****

**Natural Language Interface for Database**

**A Final Year Project**

In Partial Fulfillment of the Requirements for the degree of

**Bachelor of Science in Computer Science and Information Technology**

**Of**

**Tribhuvan University**

**Amrit Campus**

**By**

Mr. Anup Pokhrel [522/066]

Mr. Suman Adhikari [545/066]

**Under the Supervision**

**Of**

Mr. Bikash Balami

**Lecturer**

**Amrit Campus**

**Department of Computer Science and Information Technology, TU**

**Submitted to**

**Institute of Science and Technology**

**Amrit Campus**

**Department of Computer Science and Information Technology, TU**

**Thamel, Kathmandu**

**December 2013**

**Amrit Campus,**

**Tribhuvan University**

Date … … … … … …

**Supervisor’s Recommendation**

I hereby recommend that this project prepared under my supervision by Mr. Anup Pokhrel, Mr. Suman Adhikari entitled “**Natural Language Interface for Database** ”in partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Information Technology be processed for the evaluation.

… … … … … … … … … … …

**Mr. Bikash Balami**

Amrit Campus

Lainchaur, Thamel

Department of Computer Science and Technology

**Amit Campus**

**Tribhuvan University**

Date: … … … … … … …

**LETTER OF APPROVAL**

This is to certify that this project prepared by Mr. Anup Pokhrel, Mr. Suman Adhikari entitled “Natural **Language Interface for Database”** in partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Information Technology has been well studied. In our opinion it worth’s quality as a project for the required degree.

|  |  |
| --- | --- |
| … … … … … … … … … … … …  **Assoc. Prof. Mr. Deepak Pudasaini**  Head of department  Amrit Campus,Computer Science and IT | … … … … … … … … … … … …  **Lecturer Mr. Bikash Balami**  Project Supervisor  Amrit Campus,Computer Science and IT |
| … … … … … … … … … … … …  … … … … … … … … … … … …  Internal Supervisor | … … … … … … … … … … … …  **Lecturer Mr. Jagdish Bhatt**  External Supervisor |

ABSTRACT:

Automatically mapping natural language into programming language semantics has always been a major and interesting challenge. Furthermore, as now almost all IT applications are storing and retrieving information from database. Thus retrieving information form the database requires knowledge of technical languages such as Structured Query Language. Moreover most of the users who interact with databases has no knowledge or are not form any technical environment. This has led us to develop the Natural Language Interface for Database where a user from any background is able to query his/her information using natural language. Asking question to databases to in natural Language is very convenient and easy approach of data access from user points of view. Therefore we have developed a Natural Language Interface for Database which will take the query in natural language and automatically map the NL sentence to respective query and show results.

In this project, we approach such problem by carrying out a mapping between Natural Language (NL) and SQL syntactic structures. The mapping is automatically derived by applying unsupervised machine learning algorithms. In particular, we have used Stanford Dependency Parser to get dependencies that exits with in the sentence and divide the list into possible select and where statements. Then, we use our own data dictionary and MySQL’s Information Schema to train the possible select and where statements. Finally the algorithm decide the possible select clause and where clause and finally form from clause and final query is resulted.

In particular we exploit linguistic dependencies in the natural language question and the database metadata to build a set of plausible SELECT, WHERE and FROM clauses enriched with meaningful joins. Then, we combine all the clauses to get the set of all possible SQL queries, producing candidate queries to answer the question.

Keywords [Natural Language Interfaces, Database, Question Answering, Stanford Dependency Parser,]

ACKNOWLEDGEMENTS:

We would like to express the deepest appreciation and gratitude toward following people for their various contributions regarding the completion of this project. We could be unable to work on this project without their assistance and support.

* our supervisor, **Lecturer Mr. Bikash Balami**, both for his invaluable intellectual guidance and for his warmth and empathy. The warm embrace of his hands gave us a sense of encouragement that lead towards success of project. Lecturer, we owe you depths of gratitude;
* **Assoc. Prof. Deepak Pudasaini** of the CSIT department, Amrit Campus, who despite his very busy schedule both as a lecturer and HOD, generously employed his intellectual expertise to painstakingly undertake the continuously provide help support and reference. Prof. we thank you for your unparalleled kindness;
* **Mr. Hemanta G.C, Mr. Uttam Thapa Rana , Mr. Bhoj Raj Ghimire** all lecturers for having demonstrated genuine interest in our project and providing possible aids and suggestions
* our parents, brothers and sisters for their moral support, material encouragement and unflinching understanding;
* all the research participants, not only for the sacrifice of their time, but also for painstakingly supplying the data used in this study;
* all staff of the CSIT Department, Amrit Campus for their warmth and professional support;
* all our friends, among whom are Sajjan and Ravi, for their time, faith and encouragement. Many thanks to Pratap , Shirshir , Uddhav , Arun and Gokul for their assistance during the formatting of this thesis;
* all reviewers for having help in testing and have helped us to improve our project much more and their innovative ideas to design the interface

TABLE OF CONTENTS

[Introduction 1](#_Toc376663952)

[Background 3](#_Toc376663953)

[2.1 Natural Language Processing 3](#_Toc376663954)

[2.1.2 Stemming 3](#_Toc376663955)

[2.1.2 Parsing 4](#_Toc376663956)

[2.1.3 Stanford Dependency Parser 4](#_Toc376663957)

[2.2 Databases 4](#_Toc376663958)

[2.2.1 Relational Database 5](#_Toc376663959)

[2.2.2 Metadata and Information Schema 5](#_Toc376663960)

[2.3 Motivating Example 6](#_Toc376663961)

[Problem Statement 8](#_Toc376663962)

[Objective 9](#_Toc376663963)

[Scope and Limitation 10](#_Toc376663964)

[5.1. Scope 10](#_Toc376663965)

[5.2. Limitation 10](#_Toc376663966)

[State of the Arts 12](#_Toc376663967)

[6.1 Spoken Language Understanding 12](#_Toc376663968)

[6.2 Question Answering in NLI 12](#_Toc376663969)

[Requirement Analysis and Feasibility Study 15](#_Toc376663970)

[7.1 Requirement Analysis 15](#_Toc376663971)

[7.1.1 Software Requirements 15](#_Toc376663972)

[7.1.2 Hardware Requirements 15](#_Toc376663973)

[7.2 Feasibility Study 15](#_Toc376663974)

[7.2.1 Operational Feasibility 16](#_Toc376663975)

[7.2.2 Technical Feasibility 16](#_Toc376663976)

[System Design 17](#_Toc376663977)

[8.1 System Architecture 17](#_Toc376663978)

[8.2 Algorithm and SQL query Generation 18](#_Toc376663979)

[8.2.1 Building Clauses Sets 19](#_Toc376663980)

[8.2.2 Optimizing the dependency list 20](#_Toc376663981)

[8.2.3 Stem Categorization 20](#_Toc376663982)

[8.2.4 Select clauses 22](#_Toc376663983)

[8.2.5 Where Clauses 24](#_Toc376663984)

[8.2.6 From Clauses 24](#_Toc376663985)

[8.2.7 Composing Queries 26](#_Toc376663986)

[8.3 Data Dictionary and Database 26](#_Toc376663987)

[8.3.1 Data Dictionary 26](#_Toc376663988)

[8.3.2 Database 27](#_Toc376663989)

[8.5 Interface Design 28](#_Toc376663990)

[Testing and Experimental Evaluation 34](#_Toc376663991)

[9.1 Setup 34](#_Toc376663992)

[9.2 Testing 34](#_Toc376663993)

[9.2.1 Test Cases 34](#_Toc376663994)

[9.3 Some more examples 36](#_Toc376663995)

[9.4 Complete Trace of Interface 37](#_Toc376663996)

[Conclusion 39](#_Toc376663997)

[10.1 Future Work 39](#_Toc376663998)

[Appendix 41](#_Toc376663999)

[References 70](#_Toc376664000)

[Bibliography 71](#_Toc376664001)

LIST OF FIGURES

[Figure 1 SDP dependency list 4](#_Toc376608882)

[Figure 2 Logical view of Information Schema of MySQL database 6](#_Toc376608883)

[Figure 3 System Architecture 18](#_Toc376608884)

[Figure 4 Resulting SDCqopt 20](#_Toc376608885)

[Figure 5 Categorizing stems into projection and/or selection oriented sets 22](#_Toc376608886)

[Figure 6 Clauses generative grammar for fields matching stems in Π 22](#_Toc376608887)

[Figure 7 Extracting matching column of a table from IS 23](#_Toc376608888)

[Figure 8 Query Getting data type of column 23](#_Toc376608889)

[Figure 9 Relational Algebra's query for finding relation between two tables 25](#_Toc376608890)

[Figure 10 SQL query to find relationship between two tables from IS 25](#_Toc376608891)

[Figure 11 Data dictionary (left store synonyms, right store Functions and operators) 27](#_Toc376608892)

[Figure 12 Logical structure of database that interface is based 28](#_Toc376608893)

[Figure 13 Interface Diagram For user login 29](#_Toc376608894)

[Figure 14 Interface Diagram for Dashboard 29](#_Toc376608895)

[Figure 15 Interface Diagram for Query in NL 30](#_Toc376608896)

[Figure 16 Interface Diagram for Adding Functions and Operators 30](#_Toc376608897)

[Figure 17 Interface for listing functions and operators of data dictionary 31](#_Toc376608898)

[Figure 18 Interface Diagram for Updating Data Dictionary 31](#_Toc376608899)

[Figure 19 Interface Diagram for adding function to Data Dictionary 32](#_Toc376608900)

[Figure 20 Interface Diagram for showing functions and operators list 32](#_Toc376608901)

[Figure 21 Interface Diagram for updating functions and operators 33](#_Toc376608902)

LIST OF ABBREVIATIONS AND ACRONYMS

API Application Programming Interface

ATN Augmented Transition Network

CSIT Computer Science and Information Technology

DB Database (s)

DBMS Database Management System

FK Foreign Key

HDD Hard Disk

IS Information Schema

IT Information Technology

JDK Java Development Kit

NL Natural Language

NLI Natural Language Interface

NLP Natural Language Processing

NLIDB Natural Language Interfaces to Database (s)

PK Primary Key

QA Question Answering

RDBMS Relational Database Management System

RAM Random Access Memory

SDC Stanford Dependency Collapsed

SQL Sequential Query Language

SDP Stanford Dependency Parser

SDPopt Stanford Dependency Parser optimized

**Chapter – 1**

# Introduction

“Why aren’t computers easier to use?” inquired the unsuspecting beginner computer user. Such an easily posed question has many long and complicated answers [9].

“Why don’t you understand!” demanded the frustrated computer abuser. This is a situation that computer programmers and researchers spend their lives studying and trying to prevent [9].

One potential solution to these issues plaguing computer interface design is Natural Language Processing (NLP). The main goal of NLP is for an English sentence (or a sentence in any spoken language) to be interpreted by the computer and appropriate action taken. The sentence could be typed into the computer or obtained from a speech recognition program. Then the difficulty is to work out what the sentence means, whether or not some action should be taken, and what the appropriate action is.

The area of NLP research is still very experimental and systems so far have been limited to small domains, where only certain types of sentences can be used. When systems are scaled-up to cover larger domains, NLP becomes very difficult due to the natural ambiguity in spoken sentences, and the vast amount of information that needs to be incorporated in order to disambiguate such sentences [8]. For example, the sentence: “The woman saw the man on the hill with the telescope.” could have many different meanings. To understand what the intended meaning is, we have to take into account the current context, such as the woman is a witness, and any background information, such as there is a hill nearby with a telescope on it. Alternatively the man could be on the hill, and the woman may be looking through the telescope. All this information is very difficult to represent in the computer, so restricting the domain of an NLP system is the only practical way to get a manageable subset of English to work with.

The development of Natural Language Interfaces to Databases (NLIDBs) that translate the human intent into database instructions is indeed a classic problem, which is becoming of greater importance in today's world. In fact, despite the numerous attempts made in the past thirty years, current solutions are still not applicable in real scenarios. A huge variety of systems has been proposed, with increasing performance, but the task is still challenging due to the problem discussed above.

Talking about the NLIDBs is to draw the semantic from the question user poses to a system and identifying concepts and relationships between constituents and resolving ambiguities. E.g. consider a user asks the question to the NLIDBs “what is the highest salary of csit department”. Now the interface should translate the query to equivalent SQL query so that the appropriate answer can be withdrawn. The interface should translate to equivalent SQL query as:

SELECT MAX (salary)

FROM employee join department on employee.e\_id = department.e\_id

WHERE department like ‘%csit%’;

As the user asks the interface without knowing the logical structure of database so the interface should translate the query in appropriate SQL query by drawing the semantic and relationships between constituents in the sentence. The task become more complex when the user asks with more complex structure within the sentence.

**Chapter – 2**

# Background

Natural language is what humans use for communication. It may be spoken, signed, or written but for our purposes we consider only written text. In the field of question answering this text is typically modelled as questions or sentences paired with the corresponding answer. That answer may be again a natural language sentence, a machine readable instruction in an artificial language, some structured data like tables, images and graphs or raw values. The areas that are associated with this project are Natural Language Processing and Databases. This section provides a brief overview of these three areas.

## 2.1 Natural Language Processing

Natural language processing (NLP) is a field of computer science, artificial intelligence, and linguistics concerned with the interactions between computers and human (natural) languages. As such, NLP is related to the area of human–computer interaction. Many challenges in NLP involve natural language understanding -- that is, enabling computers to derive meaning from human or natural language input.

As we do not perform natural language understanding but we apply shallow semantics models. However the natural language processing (NLP) problem of automatically extracting meaningful information from natural language input in order to produce a meaningful output (e.g. the correct answer to a given question) is still demanding. The problem is that natural language grammar is ambiguous and typical sentences have multiple possible interpretations. At the same time, the expressiveness of a language allows to have many semantically equivalent sentences, i.e. syntactically different questions that have the same semantics. To cope with that we take into account only particular aspects of the natural language, considering only basic grammar relations holding between a subset of stems in a given NL question, relying only on syntax to derive the underlying semantics. This way we exploit fewer but enough information from the input NL question that is used by our mapping algorithm to find a mapping SQL query.

### 2.1.2 Stemming

In linguistic morphology and information retrieval, stemming is the process for reducing inflected (or sometimes derived) words to their stem, base or root form—generally a written word form. The stem need not be identical to the morphological root of the word; it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root. In our project we use snowball stemmer to stem the words.

### 2.1.2 Parsing

Parsing or syntactic analysis is the process of analyzing a string of symbols, either in natural language or in computer languages, according to the rules of a formal grammar. A grammatical analyzer then determines the relationship between the tokens (i.e. words of the input string) and builds a data structure. The output of this process can be a parse tree (syntax tree) or other hierarchical structure.

### 2.1.3 Stanford Dependency Parser

In our project the question is parsed by the parser obtained by the Stanford Core NLP API. The parser parse the natural language question or sentence into binary grammar relationship between the dependent and the governor as abbreviated\_relation\_name (governor, dependent). The governor and the dependent are words in the sentence associated with a number indicating the position of the word in the sentence. In particular we refer to collapsed representation, where dependencies involving prepositions, conjuncts, as well as information about the referent of relative clauses are collapsed to get direct dependencies between content words [7].

For example, the Stanford Dependencies Collapsed (SDC) representation for the question, “q: what is the highest salary of csit department?” is the following:

attr(is-2, what-1)

root(ROOT-0, is-2)

det(salary-5, the-3)

amod(salary-5, highest-4)

nsubj(is-2, salary-5)

amod(department-8, csit-7)

prep\_of(salary-5, department-8)

Figure 1 SDP dependency list

The current representation contains approximately 53 grammatical relations but for our purposes we only use the following: adverbial and adjectival modifier, agent, complement, object, subject, relative clause modifier, prepositional modifier, and root.

## 2.2 Databases

A database is an organized collection of data with implicit meaning. The data are typically organized to model relevant aspects of reality in a way that supports processes requiring this information. For example, modeling the availability of rooms in hotels in a way that supports finding a hotel with vacancies. Up to now different models of databases have been evolved, but currently Relational database model is dominant so we our concern is for relational databases.

### 2.2.1 Relational Database

A relationaldatabase is a database that has a collection of tables of data items, all of which is formally described and organized according to the relational model. Data in a single table represents relation, from which the name of the database type comes. In typical solutions, tables may have additionally defined relationships with each other. According to the relational model, data is organized into two-dimensional arrays called relations (database tables). Each relation has a heading and a set of tuples. The heading is an unordered possibly empty set of attributes (table's columns). Each tuple is a set of unique attributes and values (table's rows). Data across multiple tables is linked with a key, that is, the common attribute(s) between tables. The dominant language associated with the relational database is the Structured Query Language (SQL). In this research we experiment with relational database, about an organization including employee information, departments and locations.

### 2.2.2 Metadata and Information Schema

The DBMS manage the database(s) that resides in it by means of a storage engine. It stores all the information about the data (metadata) into internal data structures for the effective and efficient manipulation of data. In a relational-database metadata is stored into tables. Some examples are shown in Figure below and are basically the following:

* A table containing all tables names for every database, along with their size and number of rows.
* A table storing column names in each database, together with the information about which tables and database they are used in and the type of data they store.
* A table that keeps track of referred tables and columns by means of external keys.
* A table used to maintain database constraints to ensure database integrity.

And Information Schema is the database name for which the database store the information about other tables.

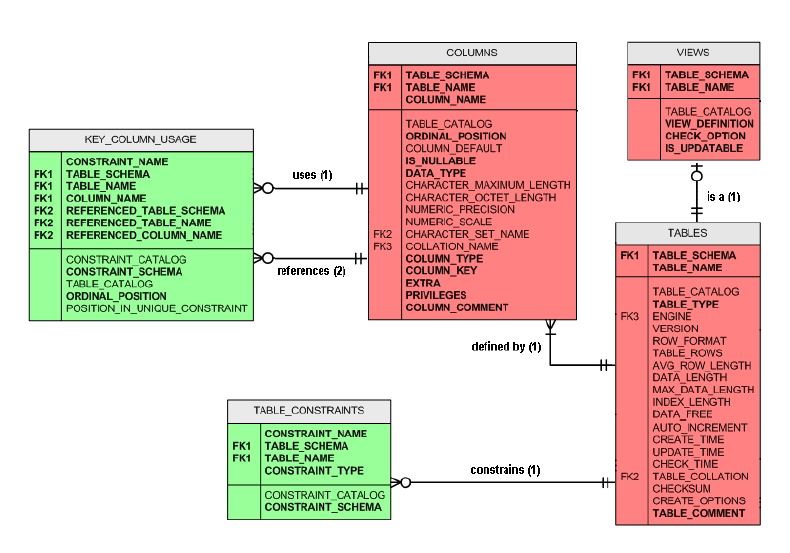


Figure 2 Logical view of Information Schema of MySQL database

## 2.3 Motivating Example

In order to illustrate how we got success within this project and how it was possible, we will demonstrate with a short example and briefly discuss the overall process. When designing the database the domain expert are suggested to design the table and fields with semantic meaning rather than like tab\_1, fld\_2 etc. The IS can be queried like a normal database for obtaining useful meta information about tables and databases by the DBMS, so we can also do that by posing the SQL queries to it and retrieve answer.

Let us consider a question “what is the highest salary of csit department”. The logical structure related to this question can be depicted in the figure 12. With the sentence above we got the matching words salary and department with the database that we are processing. Thus the salary is the column name and department is also a column name can be proved by querying the IS as normal database. Furthermore the table associated with these two columns can also be obtained by querying IS. As the question is demanding highest salary so there is no direct mapping of highest salary but the word highest is also tracked. The mapping of word highest may related with the single row aggregation functions like MAX. Thus finding the data type of the column salary the appropriate function is noted. As the word ‘csit’ is neither a column nor table so it may be the possible candidate of the where clause which is confirmed by querying the IS. Therefore, finally we are able to get the mapping of the above question to equivalent SQL query as:

SELECT MAX (salary)

FROM department JOIN employee ON department.dep\_id = employee.dep\_id

WHERE department LIKE ‘%csit%’;

**Chapter - 3**

# Problem Statement

Most of the users who interface database are normal users not technical or are novice users. Additionally most of them are not aware of high level SQL that the all database support for information retrieval or for querying the database. Thus they have to rely on the highly skilled user for retrieval of the information as they desire for their business or other analysis. Therefore this has led to a core problem in data mining for retrieving data in an easy and human friendly way. Even the sensitive information stored in database is also exposed and the users can’t kept their privacy as they have to rely on third person for information retrieval by paying handsome amount of cash. As this age the data and data is considered as the wealth for any firm but due complexity of SQL the data has been exposed to third party. In addition there is no guarantee that the third party may or not use or exploit the data of the analyzed firm for their personal benefit. Therefore with consideration with these factors and issues led us to develop the NLIDB.**Chapter - 4**

# Objective

As the NLIDB has wider domain but has only specific objectives which are:

* To develop an interface to process natural language as a query language to query the database.
* To allow the non-technical users to query in database for information retrieval forgetting the complex SQL queries.
* To maintain privacy of data.
* To replace the current Form Interface for Database with Natural Language Interface for Database.

**Chapter - 5**

# Scope and Limitation

## 5.1. Scope

System scope is one of the important factor to be considered while developing any system. System scope involves getting information required to start a project, and the features the product would have that would meet its stakeholder’s requirements. Thus the scope of our proposed system are given below:

* The natural language used is English so the input statements should be in English.
* Input form user is taken from user in any form of Natural Language sentence.
* Data Dictionary is used where all possible words related to particular system will be induced. The Data Dictionary is used to increase the domain of the system and must be regularly updated with words specific to the particular system.
* Ambiguity among the words will be taken care of while processing the natural language.
* A predefined database is used.
* Users are not necessarily should be known with the internal structure of the database by the query they possess must have semantic words related to the database or the interface that is currently serving.
* With consideration of time the system is only able to answer queries of select statements.
* To work with any RDBMS one should know the syntax of the commands of that particular database software (Oracle, MySQL, and Microsoft SQL etc.).

## 5.2. Limitation

The limitations with the current RDBMS which led us to develop the NLIDB are:

* Most user who interact database are normal users and do not have prior knowledge about query languages to query the database.
* Complexity in the SQL to query the database for information retrieval.
* No privacy of data of any firm or person as most of users are novice and normal user who don’t have knowledge about complex SQL queries.
* External skilled user is needed for the information retrieval, thus the firm or person has to pay a handsome amount of money.
* Chances of the misuse of the data when it is exposed to the third person.

**Chapter – 6**

# State of the Arts

There has been much work on NLP recently, but the area has been around for a relatively long time in the computing world. The main aim of NLP research is to create a better interface to the computer. Spoken language is the most natural interface available for humans to use, but computers are still unable to come close to the rich communication humans can achieve with each other. Science fiction has created many robots or computers that are able to understand and carry out tasks based on spoken orders or communication. So a huge variety of systems has been proposed, with increasing performances, but the task is still challenging.

## 6.1 Spoken Language Understanding

The following paragraph illustrates the techniques used by some spoken language systems that understand natural language in some extent.

* SHRDLU – by Terry [11]. This is one of the first programs that could carry out tasks and provide responses in natural language well. It was bound within an artificial blocks world of colored bricks and pyramids. SHRDLU was able to perform tasks like moving objects around within the limited world, when directed to do so in English. The program used a procedural representation for semantics. This means that each English predicate or term was associated with a procedure that conveyed the meaning (or semantics) of the term. The problem with procedural semantics is that they do not scale up into large domains.
* Phoenix uses a bottom-up semantic parser to build trees from the sentences and then extract information into slots of frames. These frames, which describe semantic entities (e.g. light, airport, time and other concepts), are then used to produce the corresponding SQL queries.

## 6.2 Question Answering in NLI

The ultimate way to handle the problem and allow to query very large data is the implementation of natural language interfaces (NLI). In particular, natural language interfaces to database (NLIDBs) take natural language questions (whole sentences as well as keywords) and translate the user intent into machine-readable instructions to retrieve the answers.

The very first attempts at NLP database interfaces are just as old as any other NLP research. In fact database NLP may be one of the most important successes in NLP since it began. Asking questions to databases in natural language is a very convenient and easy method of data access, especially for casual users who do not understand complicated database query languages such as SQL. The success in this area is partly because of the real-world benefits that can come from database NLP systems, and partly because NLP works very well in a single-database domain. Databases usually provide small enough domains that ambiguity problems in natural language can be resolved successfully.

Here are some examples of database NLP systems:

* LUNAR [4] involved a system that answered questions about rock samples brought back from the moon. Two databases were used, the chemical analyses and the literature references. The program used an Augmented Transition Network (ATN) parser and Woods' Procedural Semantics. The system was informally demonstrated at the Second Annual Lunar Science Conference in 1971. Its performance was quite impressive: it managed to handle 78% of requests without error, a figure that rose to 90% when dictionary errors were corrected. This figure is misleading because the system was not subject to intensive use. A scientist who used it to extract information for everyday work would soon have found that he wanted to make requests beyond the linguistic ability of the system. ATN parsers are useful because they are very efficient, even for large grammars; however, ungrammatical sentences are not handled well and they are not very flexible.
* LIFER/LADDER was one of the first good database NLP systems. It was designed as a natural language interface to a database of information about US Navy ships. This system, as described in a paper by Hendrix [10], used a semantic grammar to parse questions and query a distributed database. The LIFER/LADDER system used a semantic grammar (that is, it used labels such as "SHIP" and "ATTRIBUTE" rather than syntactic labels such as noun and verb). This NLP systems using semantic grammars are closely tied to the domains for which they were designed, and they can be easily adapted to suit new terms or phrases. Even today the same general method is still being used; semantic grammars are now widely used in most NLP systems, but there are many variations and new approaches are continually being developed. Akama [12] describes some variations on semantic grammars, including Montague semantics and operational semantics, which can support different forms of logic, and reasoning with incomplete information.
* EasyAsk2, also known as English Wizard, is a commercial application that offers both keyword and natural language search over relational databases. The system crawls the data to automatically construct a contextual dictionary used to identify words that correspond to values or catalog attributes and generate an SQL statement. It incorporates approximate word matching, stemming and synonyms.
* EQ, which stands for English Query [13] is a NLIDB implemented by Microsoft Corporation, as a part of the SQL Server. It creates a model, collecting database objects (tables, fields, joins) and semantic objects (entities, additional dictionary entries, etc.). However, it only extracts few basic relationships and, thus, requires refining the model manually.
* Cleverbot is a web application that uses an artificial intelligence algorithm to converse with humans.  Cleverbot's responses are not programmed. Instead, it "learns" from human input; Humans type into the box below the Cleverbot logo and the system finds all keywords or an exact phrase matching the input. After searching through its saved conversations, it responds to the input by finding how a human responded to that input when it was asked, in part or in full, by Cleverbot.
* Siri is a personal assistant and knowledge navigator which works as an application for Apple Inc.'s iOS. The application uses a natural language user interface to answer questions, make recommendations, and perform actions by delegating requests to a set of Web services.  It integrated with services such as OpenTable, Google Maps, MovieTickets and TaxiMagic. Using voice recognition technology from Nuance and their service partners, users could make reservations at specific restaurants, buy movie tickets or get a cab by dictating instructions in natural language to Siri.

**Chapter – 7**

# Requirement Analysis and Feasibility Study

## 7.1 Requirement Analysis

Requirements analysis involves frequent communication with system users to determine specific feature expectations, resolution of conflict or ambiguity in requirements as demanded by the various users or groups of users, avoidance of feature creep and documentation of all aspects of the project development process from start to finish. Requirements analysis is a team effort that demands a combination of hardware, software and human factors engineering expertise as well as skills in dealing with people. So to get the expected result that our interface have provided, the users must fulfill these requirement. Hence we have divided the requirement analysis in two topics and they are described and listed as:

### 7.1.1 Software Requirements

Users who are installing this project must have the following software requirements in order to run the system and get the proper result.

* Operating system : Windows XP, Windows 7 , Windows 8, Ubuntu > 11.4 version , Fedora > 14 and other Linux distribution with Linux kernel > 3.0, Apple
* JDK 1.7
* MySQL server version 5 or greater
* Stanford Core NLP API version 3.0

### 7.1.2 Hardware Requirements

The hardware requirements for this project are stated as below:

* RAM 1GB or greater
* Minimum free space 5GB or greater in HDD
* Processor with minimum 32 bit architecture and Pentium 4 or equivalent to Pentium 4

## 7.2 Feasibility Study

The feasibility study is an evaluation and analysis of the potential of the proposed project which is based on extensive investigation and research to support the process of decision making. Feasibility studies aim to objectively and rationally uncover the strengths and weaknesses of an existing business or proposed venture, opportunities and threats present in the environment, the resources required to carry through, and ultimately the prospects for success. In its simplest terms, the two criteria to judge feasibility are cost required and value to be attained. So we have carried out the feasibility analysis for our project on the basis of time, cost, and operation which are described in short in below section.

### 7.2.1 Operational Feasibility

Operational feasibility is a measure of how well a proposed system solves the problems, and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development. So our project is operationally feasible. As it solves about 95% of queries as stated in the scope of our project. The system has been proved more feasible when the interface is asked question using imperative sentences. As we have increased our domain and its operation using the data dictionary created by ourselves and the data dictionary poses semantic information related to the database that the interface is querying.

### 7.2.2 Technical Feasibility

The technical feasibility assessment is focused on gaining an understanding of the present technical resources of the organization and their applicability to the expected needs of the proposed system. The project that we have developed is technically feasible. As we have experienced in programming languages like PHP, c# and JAVA so technical manpower has no problem. As we have efficient and powerful working PC’s and Laptops for implementation and research.

**Chapter – 8**

# System Design

Systemsdesign is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. Systems design could be seen as the application of systems theory to product development. Systems design is therefore the process of defining and developing systems to satisfy specified requirements of the user. Object-oriented analysis and design methods are becoming the most widely used methods for computer systems design. So within this section we are mainly focused on architecture, interfaces, data and its algorithm.

## 8.1 System Architecture

In this section we had given the conceptual model of our project which describes the structure, behavior and more views of the system. A system architecture can comprise system components, the externally visible properties of those components, the relationships (e.g. the behavior) between them. It can provide a plan from which products can be procured, and systems developed, that will work together to implement the overall system.

Our architecture is divided into five components as shown in below figure. First a query in Natural Language is taken and the query is passed to the syntactic parser which parses the sentence into a collection of dependencies and it is passed to the semantic parser which looks for the semantic among the collection of dependencies using the conceptual semantic rules, data dictionary and the information schema. The data dictionary contains the more information about the database so it increase the domain of the database that the project is currently processing. Finally the query generator assembles the result of the semantic parser and result the final query.



Figure 3 System Architecture

## 

## 8.2 Algorithm and SQL query Generation

In the perspective of question answering (QA) targeting the information of databases (DBs), the automatic system only needs to execute one or more Structured Query Language (SQL) queries that retrieve the answer to the posed natural language question. This does not necessarily imply that a semantic parser has to be designed for mapping the meaning of the questions to the one of the queries. This chapter will demonstrate that it is possible to avoid full-semantic interpretation by relying on (i) a simple SQL generator, which both exploits syntactic lexical dependencies in the questions along with the target DB metadata [2]; and (ii) advanced machine learning such as kernel-based rankers or others [5], which can improve the initial candidate list provided by the generative parser.

The idea of point (i) can be understood by noting that database designers tend to choose names for entities, relationships, tables and columns according to the semantics of the application domain. Such logic organization is referred to as catalog, and in SQL systems it is stored in a database called Information Schema [1].

The values stored in IS along with their constraints and data types are important metadata, which is useful to decode a natural language question about the DB domain in a corresponding SQL query. For example, given the IS associated with a DB, shown in Figure 1, if we ask a related question, q0: What is the salary of employee ram? A human being will immediately derive the semantic predicate salary (employee, ram) from it. Then she/he will associate the argument salary, which is also the question focus, with the table EMPLOYEE. Once the latter is targeted, she/he will select the column SALARY provides the same predicative relation asked in the question. Finally, by instantiating the available argument, ram, in such predicate, she/he will retrieve the answer as 2000 from the column SALARY.

In a NL interface to a SQL database, if we want to generate all possible queries for a question q we first need to find their possible SELECT, FROM and WHERE clauses (S;F and W sets) and then combine them in a smart way such that the resulting queries are all syntactically correct and meaningful. Generated queries can be ordered based on some heuristics but if we train a support vector machine [5] and then use it as a reranker [5] we can improve the probability of finding the answer in the top position but we are not this is not in scope of our project.

### 8.2.1 Building Clauses Sets

The basic idea of our generative parser is to produce queries of the form:

∃ s ∈ S, ∃ f ∈ F, ∃ w ∈ W s.t π s ( σw ( f)) answers q, (8.1)

Where question q is represented by means of SDCq and S;F;W are the three sets of clauses (argument of SELECT, FROM and WHERE). The answering query, π s ( σw ( f)) can be chosen among the set of all possible queries A = { SELECT s × FROM f × WHERE w } in a way that maximizes the probability of generating a result set answering q. This section shows how we extract terms from the question and categorize them in order to build the sets of clauses above and then how to combine them for generating a query candidate also associated with a generation score.

In Section 2.1.3 we introduced the dependency list (SDCq) for NL question “what is the highest salary of csit department?” Indeed, given a question q we start from its SDCq and (a) prune and stem its components, (b) add synonyms, (c) create the sets of stems select and/or project oriented and (d) keep only dependencies possibly used in the recursive step to generate nested queries. Finally (e) we look for matching stems both in the metadata and in the database to build S and W. Building the set F from S and W is straightforward.

### 8.2.2 Optimizing the dependency list

As introduced in Section 2.1.3, we do not use all grammatical relations provided in output by the Stanford Dependency parser. For this reason before processing the list of dependencies we filter it pruning useless relations and removing from governors and dependents the appended number indicating the position of the word in question q. Moreover we eliminate stems of 3 or less characters since they would introduce too much noise in retrieving matching strings. In contrast, useful function words such as in, of, not, or, and, etc. are embedded in the dependency type, e.g., prep\_of(salary-5, department-8). Then, govs and deps are reduced to stems (as discussed in Section 2.1.1). Finally, we enlarge the probability to match stems and the metadata by means of substring matching.

We call the preprocessed list SDCqopt and we use it for the next step. For example, with respect to the original SDCq introduced in Section 2.1.3, the optimized list is the following:

root(ROOT-0, is-2)

amod(salary-5, highest-4)

nsubj(is-2, salary-5)

amod(department-8, csit-7)

prep\_of(salary-5, department-8)

Figure 4 Resulting SDCqopt

### 8.2.3 Stem Categorization

To build S and W sets, we identify the stems that can most probably participate in the role of projection (i.e., composing the SELECT argument) and/or selection (composing the WHERE condition). Accordingly, we create two sets of terms Π (possible select statements) and Σ (possible where clause statements). The main idea is that some terms can be used to choose the DB table and column where the answer is contained whereas others tend to indicate properties (i.e., table rows) useful to locate the answer in the column. For example, in case of q (see introduction): highest salary may indicate that salary is a SELECT argument whereas department and csit may be part of the WHERE argument, thus forming the query:

SELECT max (salary) FROM employee join department using (e\_id) WHERE department like ‘%csit%’.

**The Algorithm**

We use SDCq to automatically extract and classify such terms and relations from a question. For example, the dep of root is typically the main verb (i.e., relation) of the question, which can be used to derive properties of the question focus. Thus, it tends to be of type selection (i.e, it belongs to Σ set) like in root(ROOT, is). In case of an nsubj, the gov is typically a verb relation and it can be used to build specializers whereas the dep, i.e., the subject, is most probably a projection candidate such as for example in nsubj(is, salary). In the following, we report our heuristics for term categorization. It should be noted that they do not produce a precise and disjoint term separation but the obtained two sets are smaller than the overall term set, thus reducing the computational complexity in generation. We analyze grammatical dependencies rel (gov,dep) in SDCq in their parse order and classify their arguments (stems) in Σ and Π categories according to the following rules :

1. If it is ROOT, dep is the key to populate W so add it to Σ and remove the relation from SDCoptq . Set the flag hasRoot to true. This stem can be an auxiliary verb, e.g., is, are, has, have and so on. It is useless to build the arguments of the queries but it could be used transitively to add other stems.
2. If it starts with nsubj, we use it to add stems to Π. Set the flag hasSubj to true.

* If gov ∈ Σ add dep to Π and remove rel from SDCoptq.
* Otherwise if gov ∉ Σ and hasRoot is false, add gov to Σ and dep to Π and remove rel from SDCoptq.
* Otherwise keep it, since it could be a subject related to a subordinate (we will need it in the recursive steps).

1. If it starts with prep or it ends with obj, we used it to create conditions (possibly involving nesting):

* if gov ∈ data dictionary , it may have aggregation functions or operator , so check if it is associated with select clause then add gov-dep to Π and remove rel, other wise add gov-dep to Σ.
* otherwise gov ∈ Π and if there is no table.column like gov.dep add dep to Σ, otherwise, also add dep to Π. Remove rel from SDCoptq .
* otherwise if gov ∉ Π hasRoot and hasSubj are false add gov to Π. If there is not any table.column like gov.dep add dep to Σ, otherwise, also add dep to Π. Remove rel from SDCoptq.
* otherwise keep it, since we will need it in the recursive steps.

1. If it ends with mod, it implies that dep is a modificator of gov, so they should be paired together: if gov ∈ Σ add dep to Σ and if gov ∈ Π add dep to Π and remove rel from SDCoptq. This should be done only if dep is not a superlative (i.e. doesn't end with -st). Still if above conditions are not fulfilled then check the occurrence of dep in data dictionary. If true add dep-gov to Σ or Π based on the status of data dictionary. The non-removed dependencies will be taken into account in the recursive step, adding both dep and gov to Π.
2. If it starts with nn then modifier of an NP is any noun that serves to modify the head noun so add dep to the Σ and the rel is removed.
3. If it starts with conj then it gives the relation between two consecutive words in sentence so add dep to Π and remove rel (as our system only supports only one selection in where clause so it is added to the projection).
4. If none of the above rules can be applied, iterate the algorithm recursively until SDCoptq is empty.

### 8.2.4 Select clauses

We use the set Π to retrieve all the metadata terms that match with its elements: this will produce S according to the generative grammar shown in Figure 5.3. The arguments of the grammar are derived by executing several queries to find all matching stems and retrieve a list of table.column terms augmented by aggregation operators. For example,

root(ROOT-0, is-2) Π={}, Σ={is}

amod(salary-5, highest-4) Π={highest-salary}, Σ={is}

nsubj(is-2, salary-5) Π={highest-salary,salary}, Σ={is}

amod(department-8, csit-7) Π={highest-salary,salary}, Σ={is,csit}

prep\_of(salary-5, department-8) Π={highest-salary,salary}, Σ={is,csit, department}

Figure 5 Categorizing stems into projection and/or selection oriented sets

considering the IS scheme in Figure 2, the SELECT clauses that are generated from Π, whose elements are listed in the right side of Figure 5.

S →AGGR '(' FIELD ')' │ FIELD

AGGR→ max│min│sum │count│avg

FIELD →TAB.COL\*│TAB\*.COL

TAB ∈ ∪ x ∈ Π π table\_name (σ table\_name = x (IS:Tables))

COL ∈ ∪ x ∈ Π π column\_name (σcolumn name = x (IS:Columns))

TAB\* ∈ ∪ x ∈ Π π table\_name (σcolumn name = x (IS:Tables))

COL\* ∈ ∪ x ∈ Π π column\_name (σtable name = x (IS:Columns))

Figure 6 Clauses generative grammar for fields matching stems in Π

**Extracting Matching Fields**

According to this grammar, the stems can map with table names as well as with column names or may be with the functions enlisted in data dictionary. To retrieve such fields from the metadata, the following SQL query is executed over the database catalog every time we need to look up for a matching with stem x in the metadata of a fixed DB.

SELECT table\_name,column\_name

FROM INFORMATION\_SCHEMA.columns

WHERE (table\_name LIKE "%x%" OR column\_name LIKE "%x%")

AND table\_schema = "DB"

Figure 7 Extracting matching column of a table from IS

This query looks for a partial (substring) matching of the stem x among all the table names (retrieving also all the columns belonging to the matched tables) and among all the column names (extracting also the names of the table to which they belong). The result set consist of a two column table, and the partial list of fields’ S is obtained linking each row with the dot separator.

**Adding Aggregation Operators**

In the previous step we obtain a list of fields’ S in the format C that can be extended adding some aggregation operators based on the type of the fields. If the field type is compatible with numbers we apply all the operators (sum, average, minimum and maximum) and the functions are stored in the own created data dictionary with their synonyms like highest for max, lowest for min etc and are mapped to respective aggregation functions on the basis of result of querying data dictionary and query of figure 8. Otherwise, if it is a textual field it makes sense to apply only the operator that counts how many different values appear in the field. The DB catalog can be inspected in the following way to retrieve the subset of textual fields that are extended adding COUNT(C).

SELECT table\_name,column\_name

FROM INFORMATION\_SCHEMA.columns

WHERE table\_name = T AND column\_name = C

AND (data\_type = "TEXT" OR data\_type LIKE "%CHAR%")

AND table\_schema = "DB"

Figure 8 Query Getting data type of column

All the other fields that are in S but not in the result set of this query, are extended adding clauses SUM(C), MAX(C), MIN(C), AVG(C).

### 8.2.5 Where Clauses

For generating the WHERE clauses, we need to divide Σ in two distinct sets: ΣL and ΣR, for the left-and right-hand side of the condition, respectively. The set ΣL contains stems matching the IS metadata terms. ΣL is used to generate the left condition WL, with the rule WL → FIELD, where FIELD is the same of Figure 5.6, where ΣL is used in place of Π (this is the same task as illustrated in Section 8.1.4). In contrast, ΣR = Σ - ΣL is used to generate WR.

**Creating Expressions**

To build the WHERE clauses set W, we first generate basic expressions in the form expr = eL OP eR, ∀ eL ∈ WL, ∀ eR ∈ WR. If the type of eL is numerical then OP= {< ; > ; =}, otherwise we apply the LIKE operator. To better understand how it works, let us introduce a new example question q: “what is the highest salary of csit department”. The list SDCoptq and the derived sets of stems are shown in Figure 5. The set Σ is split into ΣL = department and ΣR =csit. We build: WL = department and WR = csit. So now with the query to the IS we get the data type of the department is csit so the final where clause is: department like‘%csit%’. The comparison operators are taken special care, for example “who are employees with salary greater than 2000”. Then the greater then must imply ‘>’ so for semantic mapping the data dictionary provides the mapping as e.g. greater >, less < etc.

**Dealing with Missing Pieces**

It could happen that the set ΣR is empty. So it implies that the parser didn’t find any constraint that can be applied to the query. So in this case the generative algorithm assigns null to the where clause. For e.g. suppose a question: ‘list name and phone of employees’. So the generative parser doesn’t assigns any word for the above question to the Σ. So in this case the equivalent SQL is:

SELECT name, salary FROM employee

So our parser also does the same and assigns null value to the where clause.

### 8.2.6 From Clauses

The generation of the FROM clause F is straightforward given S and W. This set will contain all tables to which clauses in S and W refer, enriched by pairwise joins. As stated before, this information can be found running SQL queries over IS exploiting metadata stored in table KEY COLUMN USAGE (in short, Keys; as in Figure 2). This table identifies all columns in the current databases that are restricted by some unique, primary key, or foreign key constraint. That is, for each usage of foreign key column in the table, we can determine how many aggregate table columns match that column usage.

**Retrieving Used and Useful Tables**

First of all, we extract tables appearing in S and W. This is performed when the where and select clause are extracted. Every time when IS is queried for columns check its associated table is appended to the F set either building S or building W, thus creating a set F.

**Looking up in Metadata for Table Joins**

So this section deals how to extract joins if exists between tables. Whenever the size of the set F is greater than 1 then it is eligible for checking join. So we explicitly query the IS for checking if there is relationship between this two tables. For the tables in the set F t1 and t2 we query the IS with the following query to check if relationship exists between two tables or not.

π table\_name, column\_name, referenced\_table\_name, referenced\_column\_name (

σ table\_schema = 'project' ^ ( table\_name = t1 ∨ table\_name = t2) ^ (referenced\_table\_name = t1 ∨ referenced\_table\_name = t2) ^ referenced\_column\_name !=null (IS.KEY\_COLUMN\_USAGE))

Figure 9 Relational Algebra's query for finding relation between two tables

Now for e.g. going back to our previous query “what is the highest salary of department csit”. This consists of two tables as building select clause it add employee table as salary is a column of employee table and while building where clause the WL consists of department which is the column of department table so it also appends department table to the set F. So our generative parser query the database catalog (IS) with the following query:

SELECT table\_name, column\_name, referenced\_table\_name, referenced\_column\_name"+

FROM KEY\_COLUMN\_USAGE

WHERE table\_schema = 'project'

AND ( table\_name = 'employee' OR table\_name = 'department') AND ( referenced\_table\_name = 'employee'OR referenced\_table\_name = 'department')

AND referenced\_column\_name IS NOT NULL;

Figure 10 SQL query to find relationship between two tables from IS

If this query returns result then it says that there is relationship between two tables with the following result:

table\_name = employee , column\_name = d\_id , referenced\_table\_name = department , referenced\_column\_name = d\_id

So finally we result the FROM clause as

FROM table\_name join referenced\_table\_name ON table\_name.column\_name = referenced\_table\_name. referenced\_column\_name;

As the final clause for above question the parser returns as

FROM employee join department ON employee.d\_id = department.d\_id

### 8.2.7 Composing Queries

In the previous section we saw how to create building blocks for queries starting from a question q. These elements should be paired together in a smart way to generate the set of queries that possibly answer q. This pairing is obtained joining the clause form the respective clauses. Sometimes it may be the situation that the select statement has no elements to select in this case the parser selects all rows whenever there are contents in set F. Likewise the where clause may return null so in this case no filter is added to the query (means where clause is pruned). As going with our previous question q the clauses returned by the parser is:

select clause : SELECT max (salary)

from clause : FROM employee JOIN department ON employee.d\_id = department.d\_id where clause: WHERE department LIKE ‘%csit%’

So finally the all clauses are bind together to build a query as

SELECT max (salary)

FROM employee JOIN department ON employee.d\_id = department.d\_id WHERE department LIKE ‘%csit%’

## 8.3 Data Dictionary and Database

We have created own data dictionary to extend the domain of the interface. The interface is related to the database of an organization which consists of employees, department and their locations. This section provide the brief summary of the logical structure of both the data dictionary and database the interface is based on.

### 8.3.1 Data Dictionary

A data dictionary is a centralized repository of information about data such as meaning, relationships to other data, origin, usage, and format. So we have created our own data dictionary which is database specific and stores the information about the columns names and functions that may appear in the natural language question. For this project we have created two tables which serve as data dictionary. Among them one store the information about synonyms about the columns that may appear in the natural language question. The other table store the information about the functions and operators such as greater, highest, lowest etc. The logical structure of them are depicted as below in figure.

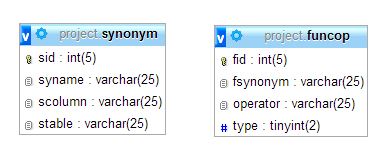


Figure 11 Data dictionary (left store synonyms, right store Functions and operators)

The left hand table store the synonyms of the columns like employee’s synonym may be name, location may be city etc. Likewise the table of right hand store the entities that may take part in either select or where clause. The entries in this table may be like highest may be function like max which may appear in select clause, greater may be the comparison may represent operator like >. The use of data dictionary during mapping is as described:

Suppose a question ‘what is the name of employee whose city is Kathmandu’. So with question the parser pares the sentence and divides into the list of dependencies then the parser checks for the synonym for columns so, it checks synonym for name and city and replaces the name and city with employee and location as per the column of the table with the help of data dictionary.

### 8.3.2 Database

The interface is associated with the only one database so the user can query to a specific database and database tables of that single database. Now for our project we have considered the database of an organization which stores information about its employees, departments and their locations. The interface is independent of database tables. So it can serve to a single database with more than 2 tables. For more practical we have cerate database table with implicit semantic i.e. the table and its column are created like employee, phone, email not like col1, col2. So the database we have used for sample consists of three tables whose logical structure is described as below in the figure.

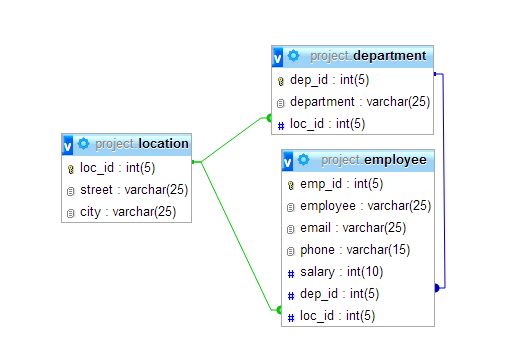


Figure 12 Logical structure of database that interface is based

## 8.5 Interface Design

Interface design deals with the process of developing a method for two (or more) modules in a system to connect and communicate. These modules can apply to hardware, software or the interface between a user and a machine. An example of a user interface could include a GUI, a control panel for a nuclear power plant. Moreover user interface is very important part of almost any software because it allows user interaction with the system in a convenient way. Thus this section gives the GUI interface description that the NLIDB possess with some of the snapshots of the user interface.

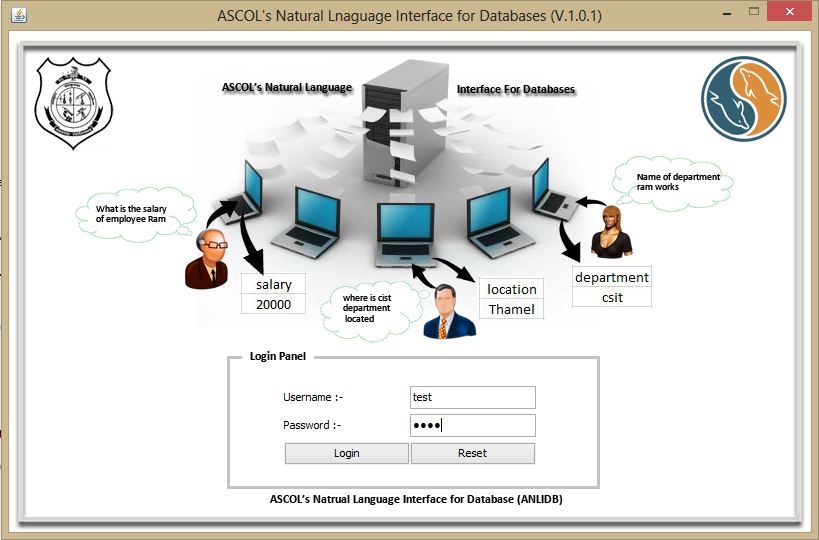


Figure 13 Interface Diagram For user login

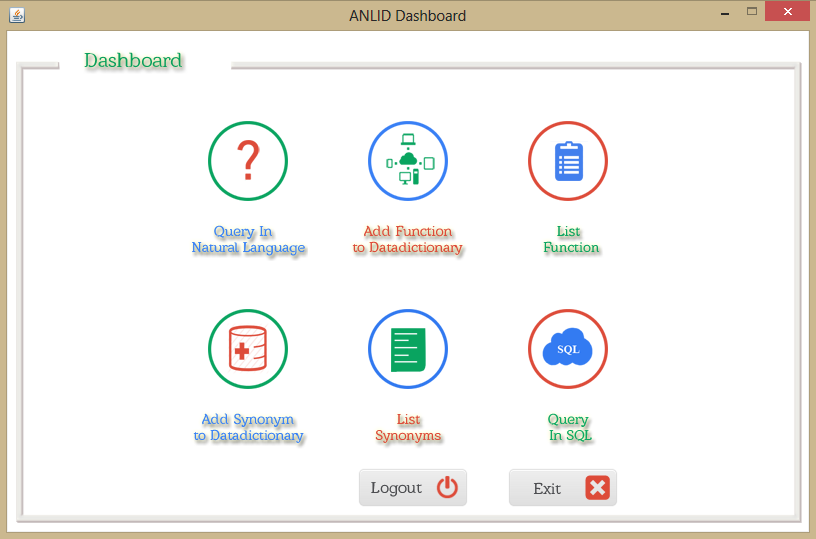


Figure 14 Interface Diagram for Dashboard

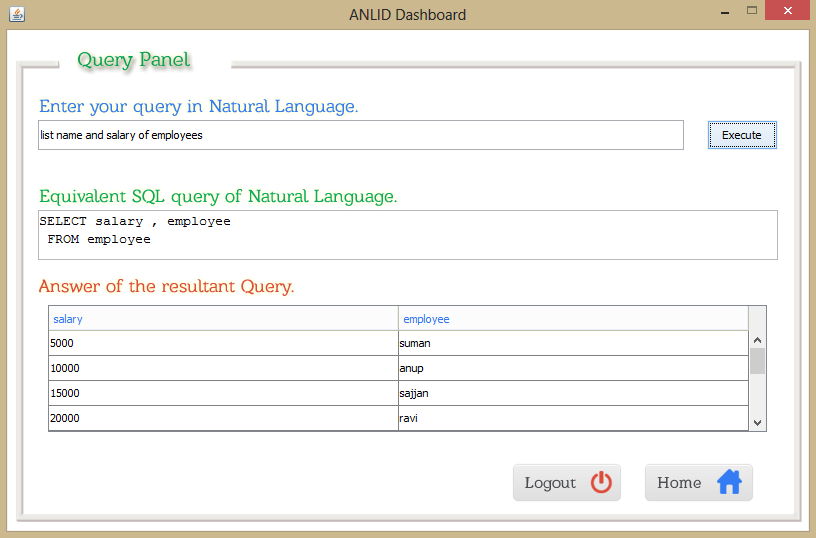


Figure 15 Interface Diagram for Query in NL

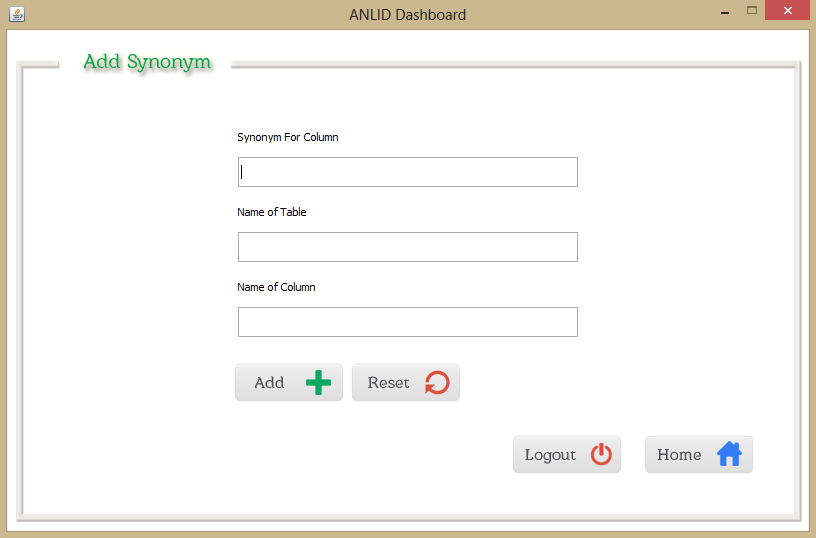


Figure 16 Interface Diagram for Adding Functions and Operators

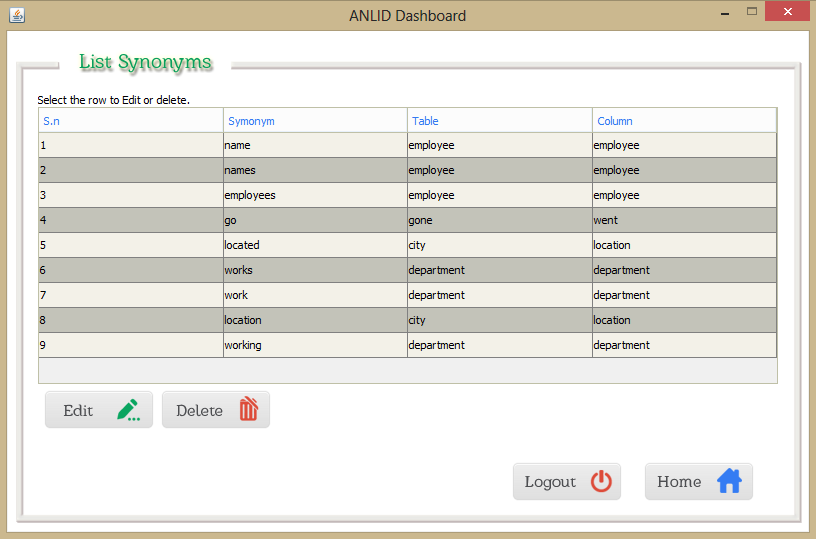


Figure 17 Interface for listing functions and operators of data dictionary

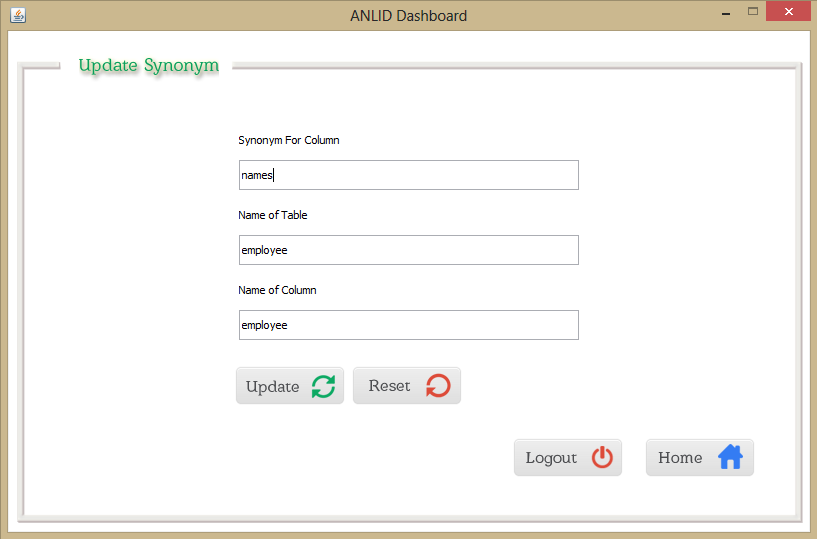


Figure 18 Interface Diagram for Updating Data Dictionary

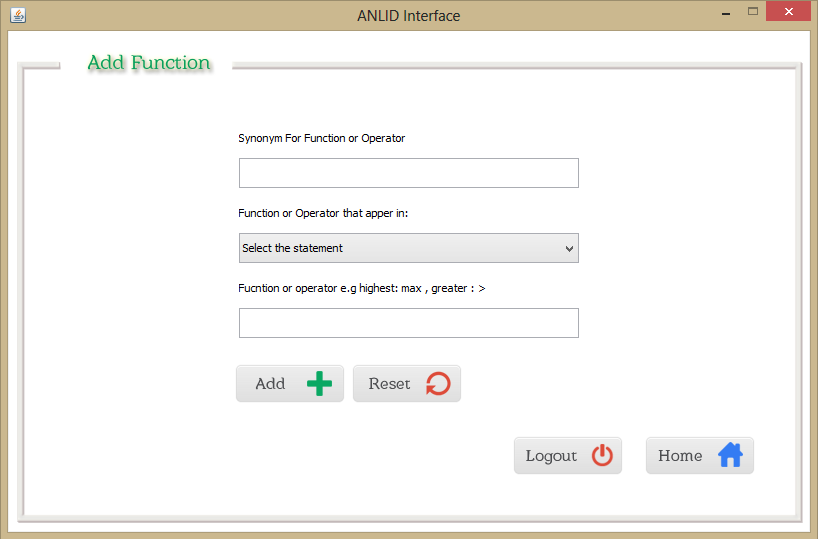


Figure 19 Interface Diagram for adding function to Data Dictionary

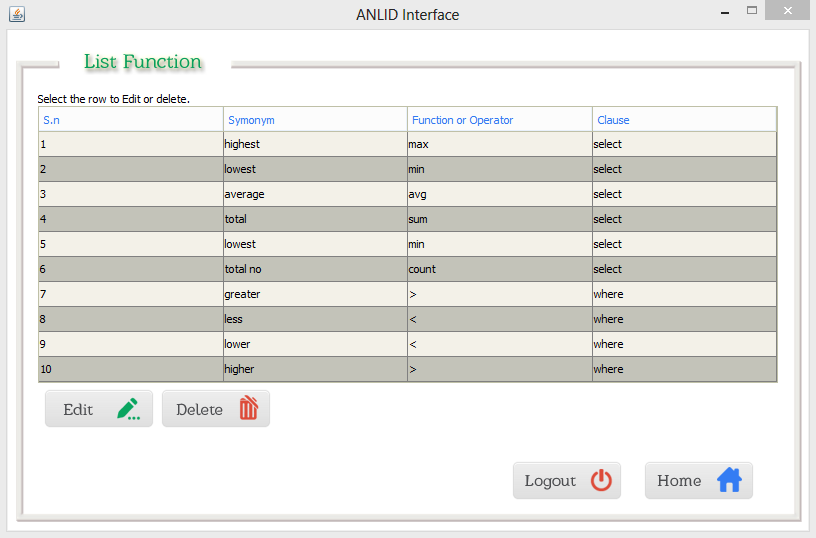


Figure 20 Interface Diagram for showing functions and operators list

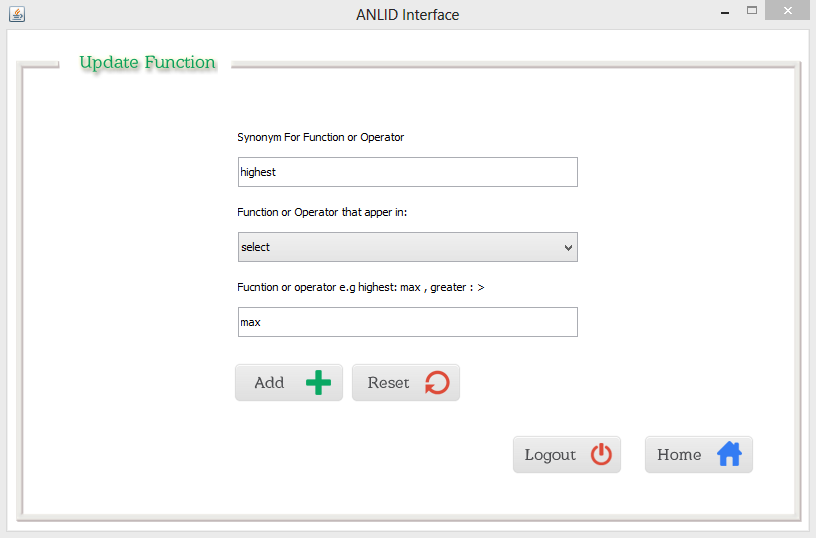


Figure 21 Interface Diagram for updating functions and operators

**Chapter – 9**

# Testing and Experimental Evaluation

In previous sections we described the algorithm and the system design process of our interface. This section gives the brief of the testing and the evaluation of the project. Thus in this section will be describing the efficiency of the generative parser that the interface is based on.

## 9.1 Setup

Moreover we need a database that the interface should process. Therefore we considered a database of an organization as described in figure 12. Thus after developing and integrating our generative approach (Chapter 8), we need to populate the database with the data with the database so that we ran several experiments to evaluate the accuracy of our approach for automatic generation and selection of correct SQL queries from questions. We populated the database with the entries considering the data of our college.

## 9.2 Testing

With the setup of the interface we begun the testing of the results that is generated by the parser. As the users are allowed to ask question in any type of sentence. Therefore the result was varied for the type of sentence that a user uses to ask the query to the interface. Moreover testing with the interface we found that too much use of proposition, conjunctions and determinants with in the sentence brings too much noise in sentence and the parser get confused which cause the parser not able to get the respective mapping SQL. We also found that whenever the complexity in the NL sentence increases the parser is not able to get the equivalent mapping. With the testing of the queries in NL we found that the sentences with the appropriate use of conjunctions, prepositions and determinants are almost completely answered by our generative parser.

### 9.2.1 Test Cases

We tested our system by asking it with different natural languages questions with and without semantic and introducing noise with in the question and analyzed the result as the following cases:

**Case 1: Question without noise and full semantic**

Question: “what is the highest salary of csit department”

Expected Result:

* SELECT max(salary)
* FROM employee JOIN department ON employee.dep\_id=department.dep\_id
* WHERE department like '%csit%'

Actual Result:

* SELECT max(salary)
* FROM employee JOIN department ON employee.dep\_id=department.dep\_id
* WHERE department like '%csit%'

Analysis:

As the above sentence is syntactically sound and possess full semantic. As it consists of words like is, of which are used by our parser to draw dependencies among the words in the sentences and with the help of dependencies it is able to do semantic analysis to finally get the expected query.

**Case 2: Question with more noise in select statement**

Question: “list the names, salary of employees”

Expected Result:

* SELECT name , salary FROM employee

Actual Result:

* SELECT employee
* FROM employee

Analysis:

As our system is able to query the database up to two fields. But the parse has some constraints that the two fields within the query must have syntactic structure that the parser understand. So in the above case the parser get confused so it only selects the employees, but if the natural language was “list the names and salary of employees” the parser would give the expected result.

**Case 3: Question with more noise in where statement**

Question: “list the employee with salary greater then 1000 and employee ram”

Expected Result:

* SELECT employee
* FROM employee
* WHERE salary > 1000 and employee like ‘%ram%’

Actual Result:

* SELECT employee
* FROM employee

Analysis:

The NLIDB that we have developed supports only one condition in where statement. Whenever there are more than one condition detected by the parser can’t parse the NL question to get the expected result, so the expected result is not obtained. If the question was “list employee with salary greater than 1000” then the parser would parse the question to get the expected result.

**Case 4: Question with much noise.**

Question: “list highest salary cist department”

Expected Result:

Should throw exception or the parse should not parse the sentence.

Actual Result:

Gives exception message as too much noise in the sentence and output query is :

* FROM employee
* WHERE salary =cist

Analysis:

As the sentence look like it has semantic but no syntax analysis can be done as it lacks the words like is, of which is extensively used by our parser to draw the semantic by analyzing the syntactic nature with help of these words.

## 9.3 Some more examples

Some more examples are listed as below. The question is asked to the parser and the query resulted by the parser is listed as below:

|  |  |
| --- | --- |
| Question in Natural Language | Output Equivalent Answer By the parser |
| * list employee with salary greater then 2000 | SELECT employee FROM employee WHERE salary > 2000 |
| * what is the employee with salary 5000 | SELECT employee FROM employee WHERE salary =5000 |
| * what is the name and salary of employee ram | SELECT employee , salary  FROM employee  WHERE employee like '%ram%' |
| * what is the location of employee ravi | SELECT city  FROM employee JOIN location ON employee.loc\_id=location.loc\_id  WHERE employee like '%ravi%' |
| * list the employee of csit department | SELECT employee  FROM employee JOIN department ON employee.dep\_id=department.dep\_id  WHERE department like '%csit%' |

## 9.4 Complete Trace of Interface

We have analyzed the complete workflow of the parser from the start to the end of the query generation in console. Therefore here we are describing the complete work flow of mapping the natural language to respective SQL query. We start with the NL question:

Question: “list the names of the employees”

The complete Trace:

STEP 1:

The first step the natural language is preprocessed and passed to the stanford parser and the stanford parser parses gives list as:

root(ROOT-0, list-1)

dep(list-1, names-3)

prep\_of(names-3, employees-6)

STEP 2:

The list is again processed by the helper class and remove unwanted dependencies if exits like det, num etc. As there is no unwanted dependencies it will give the same list as by the dependency parser:

root(ROOT-0, list-1)

dep(list-1, names-3)

prep\_of(names-3, employees-6)

STEP 3:

Now the filtered list of dependencies is processed by the dividelist parser which divides the list in to possible where and select groups.

Possible SELECT clause elements: names

Possible WHERE clause elements: list,employees

STEP 4:

Finally the possible select and where group is send to the query generator parser and the parser generates the final query by relying on the data dictionary and own data dictionary. As the select clause has names so it’s respective field name is drawn by querying the data dictionary and the table is appended to from clause by issuing following query.

SELECT scolumn,stable FROM synonym WHERE syname like '%names%'

Now the query generator deals with the where clause so it divides the list into right and left possible elements. As it found that left has only possible statement i.e. ‘employee’ and the right list is removed by the parser and founds the right to be null so finally parser decides that there is nowhere statements for this NL question by issuing the following queries:

SELECT scolumn,stable FROM synonym WHERE syname like '%list%'

SELECT scolumn,stable FROM synonym WHERE syname like '%employees%'

In last the parser generates the query as:

SELECT employee

FROM employee

**Chapter – 10**

# Conclusion

User interfaces are a very important part of Computer Science. They define the way computer programs are perceived by users, and contribute to the overall worth of the program. NLP can bring powerful enhancements to virtually any computer program interface, because language is so natural and easy to use for humans. The NLIDB is no exception. Alternatives for integrating a database NLP component into the NLIDB were considered and assessed. The goal of this project was twofold. The first goal was to create a NLI for database and integrate it. The second goal was to consider the possibility of developing an authoring tool for creating database NLP systems to reduce the complexity in the SQL. Both goals were investigated with respect to the requirements of NLIDB.

This system helps user to easily retrieve data from database using simple English language. This system works fine with JOIN condition. This system also responds to complex JOIN queries. We can add more synonyms for column names and table names so that system is able to handle more queries.

As our focus was for the generative parser that is able to map the NL sentences into equivalent SQL queries rather than the machine learning approach. In addition the machine learning approach is considered to be expensive in both the cost and computational field. Along with we showed that we can use the syntactic semantics to get the semantics of the words existed in the NL. We had done this by using the SDP to get the dependencies, then use this dependencies to project to two possible lists ie for projection and selection. Finally with the help of the data dictionary and the IS we generate the final query in SQL.

While there are still issues to be resolved for a complete integration of an NLP system with DB’s, these results show that it is a possibility. Further NLP enhancements could extend the proposed initial database NLP proposal.

## 10.1 Future Work

Up to now our interface is able to select up to 3 parameters in select clause, one condition in where clause and JOIN up to two tables. We can extend the domain of our interface for being able to answer the complex NL question. We have addressed that we can use a hybrid approach for mapping NL question into equivalent SQL query by using supervised and unsupervised machine learning approach.

Moreover in future we will use Support Vector Machine and Tree Kernels to train the parser for the selection of possible statements for select and where statements. Thus make able to answer the complex NL questions which possibly consists of nesting SQL queries.

However our main focus will be developing the NLI in our own native language i.e. Nepali.

# Appendix

This section gives the over view of the main codes and programmatic logics implemented to produce the interface. The project mainly consists of five classes which are listed as below.

1. DependencyParser
2. HelperClass
3. DivideList
4. QueryGenerator
5. SqlLibrary







****

****

****

****

****

****

****

****

****

****

****

****

****

****

****

****

****

****

****

****

****

****

****

****

****

# References

* [www.wikipedia.org](http://www.wikipedia.org)
* <http://nlp.stanford.edu/downloads/corenlp.shtml>
* <http://stackoverflow.com/>
* <http://docs.oracle.com/java/tutorial/>
* <http://www.coderanch.com/t/271631/java/>
* <http://chaoticity.com/>
* <http://www.mysql.com/>
* <http://www.towson.edu/ows/ptsspch.htm>
* <http://en.wikipedia.org/wiki/Question_answering>
* <http://cogitolearning.co.uk/?p=525>
* <http://www.wikihow.com/Avoid-Colloquial-(Informal)-Writing>
* <http://www.dailywritingtips.com/can-you-start-sentences-with-%E2%80%9Cand%E2%80%9D-and-%E2%80%9Cbut%E2%80%9D/>
* <http://semanticsql.com/>
* <http://office.microsoft.com/en-001/word-help/start-page-numbering-later-in-your-document-HA102841427.aspx>
* http://en.wikipedia.org/wiki/Dependency\_grammar
* http://english2sql.com/index.html
* http://sharpnlp.codeplex.com

# Bibliography

1. Alessandra Giordani “Structural Mapping between Natural Language Questions and SQL Queries” DISI - University of Trento, October 2012
2. Alessandra Giordani and Alessandro Moschitti “Semantic Mapping Between Natural Language Questions and SQL Queries via Syntactic Pairing” , Via Sommarive 14, 38100 POVO (TN) – Italy, 2009
3. Alessandra Giordani and Alessandro Moschitti “Translating Questions to SQL Queries with Generative Parsers Discriminatively Reranked” Computer Science and Engineering Department University of Trento, Povo (TN), Italy
4. Androutsopoulos, Ritchie and Thanish, “Natural language interfaces to databases-an introduction”, Journal of Natural Language Engineering. v1 i1. pp. 29-81 1995
5. Collins, M., & Duffy, N. (2002). New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL-2002)*, pp. 263–270, Philadelphia, PA.
6. Copestake, A. and Sparck-Jones, K. Natural language interfaces to databases. Natural language interfaces to databases. The Knowledge Engineering Review, Special Issue on the Application of Natural Language Processing Techniques, 1990.
7. Marie-Catherine de Marnee, Bill MacCartney and Manning, Christopher D. Generating typed dependency parses from phrase structure parses. In Proceedings LREC 2006, 2006.
8. Ruwanpura, S. Sq-hal: Natural language to sql translator. URL <http://www.csse.monash.edu.au/> hons/projects/2000/Supun.Ruwanpura.
9. Seymour Knowles “A Natural Language Database Interface For SQL-Tutor”, Journal of Natural Language Engineering. v1 i1. pp. 56-20 1999
10. Sacerdoti, Hendrix, G.G., E.D., Sagalowicz, D., Slocum, J. “Developing a natural language interface to complex data”, in *ACM Transactions on database systems,* 3(2), pp. 105-147, 1978
11. Winnograd, T. *Understanding natural language.* Edinburgh University Press, 1972.
12. Warren,Akama, S. (Ed.) *Logic, language and computation,* Kulwer Academic publishers, pp. 7-11, 1997.
13. Zhashi, R., Boohtra. Natural language interfaces: Comparing english language front end and english query. Virginia Commonwealth University, 2004. Thesis (M.S.).
14. Zangi Huangi,Guiang et al. “A Natural Language database Interface based on probabilistic context free grammar”, IEEE International workshop on Semantic Computing and Systems 2008