP information extraction tool, developed by Massachusetts Institute of Technology. The primary tools available include named entity recognition and binary relation detection. MITIE makes use of several state-of-the-art techniques including the use of distributional word embeddings and Structural Support Vector Machines. MITIE offers several pre-trained models providing varying levels of support for both English, Spanish, and German trained using a variety of linguistic resources. The core MITIE software is written in C++, but bindings for several other software languages including Python, R, Java, C, and MATLAB allow a user to quickly integrate MITIE into his/her own applications.
- NLTK: The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing (NLP) for English written in the Python programming language. NLTK is intended to support research and teaching in NLP or closely related areas, including empirical linguistics, cognitive science, artificial intelligence, information retrieval, and machine learning.[7] NLTK has been used successfully as a teaching tool, as an individual study tool, and as a platform for prototyping and building research systems.

- React: React (also known as React.js or ReactJS) is an open-source JavaScript library for building user interfaces. React can be used as a base in the development of single-page or mobile applications. However, React is only concerned with rendering data to the DOM, and so creating React applications usually requires the use of additional libraries for state management and routing. Redux and React Router are respective examples of such libraries.

- Node.js : Node.js is an open-source, cross-platform, JavaScript runtime environment that executes JavaScript code outside a web browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web-application development around a single programming language, rather than different languages for server- and client-side scripts.

3.5 Functional Requirements

The functional requirements of the project are:

- Training
- Testing
- English Query input
- Results from Database

3.5.1 Training

The system is trained based on a set of natural language queries possibly asked by users and outputs classified into entities and relations.

3.5.2 Testing

The system is tested based on test set of natural language queries possibly asked by users and outputs classified into entities and relations.

3.5.3 English Query Input

User can type the English query in the search box and click enter to submit the same.

3.5.4 Results from Database

The result is retrieved from the database and is presented on the screen.

3.6 Non-functional Requirements

Non-functional requirements define system attributes such as performance, quality, availability, maintainability, usability, reliability. They serve as constraints or restrictions on the design of the system across different backlogs. Also known as system qualities, non-functional requirements are justified as critical as functional requirements. They ensure the usability and effectiveness of the entire system.

3.6.1 Performance Requirements

- The performance of the algorithms employed in the project should be fast and accurate.
- The training process will take some time depending on the size of the training set used.
- Any internal errors should be handled by the system and separate log should be maintained.
- The UI screen should respond quickly.
- The system will timeout after 3s if the information is not retrieved in specified time.

3.6.2 Safety Requirements

The queries that can be used to modify the DB (Data Definition Type) is limited to prevent malicious use of the system.

3.6.3 Software Quality Attributes

The system must be capable of retrieving information with accuracy and precision. It should also provide maximum accuracy and the results must be computed in minimum time. This provides the user with a reliable experience.

3.6.4 Availability

- As part of the future scope of the project, the system will be hosted as a web service making the service available all time.
- The service must always be running properly.
- The computations will be carried out on the cloud, making it faster as optimal CPU usage.

3.6.5 Maintainability

The system will require minimum maintenance as it is offline

3.6.6 Usability

- The system should provide an easy and friendly user interface.
- The users can test/use the system easily.

3.6.7 Reliability

- Since the system is offline based, it is always available to the user.

Chapter 4

System Design

4.1 System Architecture

The system accepts a natural language query (eg: A request made in English language) via graphical user interface as input and produces the corresponding attribute values from the database. The system first performs some preprocessing steps on the input natural language sentence. The entities in the sentence are recognized and the relations are extracted - together forming an intermediate form containing node and edge information. Following this, the system automatically convert the intermediate representation into a standard graph database query to execute and retrieve results from the graph database.

Input	Teachers who take DCS
NER-G	Score: range(0, 1) 0.505: N: Teachers Score:range(3, 4) 0.296: N: DCS
NER-A	Score: range(3, 4) 0.875: Course_name: DCS
BRE prof-course	Teachers,DCS
Intermediate-form	Nodes: [{'id': 'N0', 'type': 'Professor'} ,{ 'id': 'N1', 'type': 'Course', 'attributes': { 'name': 'Course_name', 'value': 'DCS', 'operator': 'equal' }}] Edges:[{'from': 'Professor', 'to': 'Course', 'type': 'teaching', 'attributes': []}]
query	g.V().hasLabel("Course").has("name", "CS_404").out().hasLabel("Professor").values("name")
output	'lomin'

Table 4.1: Input and output

The system architecture is as shown, with detailed elaboration of each module stated in the following sub-sections:

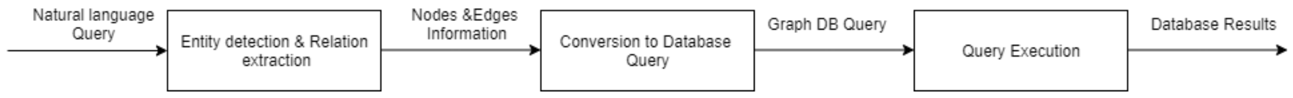

Figure 4.1: System Architecture

4.1.1 Entity detection and Relation Extraction

The natural language query sentence is input from the user - for example: ”List the students of class C7A with marks less than 40”. This English sentence is processed before proceeding further to overcome the challenges imposed by stop words or punctuation symbols, if any. It is then processed to determine the critical entities, and the relations between these entity terms. Indirectly we are attempting to infer the connections that it correspond to in the graph database.

4.1.2 Conversion to Graph Database Query

The intermediate data consisting of edge and node information is processed to generate the graph database query. We will be using the Gremlin Traversal Language to access the Graph Database. Gremlin is a functional language whereby traversal operators are chained together to form path-like expressions.

4.1.3 Query Execution

The graph Database Query does graph traversal step-by-step. An example of a gremlin query is:*g.V().hasLabel(”student”).has(”class”,”C7A”).has(”marks”,lt(40)).values(”name”)*. This style of building up a traversal/query is useful while constructing larger, complex query chains. The Graph Database Query created thus far is executed over the graph database of the institution, and the corresponding results are displayed on the monitor for the end user.

4.2 Data Description

We are using an Institutional Database - Educational Establishments in particular. We implement it in the form of a graph database - with edges and nodes to represent the inter-relationships within the system. In the following sub-sections further clarity shall be provided.

4.2.1 Database Design

We follow the Entity Relationship diagram of a University Schema, as depicted in Figure 4.2

4.3 Use Case Diagram

A use-case diagram at its simplest is a representation of a users interactions with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases in company with other types of diagrams as well. The primary user is familiar with the structure of the database and utilizes this interface to determine the results to the queries stated in english. The system processes the query and generates the results at the end of multiple stages.

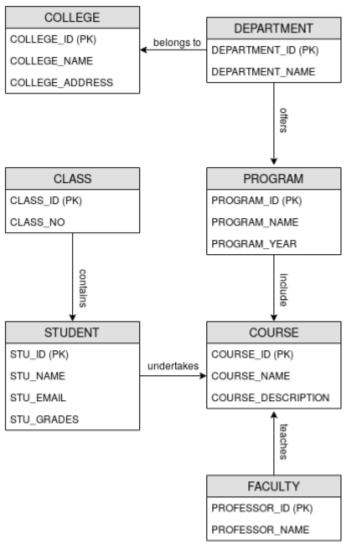

Figure 4.2: Database Design

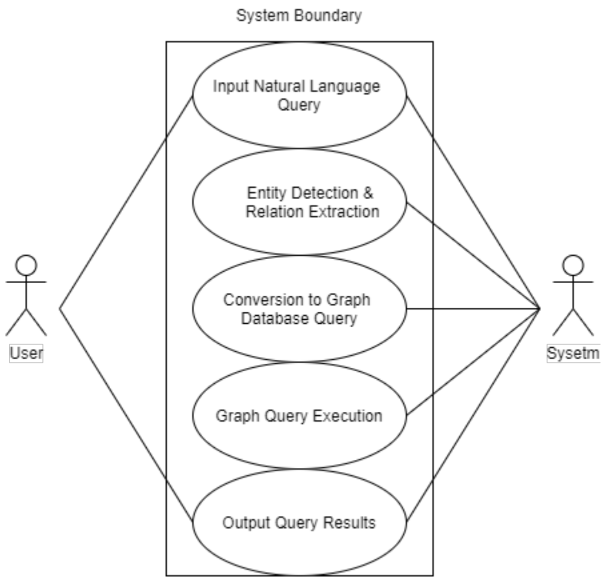
System Boundary
Input Natural Language Query
Entity Detection & Relation Extraction
Conversion to Graph Database Query
Graph Query Execution
Output Query Results
User
Sysetm

Figure 4.3: Use Case Diagram

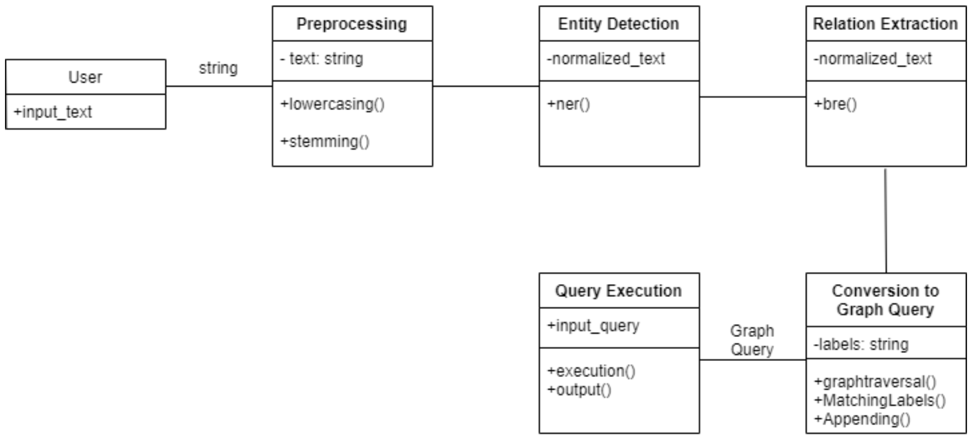

Figure 4.4: Class Diagram

4.4 Class Diagram

A class diagram models the static structures of a system. It shows relationships between the systems classes, their attributes, operations, and the relationships among objects. Essentially, there are six classes - User, Preprocessing, Entity Detection, Relation Extraction, Conversion to Graph Query, and Query Execution. Each class's associated data and functions are shown in the Figure 4.3.

4.5 Activity Diagram

Activity Diagrams are graphical representation of workflows of step-wise activities and actions with support for choice, iteration and concurrency. These diagrams show the overall flow of control through the system. Our activities commence when the user enters the English query intended to be executed. The sentence is processed to determine the critical entities and relations between them. The intermediate representation of nodes and edges is used to create a graph database query which is then executed to retrieve the results. Our activity diagram is as illustrated in Figure 4.5

4.6 Dataset Design

The dataset used in the implementation of the project is custom made set of 70 queries.

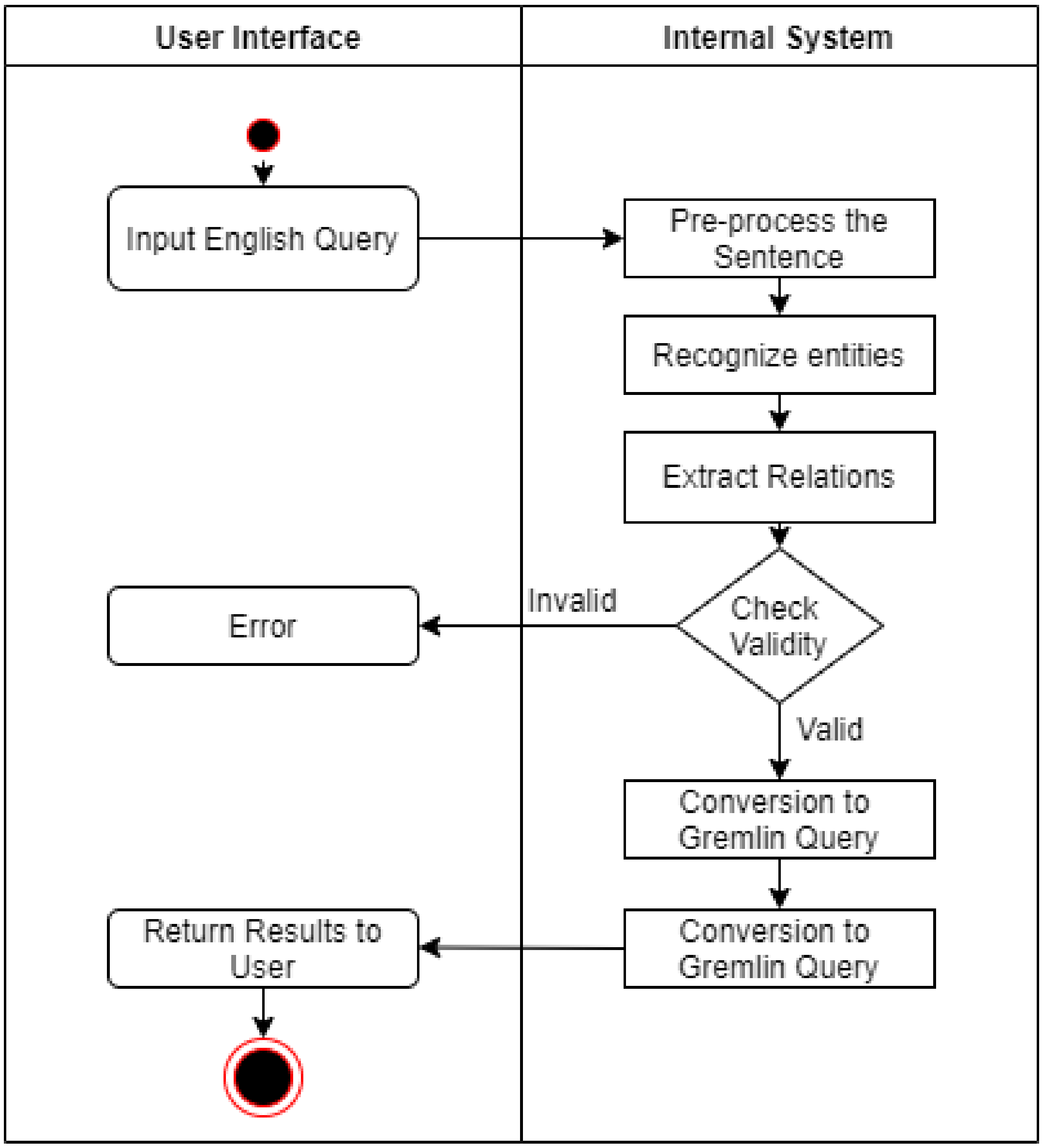

Figure 4.5: Activity Diagram

Chapter 5

Data Flow Diagram

A data-flow diagram (DFD) is a way of representing a flow of data of a process or a system (usually an information system). The DFD also provides information about the outputs and inputs of each entity and the process itself. A DFD is often used as a preliminary step to create an overview of the system without going into great detail, which can later be elaborated. This section includes the data flow diagram(DFD Level 0, Level 1, Level 2).

5.1 Level 0 DFD

A level 0 data flow diagram (DFD), also known as a context diagram, shows a data system as a whole and emphasizes the way it interacts with external entities. Here we have as input an English sentence which is the query we wish to execute with respect to the database. This is given to the system, ie., The Natural Language Query Interface, and the output produced is the result of the query.

5.2 Level 1 DFD

A level 1 data flow diagram (DFD) is more detailed than a level 0 DFD but not as detailed as a level 2 DFD. It breaks down the main processes and sub-processes that can then be analyzed and improved on a more intimate level. The main functions of the interface include graph database query creation and graph database query execution.

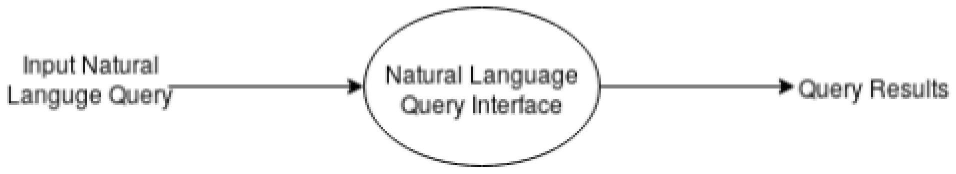

Figure 5.1: Data Flow Level 0

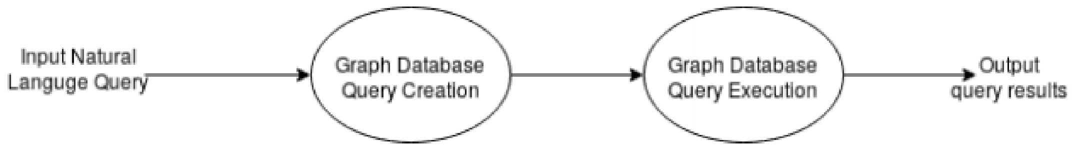
Input Natural Languge Query
Graph Database Query Creation
Graph Database Query Execution
Output query results

Figure 5.2: Data Flow Level 1

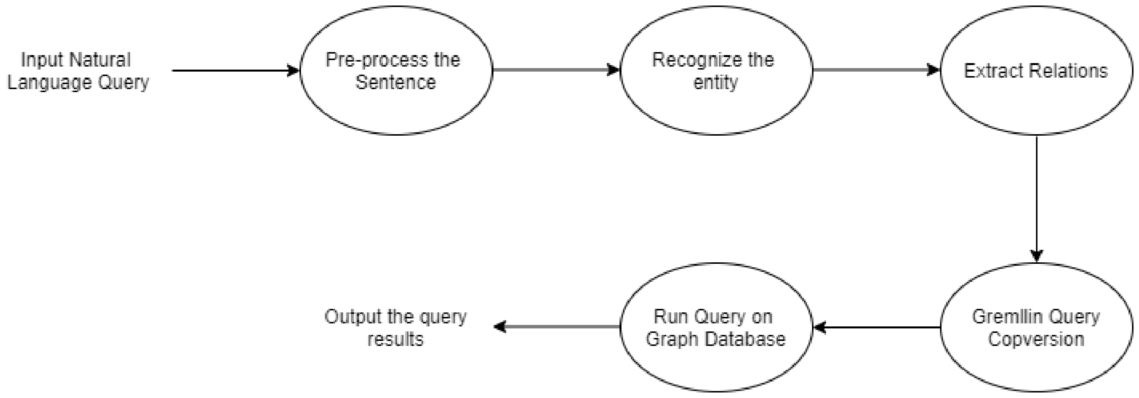

Figure 5.3: Data Flow Level 2

5.3 Level 2 DFD

A level 2 data flow diagram (DFD) offers a more detailed look at the processes that make up an information system than a level 1 DFD does. It can be used to plan or record the specific. Graph Database Query Creation includes sentence preprocessing, entity recognition and relation extraction, and Gremlin Query Conversion. Graph Database Query Execution involves running the generated graph database query to retrieve the results.

Chapter 6

Implementation

6.1 Algorithms

The proposed method has three major steps:

1. Entity Recognition and Relation Extraction
2. Graph Query Generation
3. Graph Query Execution

6.1.1 Algorithm for Entity Recognition and Relation Extraction

Algorithm 1: Algorithm for Entity Recognition and Relation Extraction

```
   Input: English Sentence
   Output: graph structure
 1 Train named entity recognizer for graph components (NER-G)
 2 Train named entity recognizer for attribute types (NER-A)
 3 For each edge type:
       Train binary relation extractor BRE-edge-type given NER-G
 4 Use NER-G to predict 'N', 'E', 'A' entities given input sentence
 5 For each entity tagged 'N' by NER-G:
       Record in node_dict (key = entity position, value = node id)
 6 Use NER-A to predict attribute tokens and their attribute type given input sentence
 7 For each entity tagged by NER-A:
       Record in node_dict (key = entity position, value = node id, attribute type)
 8 For each entity tagged 'N' by NER-G:
       Get node type through embedding comparison
       Get node id through node_dict lookup
       Create node stricture, add it to overall structure
 9 For each entity tagged by NER-A:
       Get attribute type through node dictionary lookup
       Create attribute structure
       Get node type from attribute type
       if node structure of these same type exists:
           Add attribute structure to that node
       else
           Get node id through node_dict lookup
           Create a new node structure
           Add attribute structure to the new node
           Add node structure to the overall structure
10 Get relation_pairs:
      For source_node in node_dict:
          For target_node in node_dict:
              If source_node not equal target_node:
                  relation_pair = (source_node,target_node)
11 For each relation_pair in relation_pairs:
      For each edge type:
          Use BRE-edge-type to predict edge_score
          If edge_score greater than 0: relation exists; create edge structure
```

6.1.2 Algorithm for Graph Query Generation

Algorithm 2: Algorithm for Abstract Syntax Tree to Graph Database Query conversion

```
  Input: Graph structure
  Output: Graph Database Query
1 Start
2 Iterate over the Graph structure returned.
3 For each new label detected, see if a corresponding gremlin query fragment exists.
4 If present, append it to the base query that is partially formed.
5 By the end of the processing of the graph structure, the complete gremlin query will be
    created.
6 Pass forward the Gremlin Query.
7 Stop
```

6.1.3 Algorithm for Graph Query Execution

Algorithm 3: Algorithm for Query Execution

```
  Input: Gremlin Query
  Output: Results of the query
1 Start
2 Initiate a Gremlin Server linked to a Graph Database system of the institution.
3 Execute the Gremlin query on the Gremlin Server.
4 Store the results in a suitable data structure such as a list.
5 Print the list to output the results to the user
6 Stop
```

6.2 Development Tools

A programming tool or software development tool is a computer program that software developers use to create, debug, maintain, or otherwise support other programs and applications. The term usually refers to relatively simple programs, that can be combined together to accomplish a task, much as one might use multiple hand tools to fix a physical object. The most basic tools are a source code editor and a compiler or interpreter. Other tools are used more or less depending on the language, development methodology, and individual engineer, and are often used for discrete task, like a debugger.

6.2.1 Visual Studio Code

Visual Studio Code is a source-code editor developed by Microsoft for Windows, Linux and macOS. Visual Studio Code is a source code editor that can be used with a variety of programming languages. Instead of a project system it allows users to open one or more directories, which can then be saved in workspaces for future reuse. Some of the features offered include:

- Support for debugging, embedded git control, syntax highlighting
- Intelligent code completion, snippets, and code refactoring
- Customizable themes, keyboard shortcuts, and preferences
- Plug-in based extensions available through central repository; It includes additions to the editor and language support.
- It has out-of-the-box support for almost every major programming language.

6.2.2 GitHub

GitHub is a web-based hosting service for version control using Git. It offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management etc for every project. Main features are:

- Seamless code review
- Project and team management
- Provides quality documentation
- Supports large amounts of code hosting

Chapter 7

Testing

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include the process of executing a program or application with the intent of finding software bugs, and to verify that the software product is fit for use.

Software testing involves the executing of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:

- meets the requirements that guide its design and development
- responds correctly to all kinds of inputs
- performs its functions within an acceptable time is sufficiently usable
- can be installed and run in its intended environments, and
- achieves the general results its stakeholders desire.

7.1 Testing Methodologies

Software testing methodologies are for making sure that the software products/systems developed have been successfully tested to meet their specified requirements and can successfully operate in all the anticipated environments with required usability and security. Software testing methods are traditionally divided into white and black-box testing. These two approaches are used to describe the point of view that a test engineer takes when designing test cases.

- **White-Box Testing**: By seeing the source code tests internal structures or workings of a program, as opposed to the functionality exposed to the end-user. In white-box testing an internal perspective of the system, as well as programming skills are used to design test cases, The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit. While white-box testing can be applied at the unit, integration and system levels of the software testing process , it is usually done at the unit level.

- **Black-Box Testing**: The software is treated as a black-box, examining functionality without any knowledge of internal implementation, without seeing the source code. The testers are only aware of what the software is supposed to do, not how it does it.

  The testing methods applied were:

  - **Unit Testing**: Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinized for proper operation.
  - **Integration Testing**: Integration testing is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing.
  - **System Testing**: System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system compliance with its specified requirements. System testing falls within the scope of black-box testing, and as such, should require no knowledge of the inner design of the code or logic.

7.2 Unit Testing

In unit testing NER-G , NER-A, BRE , node and edge processing were tested separately.We evaluate the performance of each module in the pipeline by computing precision, recall.

$$precision = TP/(TP + FP)$$

$$recall = TP/(TP + FN)$$

Precision is the fraction of retrieved elements, or instances predicted true, that are rel- evant. Recall is the fraction of relevant elements, or instances that are actually true, that were retrieved.

7.2.1 NER-G and NER-A

Table 7.1 presents the results for the two named entity recognition.For each model, we run k-fold cross validation by partitioning the training set into k = 5 splits. The values reported are the average of the scores when holding one split out for evaluation and using the rest of the data for training the model

Model	**Precision**	**Recall**
NER-G	0.850	0.868
NER-A	0.875	0.882

Table 7.1: Precision and recall of NER

7.2.2 Binary Relation Extraction

Tests similar to NERs are performed for BRE and the results are given in table 7.2.

Model	**Precision**	**Recall**
BRE-student-course	0.855	0.852
BRE-professor-department	0.896	0.819
BRE-professor-course	0.858	0.849
BRE-operator	0.773	0.718

Table 7.2: Precision and recall of BRE

Operator BRE has performed poorly in comparison due to ambiguities on which part of sentence the operator affects and if so are there multiple entities affected or just single ones.

7.2.3 Node and Edge processing

Since the output of same sentence may vary from expected output but is capable of giving desirable results , outputs of this phase were tested qualitatively with a set of 20 sentences of which 12 gave the correct values.

7.3 Integration Testing

Integration testing is the phase in software testing in which individual software modules are combined and tested as a group.The modules of NER-G ,NER-A, and BRE are integrated with nodes and edge processing.Further it is hosted on node server with a GUI and test were conducted with the system returning results to frontend GUI.

7.4 System Testing

After the integration testing, we do the system testing. In system testing the whole modules are connected in order.The user has to open the URL of our web app and simply enter the question on the text field or select question from recommendations. After clicking the submit button the result is received from database.

Chapter 8

Graphical User Interface

The graphical user interface (GUI) is a type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, instead of text-based user interfaces, typed command labels or text navigation. GUIs were introduced in reaction to the perceived steep learning curve of command-line interfaces which require commands to be typed on a computer keyboard. Designing the visual composition and temporal behavior of a GUI is an important part of software application programming in the area of human-computer interaction. Its goal is to enhance the efficiency and ease of use for the underlying logical design of a stored program, a design discipline named usability.

Methods of user-centered design are used to ensure that the visual language introduced in the design is well-tailored to the tasks. For several decades, GUIs were controlled exclusively by a mouse and a keyboard. While these types of input devices are sufficient for desktop computers, they do not work as well for mobile devices, such as smart-phones and tablets. Therefore, mobile operating systems are designed to use a touchscreen interface. Many mobile devices can now be controlled by spoken commands as well.

Since there are now multiple types of digital devices available, GUIs must be designed for the appropriate type of input. For example, a desktop operating system, such as OS X, includes a menu bar and windows with small icons that can be easily navigated using a mouse. A mobile OS, like iOS, includes larger icons and supports touch commands like swiping and pinching to zoom in or zoom out. Automotive interfaces are often designed to be controlled with knobs and buttons, and TV interfaces are built to work with a remote control. Regardless of the type of input, each of these interfaces are considered GUIs since they include graphical elements.

8.1 GUI Overview

A GUI uses a combination of technologies and devices to provide a platform that users can interact with, fr the tasks of gathering and producing information.
For our system, the GUI is implemented using React.js and can be access on a web browser. Key benefits of using React.js include

- Web based GUI feels authentic and provides a comfortable user experience
- It facilitates the overall process of writing components

- Component re-usability boosts productivity and facilitates further maintenance
- Ensures faster rendering
- Guarantees stable code, as child structures won't affect parents (downward data flow)
- Since it is solely browser dependent, it works cross platform.

8.2 Main GUI Components

GUI control components are the primary elements of a graphical user interface that enable interaction with the user. The GUI for our system does not require a database. The main GUI components present include:

- Query Entry Bar: A box to accept user's English query to run against the database
- Submit Button: A button, which when clicked, will convert the English query to a graph database query and return the result.
- Result section: A box to show the retrieved result from the database

The working of the GUI can be explained in a sequence of events as listed below:

- The user can click the Query Entry Bar to start typing the English Query intended to execute
- Once done typing (with the help of auto suggestions), the user clicks the submit button
- Once the conversion and execution is completed, the retrieved results are displayed in the Result section.

Chapter 9

Results

9.1 Qualitative Analysis

As we have observed in Table 4.1, there are multiple stages of the representation of the querying information. From manumally reviewing the structured language predictions, we uncover sentence patterns that the system has trouble dealing with. Incorrectly classified outputs of the named entity recognition and binary relation extraction modules contribute to some of the overall structured language prediction error. Many errors arise when combining these outputs in the later phases of the model pipeline.

9.1.1 Word embeddings

Using word embeddings alone to determine the corresponding node type of each entity is inaccurate in many cases. We hard code the correct node type rather than compare the embeddings. Instead of using word embeddings, the system could use a multi-class classifier to predict the node type of the entity.

9.1.2 Multiple attributes

The current system creates one node for each attribute and each node detected. Most of the time, these attributes are associated with the same node. Therefore, an effective node combination step in necessary in order to reduce prediction error. The system's default behavior is to combine all nodes of the same type, and this is not always correct. To rectify this, a binary relation extractor can be considered to determine whether or not two nodes relate to the same entity, and should therefore be combined.

9.2 Graphical User Interface Results

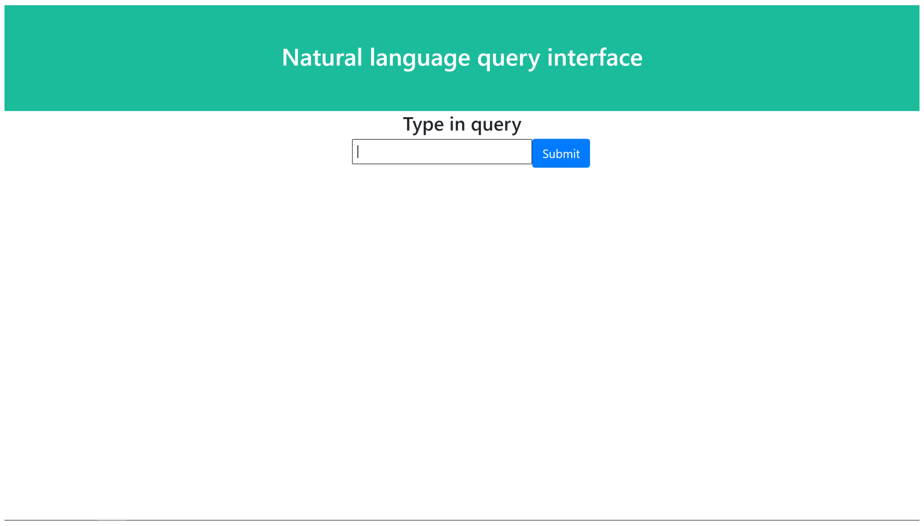

Figure 9.1: Home Screen

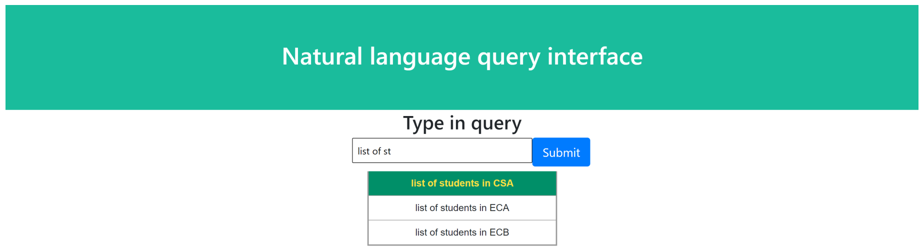

Figure 9.2: Auto-suggestion prompt

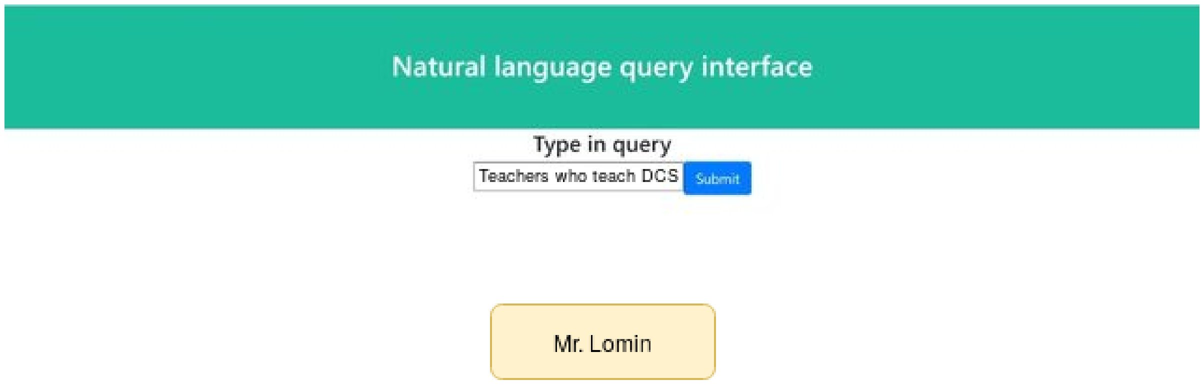

Figure 9.3: Result

Chapter 10

Conclusion

The project "**Natural Language to Query Interface**" provides a mean to access database systems using English query statements rather than specialized query languages. Presently, Natural Language Interface for Databases (NLIDB) is a sufficiently challenging task that necessitates a certain level of preprocessing to effectively cater to the requested query for the database. We have presented an approach involving NLP techniques such as Named Entity Recognition and Binary Relation Extraction to determine the graph query which best represents the intention of the user. Our project relies on prior training with expected types of user input, to provide a degree of flexibility within each query. By determining the node, the connecting edges and the associated attributes, we generate the graph database query. This query is executed against the graph database to retrieve the expected results. We have additionally included autosuggestion feature to guide the user into entering a query that best accommodates within the types the system is expecting, thus decreasing the risk of generating an incorrect query as a result of an ill formed English query format.

The following ares have been identified as possible applications:

- In Institutional Database Systems, the interface can be utilized for conveniently deriving conclusions about student performance, teaching quality and the scope for further enhancement.
- For the students of an introductory database course, the interface can abstract the syntactic complexity of a querying language (eg: SQL) by demonstrating the benefits of a database using simple English sentences to ask questions and fetch results, thus making it less intimidating to learn.

Chapter 11

Future Scope

In future work, enhancing our model to work with a greater variety of queries can be greatly promising. While the present system depends greatly prior training for each use case, it can be extended via generalization techniques to tackle any scenario by drawing the outline from a single time training.

With respect to the platform, it will be interesting to create a desktop application of the same to avoid dependency on an internet connection usage (offline mode). This would, however, necessitate the provision of usage modes (for security reasons) depending on the level of privilege permissible. Moreover, the technique can be modified to generate relational database queries (eg: SQL) instead of graph database query.

Finally, it would be compelling to further develop this system by using latest state-of-the-art sequence to sequence conversion models that can accept any language (not just English) as the option for input format. This opens the door to adoptions globally irrespective of primary language dependence to retrieve information from the database.

References

[1] Intelligent Querying system using natural language by Prashant Gupta, Aman Goswami, Sahil Koul, Kashinath Sartape

[2] DBIQS - An Intelligent System for Querying and Mining Databases using NLP by Rohit Agrawal, Amogh Chakkarwar, Prateek Choudhary, Usha A. Jogalekar, Deepa H. Kulkarni.

[3] Research on the natural language querying for remote sensing databases by Xuan Xuan, Liu Jianbo, Yang Jin.

[4] Impact of intellisense on the accuracy of Natural Language Interface to Database by Niket Choudhary Dept. of Computer Engineering, Pimpri-Chinchwad College of Engineering, Pune, India.

[5] An Independent - Domain Natural Language Interface for Relational Database : Case Arabic Language by Hanane Bais, Mustapha Machkour, Lahcen Koutti.

[6] Natural language question answering over RDF- A graph data driven approach by Lei Zou Peking University,Ruizhe Huang Peking University Haixun Wang Microsoft,Jeffrey Xu Yu.

[7] A natural language interface for querying graph databases by Christeena sun S.B., Computer Science and Engineering Massachusetts Institute of Technology.