National College of Ireland

Natural Language Search Over Database

MSc Research Project
Data Analytics

Aritra Dattagupta
x15028992

School of Computing
National College of Ireland

Supervisor: Vikas Sahni
Industry mentor: Damian Zapart

National College of Ireland
Project Submission Sheet – 2015/2016
School of Computing

Student Name:	Aritra Dattagupta
Student ID:	x15028992
Programme:	Data Analytics
Year:	2016
Module:	MSc Research Project
Lecturer:	Vikas Sahni
Submission Due Date:	12/12/2016
Project Title:	Natural Language Search Over Database
Word Count:	4828

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.

Signature:	
Date:	21st December 2016

PLEASE READ THE FOLLOWING INSTRUCTIONS:

1. Please attach a completed copy of this sheet to each project (including multiple copies).
2. **You must ensure that you retain a HARD COPY of ALL projects**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. Please do not bind projects or place in covers unless specifically requested.
3. Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Natural Language Search Over Database

Aritra Dattagupta
x15028992
MSc Research Project in Data Analytics

21st December 2016

Abstract

Natural language search from a database makes it possible for non technical users to fetch information from database in natural language. In natural language parsing getting highly accurate syntactic analysis is highly important. Parsing of natural language means mapping natural language components to its syntactic representation. This system proposes a model to implement this natural language parsing using NLTK chunking with regular expression called "grammar" and named entity recognition to extract keywords and phrases from the parse tree. The extracted keywords are then mapped to different attributes and commands of a SQL query using an algorithm. The algorithm uses rules to map different commands and attributes of the database. When the algorithm finishes forming the SQL query it executes it into the database and fetches the result to display the information to the user.

1 Introduction

The main purpose of natural language interface to a database of any form is interpretation of natural language into actions or queries understood by a machine. Natural language query processing is a critical endeavour in the field of natural language processing and despite the difficulties it has potential of highly intelligent systems capable of interacting with a user in natural language form. The applications that will be possible when NLP capabilities are fully realised, will be able to parse natural language into queries, extract information from different data sources, translate in real-time and interact with the user in natural language.

Asking questions in natural language form to a database is a very convenient and easy method of accessing data for a causal user who does not have any knowledge of database queries. Natural language search from a database is hot topic for researchers since late sixties Androutsopoulos et al. (1995). The purpose of a natural language interface is to understand any natural language and convert them to appropriate database queries. A complete natural language system will benefit in many ways. It can be used by anyone to gather information from the database. It may change our perception of the information stored in the database. In mobile systems, users can simply ask a question without the hassle of diving into the database and fetching results. This system focuses on building a natural language interface for a customer who can access his bank details without having to use any query language or navigate through applications for the right information.

The user would simply ask the application to provide details in natural language and the application will translate that NLP query to a database query, in this case MySQL, and then retrieve the information from the database and present it to the user. This system is heavily dependent on NLP libraries and techniques which makes it possible to parse the natural language into a database query.

Unlike Nihalani et al. (2011), Kaur and Bali (2012), Chaudhari (2013) which syntactic analysis and pattern matching systems, this system uses semantic analysis and NER to extract keywords and required commands to form the structured query. The accepted user input first gets tokenized, then parsed using a grammar. It also undergoes entity recognition to identify keywords required by the system to form a structured query. Finally the system traverses this tree, to extract keywords by shallow parsing also called chunking. It then combines all these extracted commands and objects to generate the final query which is then executed to fetch results from the database.

2 Related Work

2.1 History of natural language interface

The LUNAR system Woods et al. (1972) is a question answering system about rocks brought back from the moon. This system has two databases: one for the literature references and one for the chemical analysis. The LUNAR system uses Woods procedural semantics along with Augmented Transition Network(ATN) parser. The LADDER system Hendrix et al. (1978) concentrates on creating queries from English language and was developed as an aid to Navy decision makers. The LADDER system has three components. First component is called INLAND which accepts user input in English language with a restricted subset of vocabulary with respect to the database. It then translates this query into a subsequent query. The second component is called IDA (Intelligent Data Access) which accepts the query from INLAND to identify the field values from the database. INLAND provides lower level syntactic units to the IDA to from a higher level syntactic meaning of the English sentence and then finally generate a formal DBMS query. The third component FAM (File Access Manager) manages the mapping of the location of generic files located throughout the database. At the time of first release, LADDER system could process a database of 14 tables and more than 100 attributes.

PLANES system Waltz (1978) was created to provide access to aircraft data and its maintenance by using English language to query the database. The whole operation of plains can be divided into four parts: parsing, query generation, evaluation and response. The parsing phase matches the users request against a set of phrase patterns stored in ATN networks and semantic sentence patterns. The second phase translates the semantic constituents of the users request into a formal query. The third phrase uses this query to fetch information from the database. The last phase evaluates the result and displays it to the user. This system was developed in 1977 and was known as Philips Question Answering system Scha (1977).

Chat-80 Warren (1981) designed to be both precise and adaptable to variety of applications. This system was developed in Prolog, a programming language based on logic. It uses logic-based grammar formalism known as extra-position grammars to translate English language into Prolog subset of logic. This logical expression is then passed on to a planning algorithm to form a relational database query. The domain of this system is geography and it can process complex queries well within one second. According to

Amble (2000), the CHAT-80 system was sophisticated and a state of the art system in the early artificial intelligence era. Amble (2000) developed a system called BusTUC which was impressive in its own merits.

TEAM (Transportable English Access Data manager) system Grosz (1983),Grosz et al. (1987) was designed to construct an adaptable or transportable natural language system. This system can adapt itself to new systems for which it was not hand tailored. This system relies on an interactive dialogue system which provides required information to adapt to new applications. The system has two phases: first phase known as DIALOGIC constructs a logical form of the query from the user input and the second phase queries the database. But these steps require informations of the database or domain which is stored in a separate system to provide for transportability.

2.2 Recent work

NALIX Li et al. (2005) is a system that accepts arbitrary English language as input which may include aggregation, nesting ,etc. This query is reformulated to a corresponding XQuery expression which can be used against a XML database. Schema-Free XQuery is a query language designed to retrieve data from XML databases. NALIX is a syntax based system. It has three steps to formulate XQuery by processing English language. Firstly, a parse tree is generated then it is validated and then the parse tree is translated to XQuery. A typical user interaction in NALIX involves query translation and query formation. The precision of NALIX Popescu et al. (2004) is 83% on average, with an average precision of 71% for the worst query; for 2 out of each 9 tasks, NALIX achieves perfect recall.

PRECISE Nihalani et al. (n.d.) system uses a statistical parser as a plug-in to train the system with new databases. The lexicon is automatically generated from the database to map values, tables, columns, etc. It tokenizes the English sentence and maps it to the lexicon table using a statistical parser to find the most statistically relevant SQL query. PRECISE achieved 94% accuracy with ATIS dataset which consists of spoken questions about air travel, their written forms and their corresponding SQL query language. Second dataset used was GEOQUERY which contains information about U.S geography. 77.5 % of queries in GEOQUERY were semantically traced by PRECISE which gives it 100% accuracy using these queries. PRECISE has its own weaknesses. The system achieves high accuracy in semantically traceable questions at the cost of recall. The system also suffers from handling nested structures as it adopts a heuristic based approach Nihalani et al. (n.d.).

Natural Language interfaces started developing the early sixties. Prototype systems mainly depended on pattern matching systems. This system works on the basis that if an input matches one of the patterns defined, it can be used to translate to a database query. These types of systems are limited to databases and to the number of complexity of the patterns. As the use of database systems continued to increase, it became an obtrusive process to define complex patterns for each system. However, these systems do not need parsing or statistical modelling, it can be easily implemented. One of the best example of this pattern matching system is ELIZA Nihalani et al. (n.d.).

As statistical modelling developed, new systems were possible to be realised using statistical methods. One of those methods include syntax based systems which accepts the users question and analyses it syntactically. The parse tree generated from the syntactic analysis is directly mapped to an expression for a formal database query language. These

systems are created using a grammar that describes the structure of a users question. It is generally difficult to devise such systems which can directly parse a tree and match them with some real-life database query language. A semantic grammar system is like syntax based systems except it modifies the parse tree as much as possible by removing unnecessary nodes or combining them. The formal database query is then extracted from this parse tree. The main-drawback of these types of systems is that it requires intensive knowledge of the domain, therefore making it difficult to adapt to other systems [13]. Most recent NLDBs uses a method called Intermediate Representation Language. It first transforms the natural language input a higher level logical query irrespective of the database, expressed in some intermediate query format. It then combines these intermediate queries using a rule to form a database query like SQL.

Using the intermediate language representation methodology, Sangeeth and Rejimoan (2015) developed an intelligent information extraction system from database. They proposed a system which will take English language input and then tokenize, tag parts of speech and perform parsing using Hidden Markov Model. This linguistic modules output will be forwarded to a database module which will generate a SQL query or predict one if exact keywords are not available. The system could achieve 100% accuracy when keywords were available.

3 Methodology

3.1 Components of the system

The system proposed here is for banking domain which relies on customer's banking data. The whole system is divided into two parts: linguistic and database component.

Linguistic component

This component is responsible for accepting the user query in natural language form and perform natural language techniques to convert it into a structured query. This component heavily relies on natural language processing techniques which includes tokenization, part-of-speech tagging and semantic parsing.

Tokenization is a form of morphological analysis where a sentence is broken into words or tokens which can further be used for other forms of text analysis like parsing or role labelling. Tokenization occurs at the word level. Conventionally, white spaces and punctuations are removed before breaking it into tokens.

Syntactic analysis is a form of parsing where a sentence in natural language is parsed conforming to the rules of a formal grammar. This process generally deals with parts-of-speech tagging of the tokens representing the meaning of the sentence.

Chunking is a technique used in information extraction to identify entities. It segments and labels multi token sequences. The tokens that we get from morphological analysis is chunked using a former grammar to identify the entities related to the context.

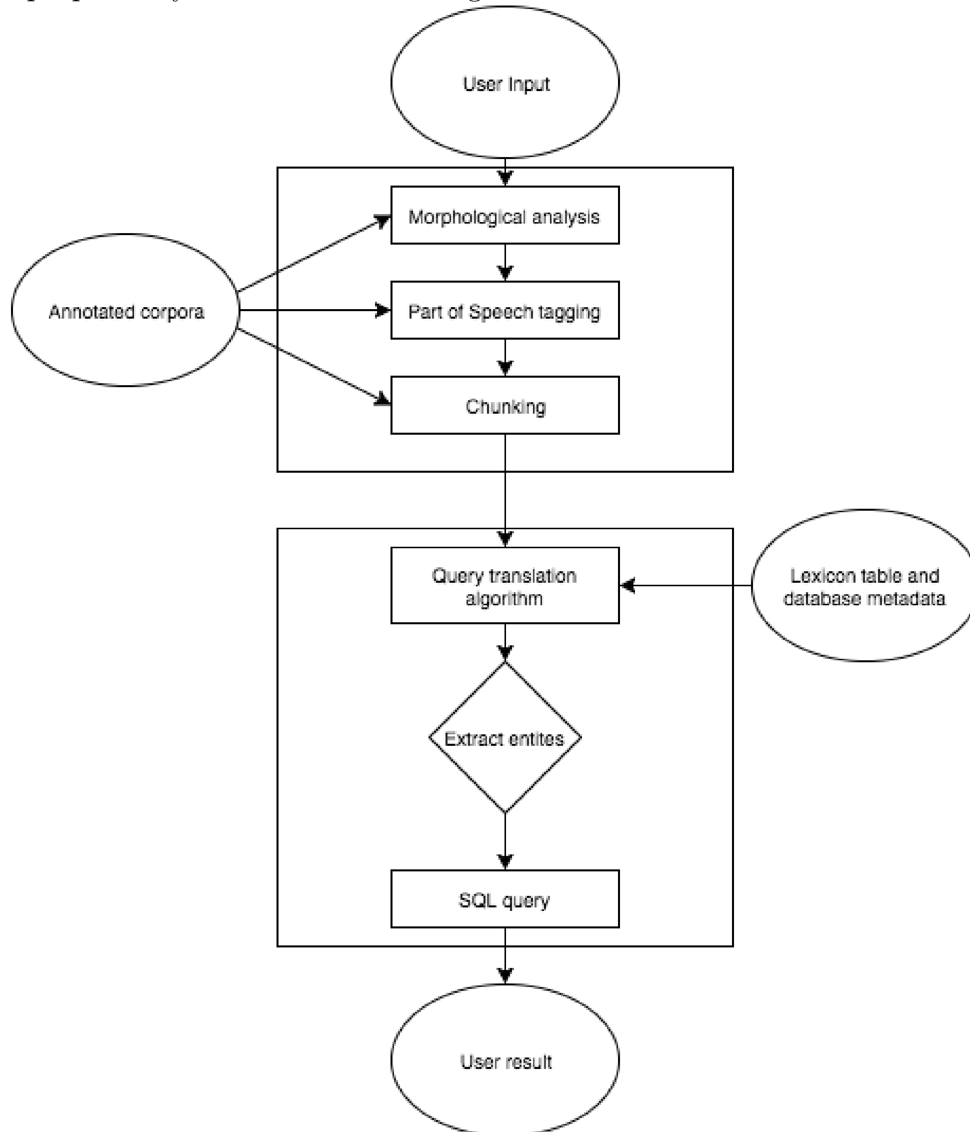
Database Component

A lexicon table is used to map the words of the user input in natural language to formal objects (tables, columns, etc.) of a database. The tokenizer and the semantic parser both uses this lexicon table. After the mapping of formal objects which is done by

the semantic parser, it generates an intermediate query. This intermediate query is then appended with other database query objects to generate a structured query language. This query is then fired into the SQL database and the extracted information is stored in an object which is then displayed to the user.

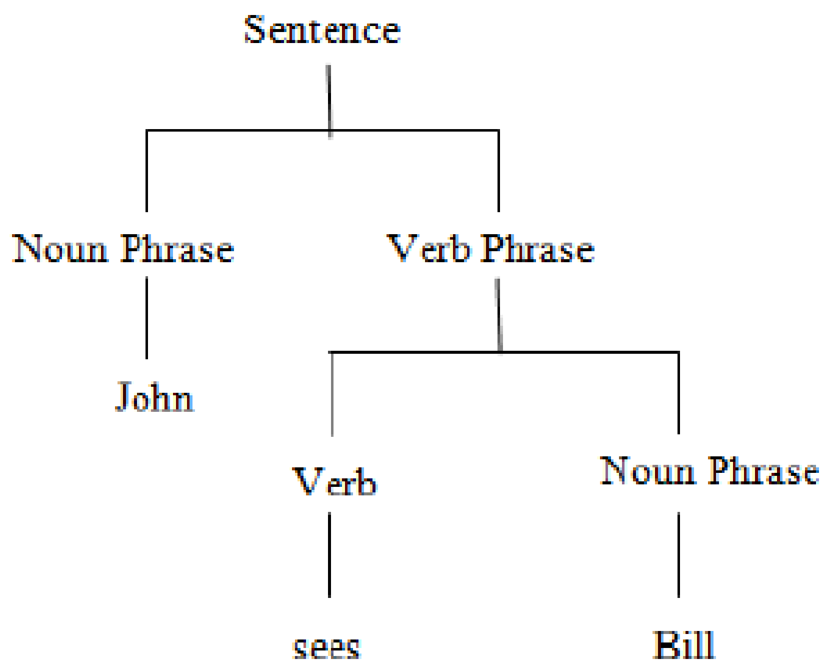
3.2 Architecture

The proposed system architecture is given below:

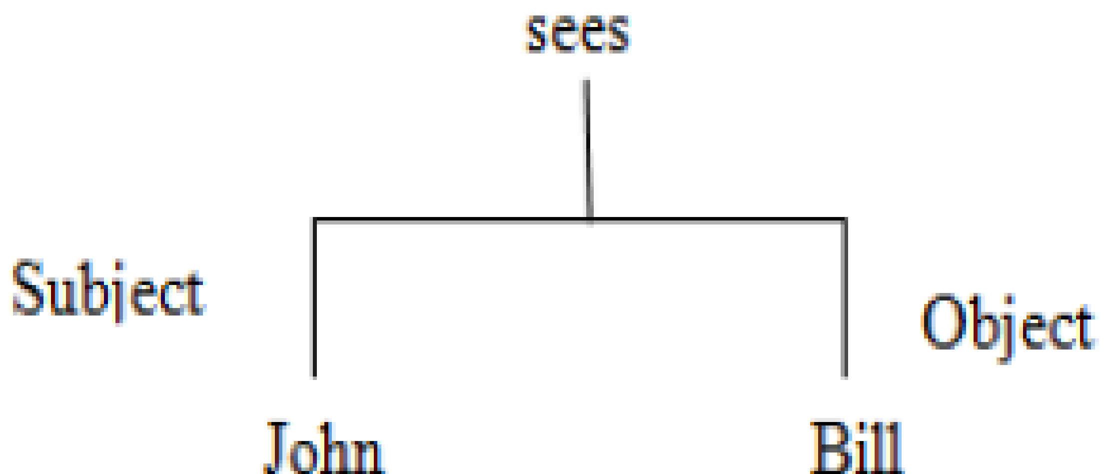


An input is taken from the user. This sentence goes under morphological analysis. This analyses and tokenizes the sentence into words or tokens. For example, if the user enters show me my account balance? the tokenizer would first remove punctuations and tokenize the words into tokens. After the tokens are generated, we parse the tree to get the part-of speech tags. This parsed tree is then chunked using regular expression. From this chunking or shallow parsing procedure, the noun phrases are extracted and mapped them to table attributes such as column name and table name. Then using named entity recognition technique, we find other entities such as date phrases needed to form a concrete SQL query. After all the entities are gathered we form the SQL query using rules defined by the system.

The system uses a form of constituency parsing to find the parse tree of the sentence. Stanford parts-of-speech tagger is used to get the tags and then a parse tree is created. Unlike Kokare and Wanjale (2015) which uses Dependency parsing to generate the parse tree, this system uses constituency parsing. In dependency parsing the tokens are related by binary asymmetric relations called dependencies which focus on relation between words. In this case, it uses constituency parsing which breaks a text into subphrases. Non-terminals are types of phrases, the terminals are the words in the sentence and the edges are unlabelled. A simple constituency tree for John sees Bill would be:



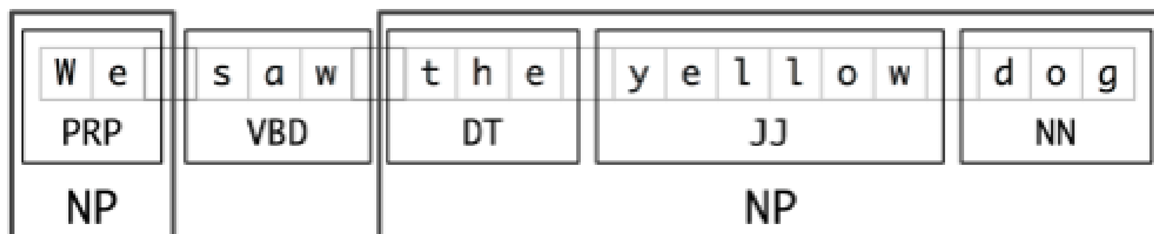
A dependency parse tree would like the figure below.



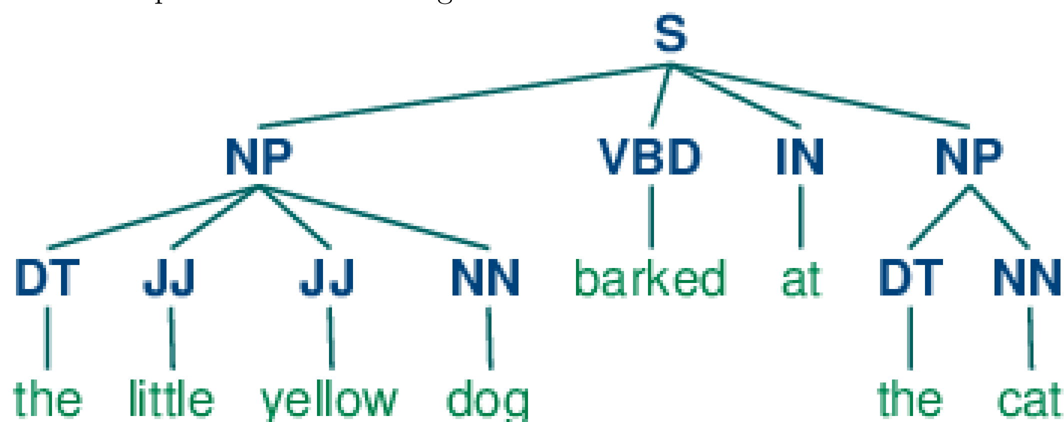
A dependency connects tokens according to their relationship in the sentence. Each vertex in a tree represents a word, child nodes are depended words of the parent, and edges are labelled by relationship.

This system modifies the tree using shallow parsing method provided by NLTK tools

using regular expression. A chunked sentence is shown below:



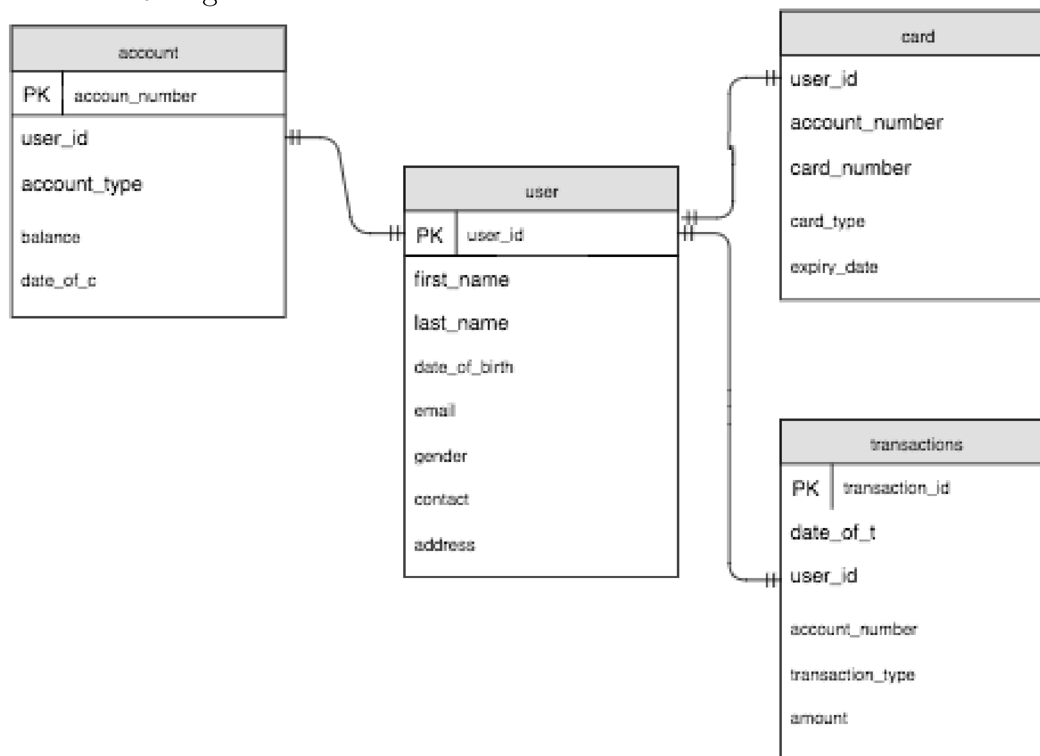
The smaller boxes show word level tokenization and part of speech tags while bigger boxes show phrases representing higher level chunking. One of the most used technique from chunking is noun-phrase extraction. In chunking, one of the most useful source of information is parts of speech tags. This is one of the motivation for part-of-speech tagging in our system for extracting information. To create chunks, we first define a grammar consisting of rules that indicate how sentences should be chunked. For the sentence above that we see in fig 4, we define a grammar using regular expression, `NP: {<DT>?<JJ>*<NN>}` to create the chunk tree. The grammar rule says that NP chunk should be formed when chunker finds an optional determiner followed by adjectives and nouns. Using this grammar, we create a chunk parser and a resulting tree is obtained which is shown below.



Next a nlp technique called named entity recognition is used to identify entities represented by complex phrases. This helps in identify date phrases which are not in numeric format. Date phrases such as "last week", "last month" should be identified by the system as date and eventually convert them into it numeric date format. This completes the linguistic component of the system. The chunked tags are given to the database module to generate a concrete SQL query. The database has a query translator algorithm based on rules which proceeds by searching table names, column names or their synonyms from the NP chunks. Column synonyms are already defined in the system using WordNet. Commands of a SQL query like AND, OR, NOT, WHICH, etc are found by searching CC, JJ and JJR tagged chunks. All these components are joined together on conditions to form a final SQL query which is the semantic representation of the users natural language input. Finally, the query is executed into the database to get the result.

4 Implementation

The need for a database system requires data which will form the base of this system. This system is built for a banking domain where customers could ask questions in plain natural language to gather information about their accounts and transactions. Data was generated from http://mockaroo.com which helps in generating any type of data in variable size. All the tables have 1000 rows and the database is named ric. Below figure shows the ER-diagram of the database.



The central table "user" has all the customers personal information like full name, date of birth, contact details, etc. The "user_id" in this table is the primary key which acts as foreign key for all the other tables. The "account" table has information regarding customers account details and their available balance. The "card" table holds information related to customers debit/credit card details. The "transactions" table has details related to transactions of every customer which has fields like amount, date of transaction, transaction type, etc. This database is going to be the source of all data required by a user to gather information from this system.

Algorithm

- Accept the input form the user in the form of text.
- Split the input into form of tokens and store it in a list.
- Find all the attributes of all tables.
- Find the table name in the query or by matching it with the synonym set of table names pre-defined by the system using WordNet.
- Find the column name in the query or by matching it with synonym set of column names.

- If there is no column name, get the primary column name of the tables.
- User part of speech tagger to generate the pos tags.
- Use a grammar defined the system in the form of regular expression to perform chunking.
- Use NER (named entity recognition) tagger to find out the entities like date, etc.
- Get conditions of WHERE, BETWEEN, AND or OR clauses from the chunking parser.
- Get date field if any from the NER tagger.
- Generate the SQL query meeting the above conditions.
- Execute the query.
- Show result to the user.

The schema of the database is automatically stored using the python scripting language in a dictionary. The keys of the dictionary are the table names and the values are their respective columns. Since it is automatically generated by reading the MySQL database it is easier for the system to adapt to new addition of tables and attributes. A lexicon table is used to store the synonyms of table names and column names and predictable keywords of the system. The synonyms are obtained using WordNet and they are stored in a list. An expression mapping table is used to map phrases to symbols required in a SQL query. The expression table is shown below.

Expression mapping	**Symbols**
greater than	$>$
lesser than	$<$
more than	$>$
less than	$<$
above	$>$
below	$<$
greater than equal to	$\geq$
less than equal to	$\leq$

As the user enters the search query in natural language it gets tokenized and all table attributes are mapped. To find out keywords and table attributes, parts of speech tagging and chunking using regular expression techniques are applied to the sentence. For a typical parts of speech tag of the sentence show me my account balance in python list looks like the one below,

```
[(u‘e’, u‘PRP’), (u‘account’, u‘NN’), (u‘balance’, u‘NN’)
, (u‘my’,u‘PRP\$’), (u‘show’, u‘VB’)]
```

The above list shows each token with its pos tag. From these tags, nouns represented by NN or NNS can be mapped to table name and column name. If there is no table name or column name found, then the query is invalid. If a table name is found the system knows which column name needs to be mapped from the schema dictionary.

Next we design regular expression for the chunk parser using part-of-speech tags. For this system the regular expression called “grammar” looks like this

NP: {<DT>?<NN.*>} CP: {<JJ.*>} CD: {<CD>} PP: {<IN>} . NP means chunk noun-phrase by combing all the determiners and nouns. CP chunks all adjectives, CD chunks all numbers and PP chunks all prepositions. We then traverse this parse tree generated by the regular expression and extract al the phrases into multiple lists. NP is used for noun-phrase extraction to map table name and column name. CP extracts all adjectives which can be mapped to the expression mapping table described above. For example, in the sentence show me all transactions greater than 100, the chunk parser will combine and extract the adjective greater into CP and than into PP. If we combine these two lists of adjective and prepositions we can get the symbols required in a SQL query. To extract numeric digits, we use the CD chunk. Given below is a parse tree generated by the script for the sentence show me all transactions greater than 100.

```
(S
  me/PRP
  all/DT
  (CP greater/JJR)
  (NP transactions/NNS)
  show/VB
  (CD 100/CD)
  (PP than/IN))
```

For queries with date, we use named entity recognition technique to identify the date phrase. For the sentence, show me all transactions from last week, the NER tagger must identify the phrase last week as date entity. As the tagger identifies the phrase, the phrase is parsed into its equivalent date format using pythons datetime library and the system knows that it must select the date column for the respective table and assign the value to it. For complex sentence like show me transactions from April to October, the NER tagger will tag the phrase April to October as date type entity. In this case, the system removes stopwords from the extracted phrase and then split them into tokens. Each of these tokens represent a date which is eventually parsed by the datetime parser to its equivalent numeric date format(yyyy-mm-dd).

After all the keyword extractions and mapping of database attributes are stored in a list, the final SQL query is built. To build this query, the system has some rules defined. It checks for the length of the keywords list to form the query. If the preposition and adjective lists are empty, it is understood that there is no expression mapping needed by the system. It can just query using the table name and column name. If the above-mentioned tables are not empty, then it maps the expression to its equivalent symbols. Since there are symbols need in the query the number list is also checked to identify the numbers. If the size of the symbol and number list is more than 1, then the system identifies that there is a need of WHERE and RANGE clause. It then generates a query by placing all the keywords in their respective positions. Similarly, if the size of the date list is more than 1, then the system knows it needs the RANGE clause for the date column in the respective table. After the query is built, it is executed and fired into the database. The result is then eventually formatted and displayed to the user.

5 Evaluation

We start the evaluation of the system by taking inputs from the user. The user needs to enter the userID and the search query in natural language. The system returns the equivalent SQL query and the results of the query below it. Each case study shown is for different types of queries which are of different complexities.

5.1 Experiment / Case Study 1

Enter User ID: 1

Search: show me my account balance

SELECT balance FROM account WHERE user_id = 1;

2442.5

The execution of the script asks the user to enter the ID and the search query. For query "show me my account balance", the system generates the equivalent SQL query and final returns the result "2442.5" which is the account balance for user ID equal to 1. This is a simple query where the system understand to select the table "account" and column "balance".

5.2 Experiment / Case Study 2

Enter User ID: 1

Search: show me my account account details

SELECT account_number, account_type, balance, date_of_c FROM account WHERE user_id = 1;

IS82 7536 0166 0220 6466 1844 savings 2442.5 2014-10-09

The query "show me my account details" is processed by the system to understand that details means selection of multiple columns which will describe the account information as stored in the database. It understands the table to select over here is "account"

and multiple columns which is smartly fetched by the system. Finally the result is displayed which contains the account number(IBAN), account type, balance and date the account was created, It does not unnecessarily fetches other columns like row number, user ID, etc.

5.3 Experiment / Case Study 3

Enter User ID: 3

Search: show me my card details

SELECT account_number, card_number, card_type, expiry_date FROM card WHERE user_id = 3;

CY26 2045 9461 N4YR ADPA WEE6 5602233638188407 bankcard 2023-12-10

AE73 4626 4357 9983 0183 402 633396694379240658 switch 2016-05-09

This time when the user ID is changed to "3" and the user asks for its card details, the system smartly selects the "card" table and meaningful columns related to card details. Finally it displays the result in multiple lines as the user has two types of card.

5.4 Experiment / Case Study 4

Enter User ID: 1

Search: show me transactions greater than 100

SELECT account_number, transaction_type, amount, date_of_t FROM transactions WHERE user_id = 1 AND amount > 100;

BG56 XPLV 2746 92UW E9IR deposit_cash 719 2011-12-05

BG56 XPLV 2746 92UW E9IR deposit_cash 314 2014-05-05

BG56 XPLV 2746 92UW E9IR deposit_cash 976 2013-07-03

BG56 XPLV 2746 92UW E9IR withdraw_cash 250 2016-12-14

The user asks for "show me transactions greater than 100". The system smartly selects the "transactions" table and column "amount". It also understands that there are multiple clauses to the WHERE statement so it uses AND operation to add more

clauses. The system knows from the expression mapping table which symbol to select and knows that amount corresponds to values more than 100. This similar execution can be done by a different query of same meaning. For example "show me transactions above 100", the system will return the same result as shown below.

Enter User ID: 1
Search: show me transactions above 100

SELECT account_number, transaction_type, amount, date_of_t FROM transactions WHERE user_id = 1 AND amount > 100;

BG56 XPLV 2746 92UW E9IR deposit_cash 719 2011-12-05

BG56 XPLV 2746 92UW E9IR deposit_cash 314 2014-05-05

BG56 XPLV 2746 92UW E9IR deposit_cash 976 2013-07-03

BG56 XPLV 2746 92UW E9IR withdraw_cash 250 2016-12-14

5.5 Experiment / Case Study 5

Enter User ID: 1
Search: show me transactions ranging from 100 to 1000

SELECT account_number, transaction_type, amount, date_of_t FROM transactions WHERE user_id = 1 AND amount>100 AND amount<1000;

BG56 XPLV 2746 92UW E9IR deposit_cash 719 2011-12-05

BG56 XPLV 2746 92UW E9IR deposit_cash 314 2014-05-05

BG56 XPLV 2746 92UW E9IR deposit_cash 976 2013-07-03

BG56 XPLV 2746 92UW E9IR withdraw_cash 250 2016-12-14

This case study shows that the system can understand ranging queries where the system has to smartly choose which query clause to use and assign proper symbols followed by cardinal numbers. The user asks "show me transactions ranging from 100 to 100" and the system produces the equivalent query and output as shown in the above fig 12. This

same output can be obtained using a different natural language search as shown in the fig 13 below.

```
Enter User ID: 1
Search: show me transactions where amount is more than 100 and less than 1000

SELECT account_number, transaction_type, amount, date_of_t FROM transactions WHERE user_id
= 1 AND amount>100 AND amount<1000;transactions WHERE user_id = 1 AND amount>1 AND a
mount<1;

BG56 XPLV 2746 92UW E9IR  deposit_cash 719 2011-12-05

BG56 XPLV 2746 92UW E9IR deposit_cash 314 2014-05-05

BG56 XPLV 2746 92UW E9IR deposit_cash 976 2013-07-03

BG56 XPLV 2746 92UW E9IR withdraw_cash 250 2016-12-14
```

5.6 Experiment / Case Study 6

The system faces challenging translations when dealing with non-numeric date phrases. It means if the user inputs any date query in natural language such as "last week" or "last month", the system should be able to identify this entity as date phrase and simultaneously convert it into its numeric date format (yyyy-mm-dd). One such demonstration is given below in fig 14.

```
Enter User ID: 1
Search: show me all transactions from last week

SELECT * FROM transactions WHERE user_id = 1 AND date_of_t >= str_to_date("2016-12-13", "%Y
-%m-%d");

1003 2016-12-14 1 BG56 XPLV 2746 92UW E9IR withdraw_cash 250
```

The algorithm can also translate two date phrases at once and separate them to form a date range query. If the user asks "show me transactions from January to December", the system translates both the dates into its numeric date format and forms a date range query using the BETWEEN clause of MySQL. This is shown in the below fig.

```
Enter User ID: 1
Search: show me all transactions from January to December


SELECT * FROM transactions WHERE user_id = 1 AND date_of_t BETWEEN str_to_date("2016-01-2
0", "%Y-%m-%d") AND str_to_date("2016-12-20", "%Y-%m-%d");


1003 2016-12-14 1 BG56 XPLV 2746 92UW E9IR withdraw_cash 250
```

5.7 Discussion

The system is completely developed using python and its libraries. It uses NLTK and Stanford parser for pos tagging and parsing and chunking. For named entity recognition, it uses spaCy package which is an industrial strength natural language processing package in python. As seen from the above figures the system has a 100% accuracy in translating natural language query to its equivalent SQL query format as long as the query is related to the database in context. The system is also adaptable to the addition of new tables into the database. The administrator just has to define the keywords related to the newly added table name and column names in the dictionary pre-defined in the system. The system will then automatically fetch all synonyms from WordNet synsets. Lastly, the administrator must define the logic for selection criteria of the new table in the algorithm defined previously and the user can go on to search from this new table using natural language.

6 Conclusion and Future Work

This system is developed in Python programming language and its libraries. A MySQL database was used as the data source. The Data Dictionary must be updated when new additions are made to the database. This system can handle questions of any form related to the domain, in this case banking, where the user can access his/her information. In future, as data storage becomes greedy, a data pipeline can be created to bulk load the data from MySQL database to Hive tables so that MapReduce framework of the Hadoop ecosystem can be leveraged to access bulk data. Furthermore, a data lake can be created using distributed computing to store all bulk data of any form including structured, semi-structured and unstructured data.

Acknowledgements

I would like to express my sincere gratitude to my supervisors for the continuous support of my thesis and related research, for his patience, motivation, and immense knowledge.

References

Amble, T. (2000). Bustuc: a natural language bus route oracle, *Proceedings of the sixth conference on Applied natural language processing*, Association for Computational Linguistics, pp. 1–6.

Androutsopoulos, I., Ritchie, G. D. and Thanisch, P. (1995). Natural language interfaces to databases–an introduction, *Natural language engineering* **1**(01): 29–81.

Chaudhari, P. P. (2013). Natural language statement to sql query translator, *International Journal of Computer Applications* **82**(5).

Grosz, B. J. (1983). Team: a transportable natural-language interface system, *Proceedings of the first conference on Applied natural language processing*, Association for Computational Linguistics, pp. 39–45.

Grosz, B. J., Appelt, D. E., Martin, P. A. and Pereira, F. C. (1987). Team: an experiment in the design of transportable natural-language interfaces, *Artificial Intelligence* **32**(2): 173–243.

Hendrix, G. G., Sacerdoti, E. D., Sagalowicz, D. and Slocum, J. (1978). Developing a natural language interface to complex data, *ACM Transactions on Database Systems (TODS)* **3**(2): 105–147.

Kaur, S. and Bali, R. S. (2012). Sql generation and execution from natural language processing, *International Journal of Computing & Business Research ISSN (Online)* pp. 2229–6166.

Kokare, R. and Wanjale, K. (2015). A natural language query builder interface for structured databases using dependency parsing, *IJMSC-International Journal of Mathematical Sciences and Computing (IJMSC)* **1**(4): 11.

Li, Y., Yang, H. and Jagadish, H. (2005). Nalix: an interactive natural language interface for querying xml, *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, ACM, pp. 900–902.

Nihalani, M. N., Silakari, D. S. and Motwani, D. M. (n.d.). Natural language interface for database: A brief review.

Nihalani, N., Motwani, M. and Silaka, S. (2011). Natural language interface to database using semantic matching, *International Journal of Computer Applications* **31**(11): 0975–8887.

Popescu, A.-M., Armanasu, A., Etzioni, O., Ko, D. and Yates, A. (2004). Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability, *Proceedings of the 20th international conference on Computational Linguistics*, Association for Computational Linguistics, p. 141.

Sangeeth, N. and Rejimoan, R. (2015). An intelligent system for information extraction from relational database using hmm, *Soft Computing Techniques and Implementations (ICSCTI), 2015 International Conference on*, IEEE, pp. 14–17.

Scha, R. J. (1977). Philips question-answering system phliqa1, *ACM SIGART Bulletin* (61): 26–27.

Waltz, D. L. (1978). An english language question answering system for a large relational database, *Communications of the ACM* **21**(7): 526–539.

Warren, D. H. (1981). Efficient processing of interactive relational data base queries expressed in logic, *Proceedings of the seventh international conference on Very Large Data Bases-Volume 7*, VLDB Endowment, pp. 272–281.

Woods, W. A., Kaplan, R. M. and Nash-Webber, B. (1972). *The lunar sciences natural language information system: Final report*, Bolt, Beranek and Newman, Incorporated.