NLIDB Converter for Customer Relationship Index

G. O. Wijayasekara

149236 L

Dissertation submitted to the Faculty of Information Technology, University of Moratuwa, Sri Lanka for the partial fulfillment of the requirements of the Degree of Master of Science in Information Technology.

May 2017

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institution of tertiary education. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

Gayanee Wijayasekara	……
Name of Student	Signature of Student
	……
	Date
S.C. Premaratne	……
Name of Supervisor	Signature of Supervisor
	.....
	Date

Acknowledgement

I take this opportunity to express my heartfelt gratitude to my supervisor, Mr. S. C. Premaratne, Senior Lecturer at the University of Moratuwa, Sri Lanka, for guiding me throughout the research project without whose support and direction this would not have been possible.

And a special heartfelt thanks goes out to my loving family for being the pillars of strength that they are at all times.

I would also like to thank my colleagues for the valuable feedback and support they extended during the course of the research.

Abstract

Information plays a significant role in our day to day life. Information affects how we look at things and make decisions. Therefore, accurate information must be readily available. With technology reaching new heights, in modern day systems and applications, the main source for information storage is an underlying database system. Retrieving information from a database management system requires a specific expert skill set which predominantly includes knowledge on Structured Query Language and domain specific knowledge. In recent times there is a rising demand for non-expert users to be able to directly extract information from an underlying database management system. For industry specific Customer Relationship Management applications like the Customer Relationship Index used in large organizations like the Brandix Group this is seen as both a challenge and an opportunity to be explored. Implementation of a Natural Language Interface to the underlying database of the Customer Relationship Index allows a typical user to retrieve the required information from the underlying database using natural language like English without prior programming or technical knowledge. For above reasons the NLIDB converter for the Customer Relationship Index is introduced. This application takes the users query to be run against the underlying database in natural language and returns the corresponding T-SQL statement which can then be run against the database to extract results. The NLIDB converter has shown that it can successfully and efficiently convert questions in natural language to the corresponding T-SQL statement with an accuracy rate of more than 80% for two types of output T-SQL statement formats for the Customer Relationship Index database.

Contents

List of Figures

List of Tables

Chapter 1

1 Introduction

1.1 Prolegomena

With the evolution of information technology reaching new heights in the recent decades, databases are gaining prime importance in a huge variety of application areas employing private and public information systems. Databases play a critical role and are built with the objective of facilitating the activities of data storage, processing, and retrieval associated with data management in information systems. Due to the progress made in computer technologies, software engineering and system development in the recent years, databases have become the repositories of huge volumes of data [1]. These databases could be either object oriented or relational, but irrespective of how they are storing the data, all the databases contain a collection of related information stored in a systematic way modelling a part of the real world [2].

In order to retrieve information from a database one needs to learn a querying language that the database management system is able to understand regardless of the database management system being a relational database management system or an object oriented database management system.

Structured Query Language or SQL is a form that is widely used for submitting commands to the database management system in order to produce the desired results. However, writing SQL commands known as SQL queries is not a skill that all users poses. Even for a person coming from an IT or computer systems background unless he or she has learned SQL terminology specifically, they will not be able to retrieve the information they require from a database.

As the information systems are intertwined with all aspects of our daily lives, it's only natural for the people to want to do more with these systems around them without going through the trouble of learning programming or querying languages.

Therefore, in recent times, there is a rising demand from non-expert users to query relational databases in a more natural language, encompassing linguistic variables and terms, instead of operating with SQL terminology. Thus, the idea of using natural language instead of SQL has prompted the development of a new type of processing method called natural language interface to database systems (NLIDB) [1]. To elaborate more, a natural language interface to a database (NLIDB) is a system that allows the user to access information stored in a database by typing requests expressed in some natural language. e.g.: English.

Usage of NLIDB has the following benefits of making information available to a large number of people because more people can now utilize the database management system, improving the decision making process as information is readily available to be extracted by the stakeholders and to be able to interrelate information from different sources as information is more easily absorbed and utilized by the average user [1].

However, mapping a natural language in to the programming language semantics have always been a major interest and a challenge. Therefore, natural language interfaces are a hot area of research since long. Although the earliest research has started since the late sixties [3], it is still considered a major challenge and an open research problem only increasing the appeal of natural language interfaces to databases [4].

1.2 Background and Motivation

Computerized relational databases are used to form information systems which model real world issues and are composed of objects, the relationships i.e., facts, between those objects and the constraints and rules which govern these relationships and objects. Objects are physical or logical entities, capable of being uniquely identified. In this respect, objects are said to be essentially noun-like. Facts define the manner in which objects interact with one another, and are essentially verbs or are verb-like. Constraints modify or constrain the inter-relationships between objects and facts, and as such are analogous to adverbs and pronouns. As the use of information systems increases and the design of such systems advance, so increases the complexity of the real world issues they are expected to accurately model [5].

In creating an information system, a user needs to accurately transform the real world model, also known as the external view of the data, to its actual physical implementation, using a particular database language on a particular computer system. This implementation is also called the physical view. In order to realize the power inherent in relational databases, it must be made possible for someone with no computing background or education to be able to query meaningful data from them without having to learn a specific computer language.

The physical view of an information system is expressed in one of a number of database design languages. Examples of database design languages well known to those skilled in the art include Structured Query Language (SQL) and Microsoft Access. These database design languages are well adapted to carry out the storage and subsequent retrieval of data stored therein, but the languages themselves are both unnatural and highly technology specific. This means that database design languages are not typically used or understood by the end users of the information systems the languages model. The use of these design languages is a largely intuitive process practiced by database analysts who are familiar with the internal complexities of such languages.

Natural language interfaces to computer systems that have been constructed to employ a grammar which characterizes the set of acceptable input strings. A parser then accesses this grammar to produce a parse tree (parse trees for ambiguous input) for the input string. The parse tree is then translated into an expression (or set of expressions) which represent the meaning of the input string and which are interpretable by the target computer system for which the interface is being built. A wide variety of grammar formalisms and parsing strategies have been developed.

For all natural language systems, the user is required to type his question using the keyboard of a computer terminal. When the entire query has been received, the natural language interface processes the input. Processing typically results in either an answer to the user's query or a response indicating that the query was not well-understood.

Most natural language interfaces to database systems have been prototypes, built by the research community. The primary application for natural language interfaces has been to

natural language database query systems. Some of these prototype systems include the LUNAR system [6], the PLANES system [7], the LADDER system [8], and the TQA system [3]. Some commercial systems exist. Larry Harris of Artificial Intelligence Corp, Roger Schank of Cognitive Systems Inc., and Gary Hendrix of Symantec are all marketing natural language interfaces for software systems.

A complete NLIDB system will benefit a user in many ways. The most significant benefit is that anyone may be able to gather the information he or she wants from a database without having to specifically learn a database programming language. Additionally, it may change the perception about the information in a database. Traditionally, people are used to working with a form; their expectations depend heavily on the capabilities of the form. NLIDB makes the entire approach more flexible, therefore will maximize the use of a database.

Similarly, if an interface of such capabilities is designed for the Customer Relationship Index it would be beneficial for all the users of the Brandix Group in terms of querying and retrieving data.

1.3 Problem in Brief

Managing fruitful relationships with the customers of the Brandix group has been of utmost importance to the senior management team of Brandix. This is in par with the Brandix vision statement hence it is an ever prevailing requirement of the steering committee of the Brandix Group which consist of the board of directors and the group CEOs.

Essentially, managing fruitful business relationship with the customer contacts require defining who is responsible for communicating with whom in the customer brand as Brandix collaborates with a vast number of multimillion dollar customers and having good customer relations with each of the brand contacts is therefore curial. It is also necessary to define precedence in a customer relationship to so that there is always another person to take care of the customer relationship in case the primarily responsible party moves out of the relationship due to strategy changes.

In order to maintain visibility across all 5 clusters of the Brandix group and to see how well these customer relationships are being maintained, the steering committee suggested to have a software application that would hold and provide them with the necessary information as and when needed of the customer relationships that the group maintains. This is how the Customer Relationship Index came in to being. A system to fulfil the information needs of the top management of the Brandix group.

An interaction is known as a customer relationship in the system, and each customer relationship contains the following five elements.

1. The brand contact – The contact person from the customer brand.
2. Two people known as the primary and secondary contact from Brandix who are responsible for maintaining and managing the interaction with the contact person from the customer brand.
3. Frequency of communication for the interaction.
4. Mode of contact for the interaction.
5. The quality of the interaction known as the CRI strength.

The Customer Relationship Index provides means to define these interactions and record them. Each interaction, once defined, gets tasks assigned to it based on the frequency and the mode of contact. The primary and the secondary contact needs to complete these tasks and update the system. The system facilitates updating / recording feedback on the tasks for later reference or sharing notes with each other if they are of interest to other associates who are not assigned in that specific customer relationship.

This system and the information in it is mostly used by the CEOs and the general managers of Brandix and the information held in the system holds substantial value to them in terms of strategic decisions. Periodic reviews are done every month by the group CEO to see that the customer brand interactions are being maintained properly by all clusters and that they are improving in each iteration.

Depending on different requirements and the business direction that the board of directors of Brandix would like to undertake, the information extracted from the Customer

Relationship Index differs. In order to meet these specific information needs of the top management, the software team of Brandix Apparel Solution LTD – Essentials need to extract the necessary information from the Customer Relationship Index database using T-SQL and generate the appropriate reports. This is time consuming and the completion dates of each report depends on the work load of the software team of Brandix Apparel Solution LTD – Essentials as new and change requests are facilitated based on a first in first out policy unless its highly critical or urgent. Nevertheless, since the information is already there in the Customer Relationship Index database it should be readily available to the Customer Relationship Index users.

1.4 Hypothesis

Successful implementation of Natural Language Interface to Database for the Customer Relationship Index can bridge the gap between the languages used by humans and the computers, allowing the system users to retrieve the required information they need directly from the database without seeking assistance or depending on the software team of Brandix Apparel Solution LTD – Essentials.

1.5 Objectives

Considering the inconveniences mentioned under the problem statement section in this document, I would like to explore the feasibility of empowering the users of the Customer Relationship Index to extract the information they need directly from the database without having to call for assistance from the software team of Brandix Apparel Solution LTD – Essentials. The aim is to successfully be able to convert queries in natural language to T-SQL so that the database management system is able to produce accurate results for the Customer Relationship Index user.

1. To critically study and do and in depth exploration of the natural language interfaces to databases.
2. To critically study the issues faced when implementing a natural language interface to databases.
3. To identify and evaluate the technologies used for implementing a natural language interface to databases.

4. To implement a prototype of a natural language interface to the Customer Relationship Index based on research findings.
5. To evaluate the accuracy and the performance of the implemented solution.

1.6 Proposed Solution

As an innovative approach to address this type of problem the adoption of NLIDB or natural language interface to database is proposed. The proposed solution is a software interface that allows to input the questions that the users need to run against the Customer Relationship Index database in English and after the conversion takes place, would return the corresponding T-SQL statement which can extract the required information.

1.7 Structure of the Thesis

This thesis is organized as follows, Chapter 2 critically reviews the research done in natural language to SQL query processing and defines the research problem together with the identification of technologies to solve the problem. Chapter 3 is a detailed description on the technologies used to convert natural language to its corresponding T-SQL statements. Chapter 4 discusses about the approach taken to convert natural language to its corresponding T-SQL statements while Chapter 5 illustrates and discusses about the design of the application. Chapter 6 gives details about how this solution is implemented whereas Chapter 7 shows how the NLIDB converter for Customer Relationship Index works in the real world after its implementation. Chapter 8 evaluates the solution and the approach along with its success as a whole and Chapter 9 concludes the research findings with a note on future work.

1.8 Summary

This chapter gives an overall picture of the entire project presented in this thesis. The background and motivation, problem definition, hypothesis, objectives and a brief overview of the solution is discussed in this chapter. The next chapter presents a critical review of literature written on natural language interfaces to databases.

Chapter 2

2 Challenges in Natural Language Interfaces for Databases

2.1 Introduction

In Chapter 1, an overall introduction to the research project is presented. This chapter critically reviews the literature on natural language interfaces for databases. In order to do so this chapter is structured with the following major sub sections, namely; early developments in natural language interfaces for databases, approaches used for NLIDB, techniques used to implement NLIDB and recent developments in natural language interfaces for databases. Finally, this chapter discusses the research gap in the current domain and defines the research problem considering what was mentioned under problem in brief in Chapter 1. This chapter also highlights and reviews the sustainability of the technologies and tools that are used for addressing the research problem.

2.2 Early Developments in Natural Language Interfaces to Databases

Prototype natural language interfaces for databases started appearing in the late sixties and the early seventies [3]. Even though the natural language interfaces for databases received considerable attention during this time period and underwent quite an amount of development, it was met with moderate success [9]. However, there are some applications and systems that do stand out despite their limitations during the time when the technology started to emerge.

The best known natural language interface for database of that period is LUNAR [3], [6], [9], LUNAR is a natural language interface to a database containing chemical analyses of moon rocks. LUNAR answered questions about rock samples brought back from the moon. In this system two databases were used, the chemical analyses database and the literature references database. The program used an Augmented Transition Network (ATN) parser and Woods' Procedural Semantics. The performance of LUNAR was very impressive. It

managed to handle 90% of requests without any error [1]. The system was informally demonstrated at the Second Annual Lunar Science Conference in 1971 [10].

By the late seventies several more natural language interfaces to databases appeared.

RENDEZVOUS, built in the IBM laboratory in San Jose, California is one of them. Here, the users were able to access databases via relatively unrestricted natural language [11]. RENDEZVOUS engaged the user in dialogues to help him/her formulate his/her queries [12] implying that special emphasis was given to query paraphrasing and engaging users in clarification dialogs when there is difficulty in parsing user input.

PHILIQA (1977) this was known as Philips Question Answering System [3], uses a syntactic parser which runs as a separate pass from the semantic understanding passes. This system is mainly involved with problems of semantics and has three separate layers of semantic understanding. The layers are called “English Formal Language”, “World Model Language”, and “Data Base Language” and appear to correspond roughly to the “external", "conceptual", and "internal" views of data.

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. Therefore, this system could be used with large databases, and it could be configured to interface to different underlying database management systems [3], [9]. LADDER used semantic grammars a technique that interleaves syntactic and semantic processing. Although semantic grammars helped to implement systems with impressive characteristics, the resulting systems proved difficult to port to different application domains. Indeed, a different grammar had to be developed whenever LADDER was configured for a new application.

CHAT-80 is one of the best known NLIDB of the early eighties. CHAT-80 is developed in Prolog language. In which English text is transferred into prolog expressions, which were evaluated against the Prolog database. The code of CHAT-80 was circulated widely and formed the basis of several other experimental NLIDB’s [3] [13].

PLANES is another natural language interface to database system developed at the University of Illinois Coordinated Science Laboratory which enabled the casual user to obtain explicit answers from a large relational database of aircraft flight and maintenance data using English. The language processing portion of PLANES uses a number of augmented transition networks, each of which matches phrases with a specific meaning, along with context registers (history keepers) and concept case frames; these are used for judging meaningfulness of questions, generating dialogue for clarifying partially understood questions, and resolving ellipsis and pronoun reference problems [14].

One of the main goals of the eighties was portability of the natural language interfaces to different domains. Therefore, a large part of the research of that time was devoted to portability issues.

ASK (1983) [3], [15] allowed end-users to teach the system new words and concepts at any point during the interaction. ASK was actually a complete information management system providing its own built-in database and ability to interact with multiple external databases electronic mail program and other computer applications. All the applications connected to ASK were accessible to the end-user through natural language request. The users started his/her request in English and ASK transparently generated suitable requests to the appropriate underlying system.

JANUS [16] had similar abilities to interface to multiple underlying systems (databases, expert systems, graphics devices, etc.). All the underlying systems could participate in the evaluation of a natural language request, without the user ever becoming aware of the heterogeneity of the overall system. JANUS is also one of the few systems to support temporal questions (queries involving time).

EUFID (1983) or End-User Friendly Interface to Data management [17] is a system that is independent of both the application and the database management system. EUFID consists of three major modules, analyzer module, mapper module and translator module.

DATALOG (1984) is an English database query system based on Cascaded ATN grammar and provides separate representation schemes for linguistic knowledge, general world

knowledge, and application domain knowledge, DATALOG was highly portable and extendable[18].

TEAM (1987) was a portable easily configurable interface. It was designed to be easily configurable by database administrators with no knowledge of natural language interface to databases. The TEAM system had three major components, an acquisition component, the dialogic language system and a data access component [19].

In spite of the great effort made in the eighties decade hoping that this type of interfaces became of common use failed, probably due to the fact that there were inherent difficulties of language and to the emergence of friendlier graphical and form based interfaces.

In the nineties, although the research in this specific area was no longer as rigorous as in the previous decade, NLIDBs continued to evolve adopting the progress in general NLIs, as well as in the theory of speech, integration of agents to reasoning, multimedia interfaces, generation of more complete dictionaries, search for formalisms, etc. These translated a query in natural language to a logical form that was transformed into a standard form understandable for the database management systems.

2.3 Approaches Used for NLIDB

There are number of approaches that are used for the development of natural language interfaces to database. They mainly consist of Symbolic approaches (rule based approaches), Empirical approaches (Corpus based approaches) and Connectionist approaches (using neural networks)[1]. A brief overview of each approach is presented next.

2.3.1 Symbolic Approach (Rule Based Approach)

Natural language processing appears to be a strongly symbolic activity. Words are symbols that stand for objects and concepts in the real world and they are put together into sentences that follow well specified grammar rules. Hence for several decades' natural language processing research has been dominated by the symbolic approach [20].

Knowledge about language is explicitly encoded in rules or other forms of representation. Then the language is analyzed at various levels to obtain information. Then rules are applied on this obtained information to achieve linguistic functionality. As Human Language capabilities include rule-based reasoning, it is supported well by symbolic processing. In symbolic processing rules are formed for every level of linguistic analysis. It tries to capture the meaning of the language based on these rules.

2.3.2 Empirical Approach (Corpus Based Approach)

Empirical approaches are based on statistical analysis as well as other data driven analysis, of raw data which is in the form of text corpora. A corpus is collections of machine readable text. The approach has been around since NLP began in the early 1950s. Only in the last 10 years or so did the empirical NLP has emerged as a major alternative to rationalist rule-based natural language processing.

Corpora are primarily used as a source of information about language and a number of techniques have emerged to enable the analysis of corpus data. Syntactic analysis can be achieved on the basis of statistical probabilities estimated from a training corpus. Lexical ambiguities can be resolved by considering the likelihood of one or another interpretation on the basis of context.

Recent research in computational linguistics indicates that empirical or corpus based methods are currently the most promising approach to developing robust, efficient natural language processing systems [21], [22]. These methods automate the acquisition of, most of the complex knowledge essential for NLP by training on suitably annotated natural language corpora, e.g. the Penn Treebank of parsed sentences [23].

2.3.3 Connectionist Approach (Using Neural Network)

Since human language capabilities are based on neural networks in the brain, artificial neural networks (also called as connectionist network) provides an essential starting point for modeling language processing. In the recent years, the field of connectionist processing has seen a remarkable development. The sub-symbolic neural network approach holds a lot of promise for modeling the cognitive foundations of language processing. Instead of

symbols, the approach is based on distributed representations that correspond to statistical regularities in language.

There has also been significant research applying neural network methods to language processing [20], [24]. However, there has been relatively little recent language research using sub-symbolic learning, although some recent systems have successfully employed decision trees transformation rules and other symbolic methods. SHRUTI [25] system is a neutrally inspired system for event modeling and temporal processing at a connectionist level.

2.4 Techniques Used to Implement NLIDB

2.4.1 Pattern Matching Techniques

Pattern matching techniques rely on directly mapping the user input to the desired output. Implying that the query in natural language is directly mapped to the required T-SQL statement. However, applying this technique meant that the solution is limited to the specific database. But this technique has the advantage of being simple to implement since this technique does not require elaborate parsing and interpretation modules.

2.4.2 Syntax Based Techniques

In syntax based systems the user input is parsed (syntactically analyzed) and the resulting parse tree is directly mapped to a SQL expression. Syntax-based systems use a grammar that describes the possible syntactic structures of the user's questions. Syntax based NLIDBs usually interface to application-specific database systems that provide database query languages carefully designed to facilitate the mapping from the parse tree to the database query.

The main advantage of using syntax based approaches is that they provide detailed information about the structure of a sentence. A parse tree contains a lot of information about the sentence structure; starting from a single word and its part of speech, it identifies the dependencies between words. Therefore, it says how words can be grouped together to form a phrase, how phrases can be grouped together to form more complex phrases, until

a complete sentence is built. Having this information, we can map the semantic meanings to certain production rules (or nodes in a parse tree).

Unfortunately, not all nodes should be mapped, some nodes have to be left just as they are without adding any semantic meanings. And it is not always clear which nodes should be mapped and which should not. The second problem is a sentence can have multiple correct parse trees, and if all are translated, they may lead to different query results. The last problem is that it is difficult for a syntax based approach to directly map a parse tree into some general database query language, such as SQL (Structured Query Language).

2.4.3 Semantic Grammar Techniques

A semantic grammar system is very similar to the syntax based system, meaning that the query result is obtained by mapping the parse tree of a sentence to a database query. The basic idea of a semantic grammar system is to simplify the parse tree as much as possible, by removing unnecessary nodes or combining some nodes together. Based on this idea, the semantic grammar system can better reflect the semantic representation without having complex parse tree structures.

The main drawback of semantic grammar approach is that it requires some prior-knowledge of the elements in the domain, therefore making it difficult to port to other domains.

2.5 Recent Developments in NLIDB

The following table lists the recent developments in the field of natural language interface to database.

Research	**Technologies Used**	**Achievements**	**Limitations**
Constructing an Interactive Natural Language Interface for Relational Databases	Dependency Parsing Parse tree node mapping Parse tree structure adjusting Interactive communicating for clarification	Naive users are able to accomplish logically complex query tasks. High effectiveness and usability	Solving ambiguity in tokens when used in different contexts

Natural language web interface for database	Word check Tokenization Excess word removing Mapping rules SQL elements identifier Mapping SQL templates SQL query generation	NLWIDB system which converts a wide range of text queries (English questions) into formal ones that can then be executed against a database by employing robust language processing techniques and methods. Shows a realistic potential for bridging the gap between computer and the casual end users.	Portability of the system due to lexicon
Modern Natural Language Interfaces to Databases: Composing Statistical Parsing with Semantic Tractability	Statistical parser Strong semantic model coupled with "light re-training"	Correctly map from parsed questions to the corresponding SQL queries achieves 94% accuracy	Database dependent

Table 1 Summary of recent developments

2.6 Research Problem

The Customer Relationship Index is an internally developed web application by the Software Team of Brandix Apparel Solution LTD - Essentials. This system is being used by the CEOs and the general managers of the Brandix Group. Thus making it a critical system for the stakeholders. The information in the Customer Relationship Index is used by the steering committee of the Brandix group to make business decisions on a regular basis therefore, information from the system is queried every so often.

Apart from the reports that the system provides, there is no way for the Customer Relationship Index user to access the data or information available in the Customer Relationship Index database without the assistance of the Software Team of Brandix Apparel Solution LTD – Essentials.

It is also noticed that the informational requirements of a typical user of the Customer Relationship Index changes from time to time based on the direction the steering committee provides. Therefore, it would be convenient if the user can query the information from the Customer Relationship Index database himself or herself using a natural language, preferably English.

This brings light to the research problem at hand. How can a natural language interface to the Customer Relationship Index be implemented? Is it possible to implement a natural language interface to the Customer Relationship Index database? Will the Customer Relationship Index database design support such an implementation? What natural language interface to database conversion approach or technique should be adopted for this implementation? Will the implementation prove successful and benefit the typical Customer Relationship Index user? Therefore, this research is conducted to explore the feasibility of a NLIDB implementation for the Customer Relationship Index and to assess its success in terms of accuracy, efficiency and performance.

2.7 Summary

This chapter critically reviewed the early and recent developments in the research domain of natural language interfaces to databases. It also discusses how natural language interfaces can be implemented, the techniques used for the said implementation and the issues faced during implementation. There are many literatures available in this problem domain and each presents a different approach in solving the problem at hand. Therefore, it is evident that the technique used for natural language interface to database and SQL conversions is mostly problem specific. Finally, this chapter presents the research problem that this research intends to address. Next chapter presents the technology adopted for solving the research problem.

Chapter 3

3 Technology

3.1 Introduction

The previous chapter critically reviews the current developments, major recent breakthroughs and the issues faced when implementing a natural language interface to a database. This chapter discuss the technologies adopted for implementing a natural language interface for Customer Relationship Index database and gives explanation as to why each technology is selected.

3.2 State of the art in natural language processing

“Understanding” language means, among other things, knowing what concepts a word or phrase stands for and knowing how to link those concepts together in a meaningful way. In order for the machines to understand natural language used by human, it has to be processed in such a way so that meaning can be extracted from it in a way that a machine understand. Essentially natural language processing and meaning extraction can be done using the following steps.

1. Sentence splitting: Identifying sentence boundaries in text.
2. Tokenization: Splitting a sentence into individual words called tokens.
3. Lemmatization: Converting a word to its root form. E.g. says, said, saying will all map to root form - say
4. Stemmer: It is similar to a lemmatizer, but it stems a word rather than get to the root form. e.g. laughed, laughing will stem to laugh. However, said, saying will map to sa - which is not particularly enlightening in terms of what "sa" means.
5. POS tagger: Tags a word with the Part of Speech - what is a noun, verb, preposition etc.
6. Parser: Links words with POS tags to other words with POS tags.

There are many natural language processing libraries, packages and toolkits freely available at the moment. Listed below are some of the widely used natural language processing libraries for natural language processing application development.

1. Stanford CoreNLP
2. SharpNLP - open source natural language processing tools
3. Syn Bot (Oscova and SIML)
4. OpenNLP

After critically reviewing the available libraries based on the functions they provide for natural language processing, their capabilities, the accuracy of results, support provided, ease of use and documentation, the Stanford CoreNLP and Stanford Parser is selected for the implementation of NLIDB converter for Customer Relationship Index.

3.2.1 Stanford CoreNLP

Stanford CoreNLP toolkit provides a set of natural language analysis tools which can take raw English language text input and give the base forms of words, their parts of speech, whether they are names of companies, people, etc., normalize dates, times, and numeric quantities, and mark up the structure of sentences in terms of phrases and word dependencies, and indicate which noun phrases refer to the same entities.

Stanford CoreNLP is an integrated framework, which make it very easy to apply language analysis tools to a piece of text. Starting from plain text.

3.2.2 Stanford Parser for .NET

The natural language parser program provided by the Stanford Parser works out the grammatical structure of sentences, for instance, which groups of words go together (as "phrases") and which words are the subject or object of a verb. The parser provides Universal Dependencies and Stanford Dependencies output as well as phrase structure trees. For the development of NLIDB converter for Customer Relationship Index the parser output of Stanford dependencies is used. Stanford dependencies provides a representation of grammatical relations between words in a sentence. They have been designed to be

easily understood and effectively used by people who want to extract textual relations. Stanford dependencies (SD) are triplets: name of the relation, governor and dependent.

3.3 Microsoft .NET Framework

A programming infrastructure created by Microsoft for building, deploying, and running applications and services that use .NET technologies, such as desktop applications and Web services. The .NET Framework contains three major parts: The Common Language Runtime. the Framework Class Library. ASP.NET.

3.4 Console Applications

A console application is a computer program designed to be used via a text-only computer interface, such as a text terminal. In the context of C#, a console application is an application that takes input and displays output at a command line console with access to three basic data streams: standard input, standard output and standard error. A console application is robust, light-weight and efficient in its execution. When combining with heavy external natural language processing libraries it’s better to focus on efficient execution.

3.5 Three Tier Architecture

The three-layer architecture is a software design pattern and well-established software architecture. The three-tier architecture model, which is the fundamental framework for the logical design model, segments an application's components into three tiers of services. These tiers do not necessarily correspond to physical locations on various computers on a network, but rather they are logical layers of the application. The functional processes of user interface, business logic and data access layer are developed and maintained as independent modules on separate platforms.

Following are brief descriptions of the services allocated to each tier:

- Presentation tier

This layer gives a user access to the application. The main functionality of the presentation layer is to present data to the user and permits and data entry.

- Middle tier / business logic layer / business layer / business class

This layer consists of business and data rules. Hence this layer can be used to enforce business rules, such as business algorithms and legal or governmental regulations, and data rules, which are designed to keep the data structures consistent within either specific or multiple databases.

- Data tier / data services layer / data access layer

This layer interacts with persistent data usually stored in a database or in permanent storage. This is the actual DBMS access layer. The data tier can be accessed through the business logic layer and on occasion by the presentation tier. This layer consists of data access components (rather than raw DBMS connections) to aid in resource sharing and to allow clients to be configured without installing the DBMS libraries and ODBC drivers on each client.

The below image shows the layers of the three tier architecture and how each layer communicates with each other.

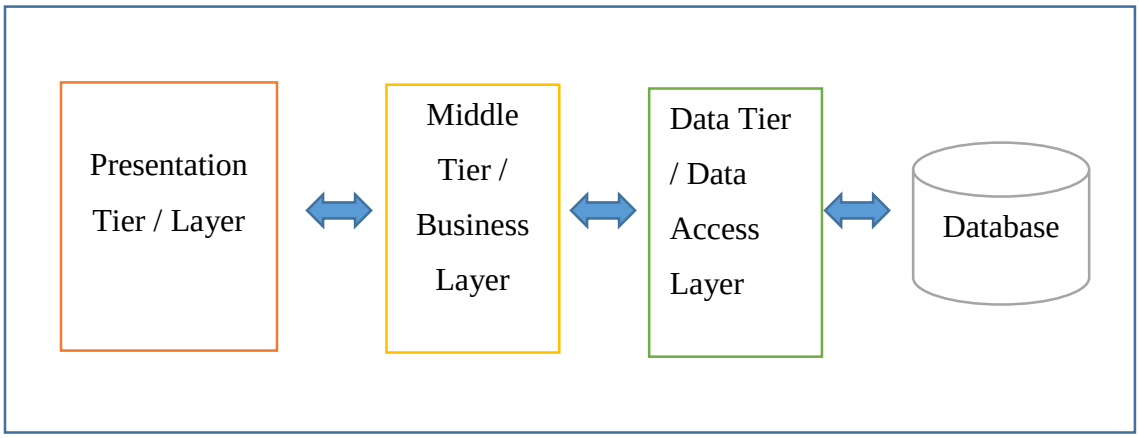


Figure 3.1 Layers of the three tier architecture

During an application's life cycle, the three-tier approach provides benefits such as reusability, flexibility, manageability, maintainability, and scalability. The components that are created can be shared, reused. This architecture allows changes to be applied easily without having to change or update a lot of code which makes it easier to develop an application which is subjected to change throughout the initial stages based on research findings.

3.6 Microsoft Entity Framework

The Microsoft ADO.NET Entity Framework is an Object/Relational Mapping (ORM) framework that enables developers to work with relational data as domain-specific objects, eliminating the need for most of the data access plumbing code that developers usually need to write. Using the Entity Framework, developers issue queries using LINQ, then retrieve and manipulate data as strongly typed objects. The Entity Framework's ORM implementation provides services like change tracking, identity resolution, lazy loading, and query translation so that developers can focus on their application-specific business logic rather than the data access fundamentals. It is an enhancement to ADO.NET that gives developers an automated mechanism for accessing & storing the data in the database.

An ORM is a tool for storing data from domain objects to relational database like MS SQL Server, in an automated way, without much programming. ORM includes three main parts: Domain class objects, Relational database objects and Mapping information on how domain objects map to relational database objects (tables, views & stored procedures). ORM allows us to keep our database design separate from our domain class design. This makes the application maintainable and extendable. It also automates standard CRUD operation (Create, Read, Update & Delete) so that the developer doesn't need to write it manually.

3.7 Microsoft SQL Server

This is a relational database management system developed by Microsoft. As a database server, it is a software product with the primary function of storing and retrieving data as requested by other software applications—which may run either on the same computer or on another computer across a network (including the Internet).

3.8 Nugget Packages

NuGet is the package manager for the Microsoft development platform including .NET. The NuGet client tools provide the ability to produce and consume packages. The NuGet Gallery is the central package repository used by all package authors and consumers. Through NuGet the following package libraries were added to the NLIDB converter for Customer Relationship Index – Stanford CoreNLP, Stanford Parser, and Microsoft Entity Framework.

3.9 Summary

This chapter discusses about the technologies adopted to provide a solution to the problem discussed in chapter 2. The technology, its state of the art, its advantages and why it was selected to address the current problem at hand is explored and discussed as well. Next, we describe the approach taken to address the problem in Chapter 4.

Chapter 4

4 Approach

4.1 Introduction

Having defined the problem that needs to be addressed in Chapter 2, the technology required for the proposed solution is presented in Chapter 3. The current chapter, Chapter 4 describes the approach used to implement a Natural Language Interface for the Customer Relationship Index in order to address the research problem. This chapter will present the hypothesis and discuss details of the inputs and outputs of the system, the process used to convert the inputs in to the output and the overall features and the typical users of the system.

4.2 Hypothesis

Successful implementation of Natural Language Interface to Database for the Customer Relationship Index can bridge the gap between the languages used by humans and the computers, allowing the system users to retrieve the required information they need directly from the database without seeking assistance or depending on the software team of Brandix Apparel Solution LTD – Essentials.

4.3 Input

The inputs used in the implementation of Natural Language Interface to Database for the Customer Relationship Index uses textual input provided by the user via a key board. The textual input is in English and the NLIDB converter for the Customer Relationship Index will support only English, as many natural language processing libraries support English and very rarely other languages such as Chinese sometimes. English is widely used by the senior management of Brandix Apparel Solution LTD – Essentials, hence the preferred language for the NLIDB converter for the Customer Relationship Index is English.

The NLIDB converter for the Customer Relationship Index is designed as a console application. When the console application runs, it prompts the user to type the question he or she wishes to run against the Customer Relationship Index database. As response to the

message prompted by the NLIDB converter, the user types in the question in natural language which is considered as the input string which is then converted to the SQL query by the NLIDB converter for the Customer Relationship Index.

The figure below shows the wireframe design for the NLIDB converter for the Customer Relationship Index console application at the time it accepts the textual inputs from the user.

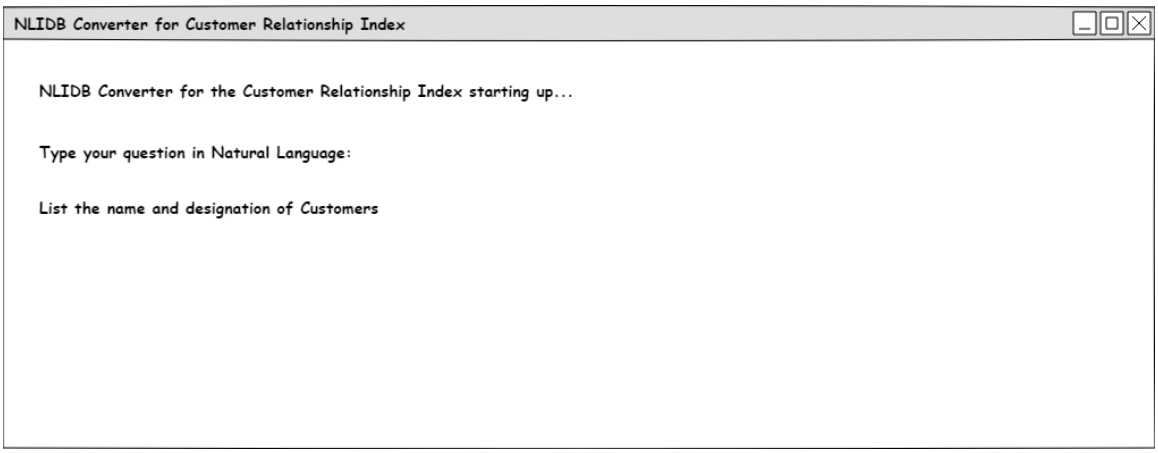


Figure 4.1 Wireframe design for NLIDB converter for Customer Relationship Index – Inputs

The user input is a text string written in English. For example, “List the name and designation of Customers”

4.4 Output

The output of the NLIDB converter for Customer Relationship Index is the corresponding SQL statement which is generated as a result of executing the console application. The corresponding SQL statement will be T-SQL as this solution is designed and generated for a Microsoft SQL Server database.

The figure below shows the wireframe design of the NLIDB converter for the Customer Relationship Index's output after the conversion of natural language to SQL has successfully completed.

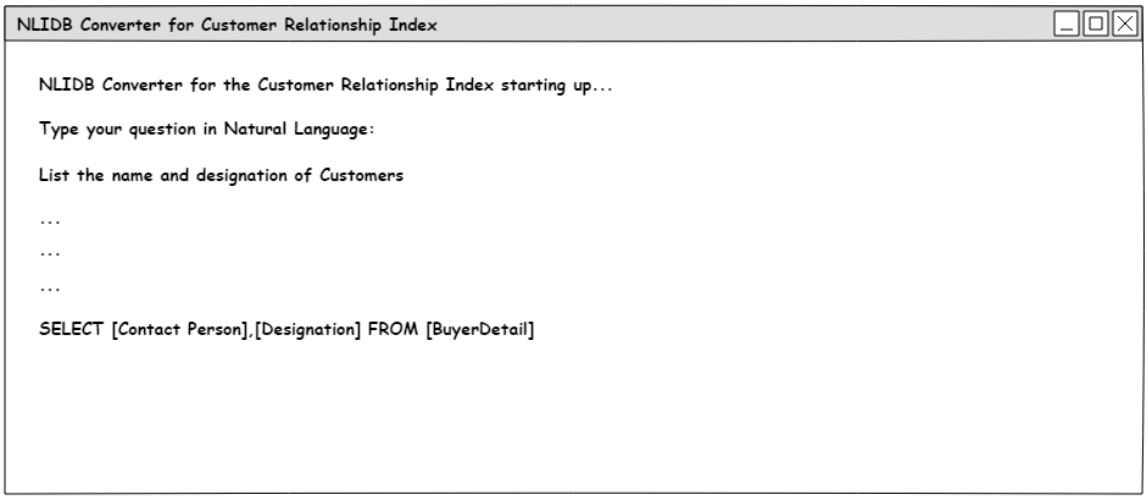


Figure 4.2 Wireframe design for NLIDB converter for Customer Relationship Index – Outputs

The system output of the NLIDB converter for Customer Relationship Index is also a text string which is the corresponding T-SQL statement for the input user query in natural language. For example, "SELECT [Contact Person], [Designation] FROM [BuyerDetail]"

4.5 Process

In order to convert an input in natural language to its corresponding T-SQL form the NLIDB converter for Customer Relationship Index first needs to understand the meaning of the posted question. To be able to understand the meaning of the posted question the NLIDB converter for Customer Relationship Index must collaborate with a natural language processing tool kit which supports functions like sentence splitting, tokenizing, part of speech tagging, named entity recognition and identification of dependencies between tokens to extract meaning.

After the linguistic analysis of the input sentence completes the tokens are matched against a predefined set of natural language rules stored in a lexicon table. This table maps the tokens of the natural language input onto the formal objects (relation names, attribute

names, etc.) Of the Customer Relationship Index database. Through the lexicon table the NLIDB converter for the Customer Relationship Index is able to identify and map the tokens with the corresponding SQL elements or nodes.

By placing the mapped tokens in its respective positions and by combining the tokens with the key words in T-SQL, primarily “SELECT”, “FROM” and “WHERE”, the corresponding T-SQL statement is then generated by the NLIDB converter for Customer Relationship Index.

The process in which the NLIDB converter for Customer Relationship Index transforms its inputs to the outputs are illustrated in the figure below.

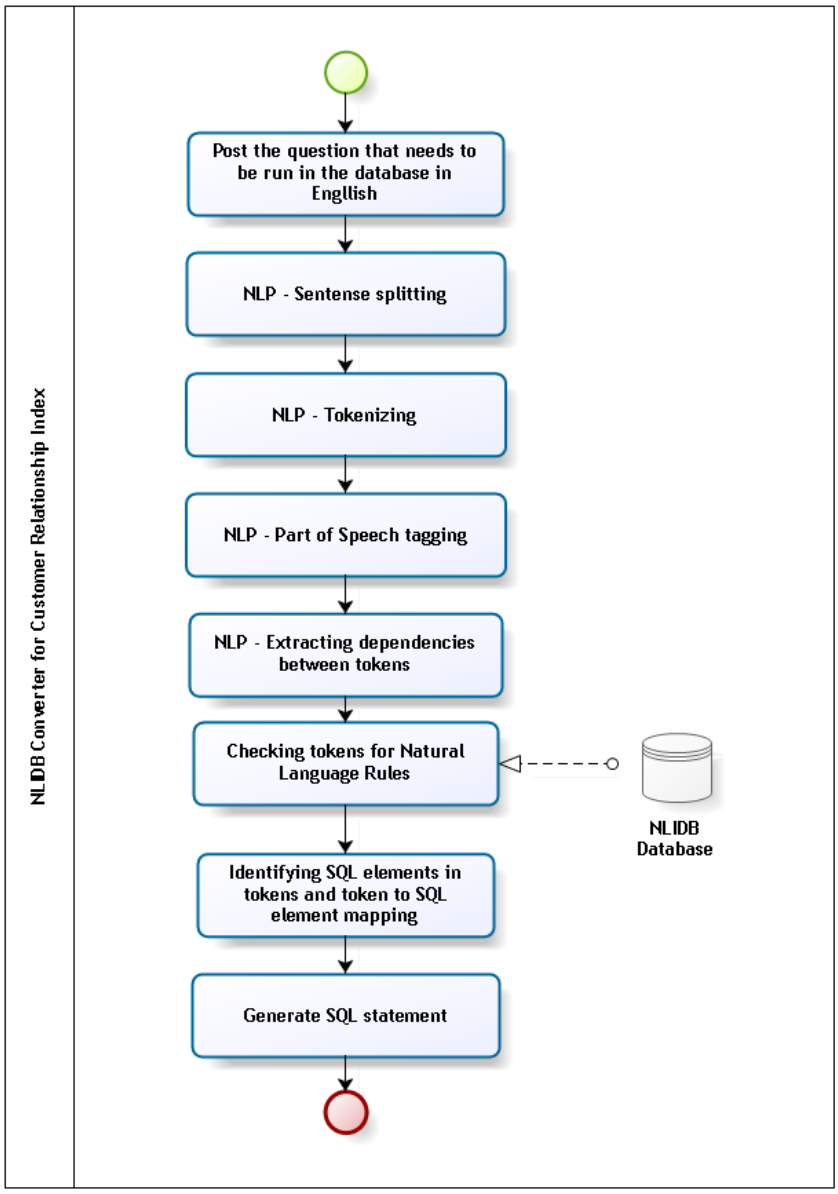


Figure 4.3 Process diagram of the NLIDB converter for Customer Relationship Index

4.6 Features

1. The NLIDB converter for Customer Relationship Index uses English for its natural language to T-SQL conversion.

2. Input collection and output display is done through the keyboard and monitor respectively.
3. The NLIDB converter for Customer Relationship Index is shipped with a simple console UI as to make the processing more efficient and fast.
4. Output generation is done within 10-20 seconds.
5. The NLIDB converter for Customer Relationship Index is expressive and a user friendly due to explaining the process that takes place within the converter.

4.7 Users

The primary users of the NLIDB converter for Customer Relationship Index are mainly the senior management and the leadership team of Brandix Apparel Solution LTD – Essentials who will use this system in their day to day lives to extract the information they need to make the business decisions with regard to managing the company's clients and the business relationships they maintain with them. The senior management and the leadership team are fluent in English and do not share an IT, tech savvy background.

The secondary users of the NLIDB converter for Customer Relationship Index is the Brandix Apparel Solution LTD – Essentials software development team who will evaluate the solution before its released to the management's use. This is a team of highly skilled IT professionals consisting of Database Architects and Software Engineers. This team will later play the administrator role of the system by performing the administrative functionalities and provide the necessary training to the primary users as and when needed.

4.8 Summary

This chapter discusses the approach taken to convert a question in natural language form in to its corresponding T-SQL statement by presenting the hypothesis, the inputs and the outputs and elaborating on the process which converts the inputs in to the desired outputs. The next chapter presents the design of the NLIDB converter for Customer Relationship Index by discussing the modules and components of the application and its functionality.

Chapter 5

5 Design

5.1 Introduction

Chapter 4 presented the overall approach to the proposed solution, the NLIDB converter for Customer Relationship Index for converting questions posted in natural language to T-SQL statements. This chapter presents the details of the design of NLIDB converter for Customer Relationship Index by discussing the high level architecture of the system and each of the modules the system consists of in detail. The NLIDB converter for the Customer Relationship Index consist primarily of five main components namely, the User Interface, the Linguistic Component, the Business Logic Module, the Data Access Component and the Database Component collectively and individually interacting with each other to produce the desired results.

5.2 Top Level Architecture

The top level architecture of the NLIDB converter for Customer Relationship Index is shown in the figure below.

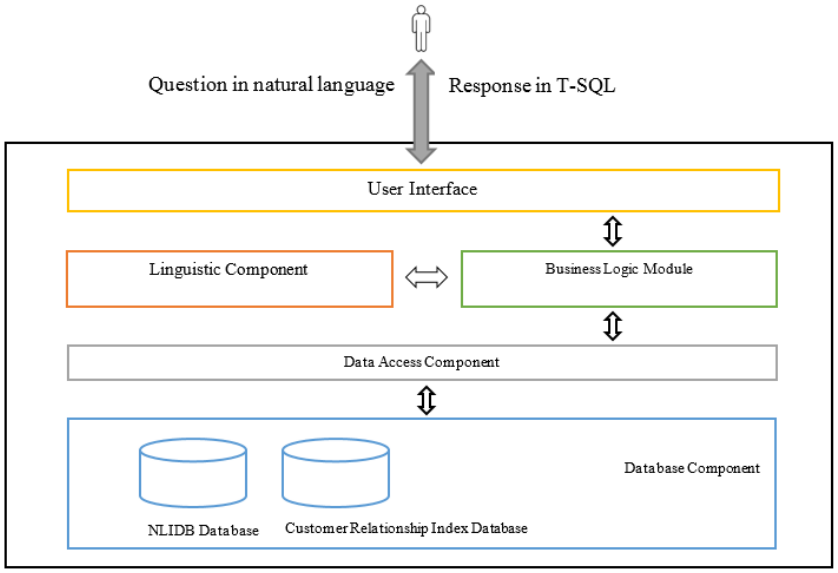


Figure 5.1 Top level architecture of NLIDB converter for Customer Relationship Index

The interconnections between each module is displayed in this diagram. The two way arrows indicate that the communications between modules are done both ways.

In order to make the application more light-weight and simple the NLIDB converter for Customer Relationship Index is designed as a console application. As the NLIDB converter for Customer Relationship Index communicates with an external natural language processing library for linguistic processing, it's necessary to keep the main application as leaner, faster and more light-weight as much as possible in order to avoid longer processing times.

Under the forthcoming subsections, each module of the NLIDB converter for Customer Relationship Index and the role it plays is discussed in detail.

5.3 The User Interface

This component is responsible for interacting with the user. The user interface for the NLIDB converter for the Customer Relationship Index is a console interface. Its main task is to collect the user input (the natural language question) and display the notifications and the T-SQL statement returned from the NLIDB converter for Customer Relationship Index.

The user input is collected via key board and the output is displayed to the user through a screen.

The figure below shows the wireframe diagram for when the user is prompted to type in the input and the NLIDB converter for Customer Relationship Index accepts the input text.

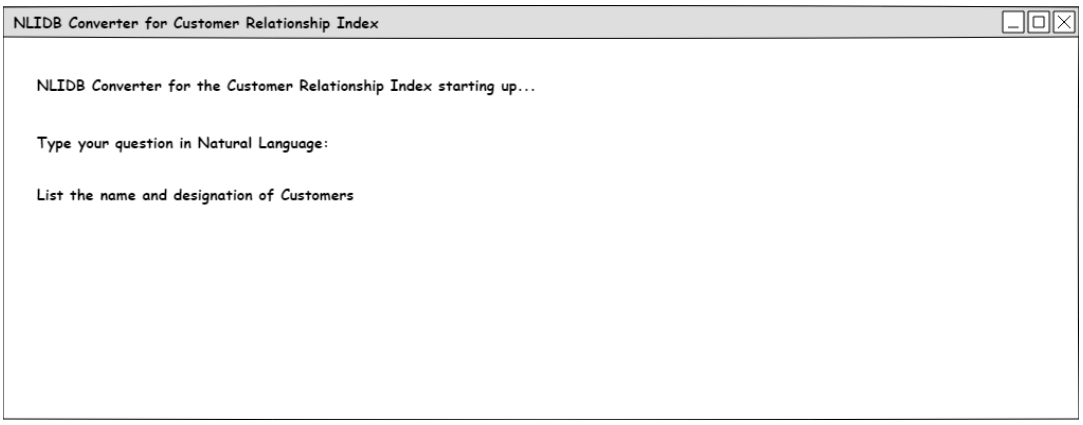


Figure 5.2 Wireframe diagram for NLIDB converter for Customer Relationship Index - Inputs

The figure below shows the wireframe diagram for when the NLIDB converter for Customer Relationship Index displays the output to the user.

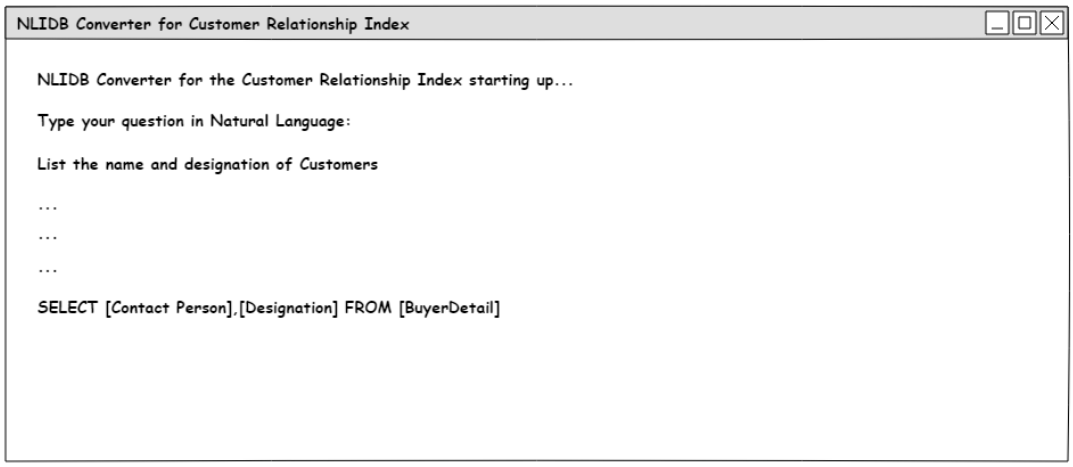


Figure 5.3 Wireframe diagram for NLIDB converter for Customer Relationship Index - Output

5.4 The Linguistic Component

This component is responsible for assisting the NLIDB converter for the Customer Relationship Index translate the raw English input string submitted by the user in to an

understandable format by performing certain language analysis techniques. For the linguistic component an external natural language library has to be in cooperated as it allows for fast, reliable analysis of arbitrary texts with an overall high quality of text analysis.

In order to understand the linguistics of the input string submitted by the user, the linguistic component needs to perform the following functions on the input.

1. Sentence Splitting

Splits a sequence of tokens into sentences. This allows the NLIDB converter for the Customer Relationship Index to identify if the input string is a complex query or a simple query.

2. Tokenizing

Given a character sequence and a defined document unit, tokenization is the task of slicing it up into pieces, called tokens. The extracted tokens are then processed to identify the corresponding SQL elements in the Customer Relationship Index database.

3. Identification of base form of words

Morphological analysis to accurately identify the lemma for each word. Lemma of each word is extracted from a process called Lemmatization. Lemmatization uses a vocabulary and morphological analysis of words aiming to remove inflectional endings and to return the base or dictionary form of a word.

4. Part of Speech tagging for tokens

This function assigns part of speech tags to each token in the input string. These part of speech tags for English are based on the Penn Treebank tag set which is widely used for POS tagging in natural language libraries. This tag set is a result of a project carried out in the University of Pennsylvania about how to annotate naturally-occurring text for linguistic structure.

Listed below are the Penn Treebank tag set and the meaning of each tag.

Tag	Tag Description
$	dollar
``	opening quotation mark
''	closing quotation mark
(	opening parenthesis
)	closing parenthesis
,	comma
--	dash
.	sentence terminator
:	colon or ellipsis
CC	conjunction, coordinating
CD	numeral, cardinal
DT	determiner
EX	existential there
FW	foreign word
IN	preposition or conjunction, subordinating
JJ	adjective or numeral, ordinal
JJR	adjective, comparative
JJS	adjective, superlative
LS	list item marker
MD	modal auxiliary
NN	noun, common, singular or mass
NNP	noun, proper, singular
NNPS	noun, proper, plural
NNS	noun, common, plural
PDT	pre-determiner
POS	genitive marker
PRP	pronoun, personal
PRP$	pronoun, possessive

RB	adverb
RBR	adverb, comparative
RBS	adverb, superlative
RP	particle
SYM	symbol
TO	to as preposition or infinitive marker
UH	interjection
VB	verb, base form
VBD	verb, past tense
VBG	verb, present participle or gerund
VBN	verb, past participle
VBP	verb, present tense, not 3rd person singular
VBZ	verb, present tense, 3rd person singular
WDT	WH-determiner
WP	WH-pronoun
WP$	WH-pronoun, possessive
WRB	Wh-adverb

Table 2 Penn Treebank tags

By being able to tag the tokens for part of speech, it enables the NLIDB converter for the Customer Relationship Index identify nouns in the tokens that can directly be cross checked against a lexicon table for identifying the database elements, primarily tables, views and columns.

5. Dependency Parsing

Mark up the structure of the sentence in terms of phrases and word dependencies. Dependency parsing analyzes the grammatical structure of a sentence, establishing relationships between "head" words and words which modify those heads.

By evaluating the dependencies of the sentences and tokens the NLIDB converter for the Customer Relationship Index is able to recognize the interdependencies of words when its mapping the tokens to its corresponding database elements through the lexicon table.

5.5 Business Logic Module

This module defines and contains the business logic specific to the NLIDB converter for Customer Relationship Index for converting the questions in natural language to its corresponding T-SQL statement.

The business logic module comprises of the following,

1. The NLIDB converter sub module that contains the main program for converting the question in natural language to T-SQL
2. The business layer or commonly known as the business class which manages the communication between the data access component and the NLIDB converter sub module.
3. The NLIDB logger that logs all activities taking place in the NLIDB converter for Customer Relationship Index in to text file from the moment a new question is posted to the system for later reference.
4. The Error logger which logs the error message when an exception is thrown while the system executes.

5.6 Data Access Component

This module, commonly known as the data access layer sits between the business logic module and the database component enabling communication between the two modules. The data access component contains classes and functions for reading data from and writing data to the NLIDB database.

5.7 NLIDB Database Component

This is responsible for performing traditional database management functions essentially consisting of inserts, updates and deletes in the NLIDB database. For example, storing the natural language question in the database when its posted by the user, storage and

management of tokens, mapping tokens to its corresponding SQL nodes, storing of the converted T-SQL statement in the database for later reference etc.

Apart from performing the traditional database management functions, the most significant task the database component performs is maintaining the lexicon table. The lexicon table is a table that is used to map the words of the natural language input onto the formal objects (relation names, attribute names, etc.) of the Customer Relationship Index database. Both parser and semantic interpreter make use of the lexicon table. A natural language database systems make use of syntactic knowledge and knowledge about the actual database in order to properly relate natural language input to the structure and contents of that database.

The following table contains details of a sample lexicon maintained for the NLIDB converter for Customer Relationship Index.

Token	**Database Structure Name**	**SQL Element**
Brand	BuyerDetail	Table
Brands	BuyerDetail	Table
Brand_Contacts	BuyerDetail	Table
Customer	BuyerDetail	Table
Customers	BuyerDetail	Table
Buyer	BuyerDetail	Table
Buyers	BuyerDetail	Table
Name	[Contact Person]	Column
Names	[Contact Person]	Column
Calling_Name	[Contact Person]	Column

Table 3 Lexicon table in NLIDB database

The lexicon defines a set of natural language mapping rules. If we take the first row as an example, it tells the NLIDB converter for the CRI that if the token “Brand” was found in the list of input tokens, then the name of the database element it should refer to is “BuyerDetail” which is a table signifying a mapping between the input token and the SQL node or element.

5.8 Summary

This chapter presented the overall high level architecture of the NLIDB converter for Customer Relationship Index. It also listed out the main components of the system describing the interconnections between components and the functionality of each component. The next chapter discusses about the implementation of the NLIDB converter for Customer Relationship Index including the algorithms, pseudo code and relevant code segments.

Chapter 6

6 Implementation

6.1 Introduction

Chapter 5 presented the design of the NLIDB converter for Customer Relationship Index covering the high level architecture, the components the system is made up of and discussed each component in detail with regard to its intended functionality, coupling and cohesion of components and the responsibilities of each component. This chapter presents the details of the implementation of each component presenting the algorithm, pseudo code and relevant code segments while indicating where each technology mentioned under Chapter 3 is used in the development of the NLIDB converter for Customer Relationship Index.

6.2 Implementation Essentials for the NLIDB converter for Customer Relationship Index

The NLIDB converter for Customer Relationship Index is developed using the .NET Framework 4.5 in Visual Studio 2013 using the C# language. The NLIDB converter for Customer Relationship Index is a console application which in cooperates the Stanford CoreNLP toolkit and the Stanford Parser for the natural language processing it performs. This is a client server application as the NLIDB database and the Customer Relationship Index database is located in the application server whereas the NLIDB converter for the Customer Relationship Index is going to be located in the client machines of the system users.

6.2.1 Softwares Used

For the development of the NLIDB converter for Customer Relationship Index the following software is needed.

- SQL Server 2012
- Visual Studio 2013

6.2.2 Hardware Requirements

To run the NLIDB converter on client machines the following hardware requirements have to be met.

- Processor – Intel core i5 CPU @ 1.90GHz 2.50GHz
- RAM – 8GB
- This application is OS independent. – can be run on both 32bit and 64bit machines

However, for the implementation of the NLIDB converter for Customer Relationship Index there are not specific hardware requirements.

6.2.3 Frameworks Used

- NET Framework 4.5
- Microsoft Entity framework 6.1.3

6.2.4 Languages Used for Implementation

- The NLIDB converter for Customer Relationship Index is developed using C# and LINQ
- The database module is developed using T-SQL

6.3 Implementation of the NLIDB Database

The NLIDB database component is developed as a lightweight MS SQL Server 2012 database with the collation set to "SQL_Latin1_General_CP1_CI_AS". The collation is a set of rules that define how data is sorted and compared. This database consists of the following SQL elements – Tables, Views and Stored Procedures created using data definition (DDL) and data manipulation (DML) T-SQL queries.

Depending on the role it plays and the data it holds, the tables in the NLIDB database is of two types. Master data tables and Transactional data tables.

Master data tables contain data that seldom changes. For example, the tables containing the natural language rules, the table containing the token types based on the Penn Treebank tags and the table containing the SQL node types are master data tables in the NLIDB

database. The other tables that record and store the natural language questions, the corresponding T-SQL statement, the distinct tokens in each question, the dependencies among each tokens in the question and the token to SQL node mapping per question are all transactional data tables.

After the identification of the entities, attributes and the connection between the entities the NLIDB database was implemented. During the implementation all identified tables were put through normalization up to the 3rd level in order to maintain the data integrity and to make the NLIDB database more efficient.

The figure below shows the database diagram of the NLIDB

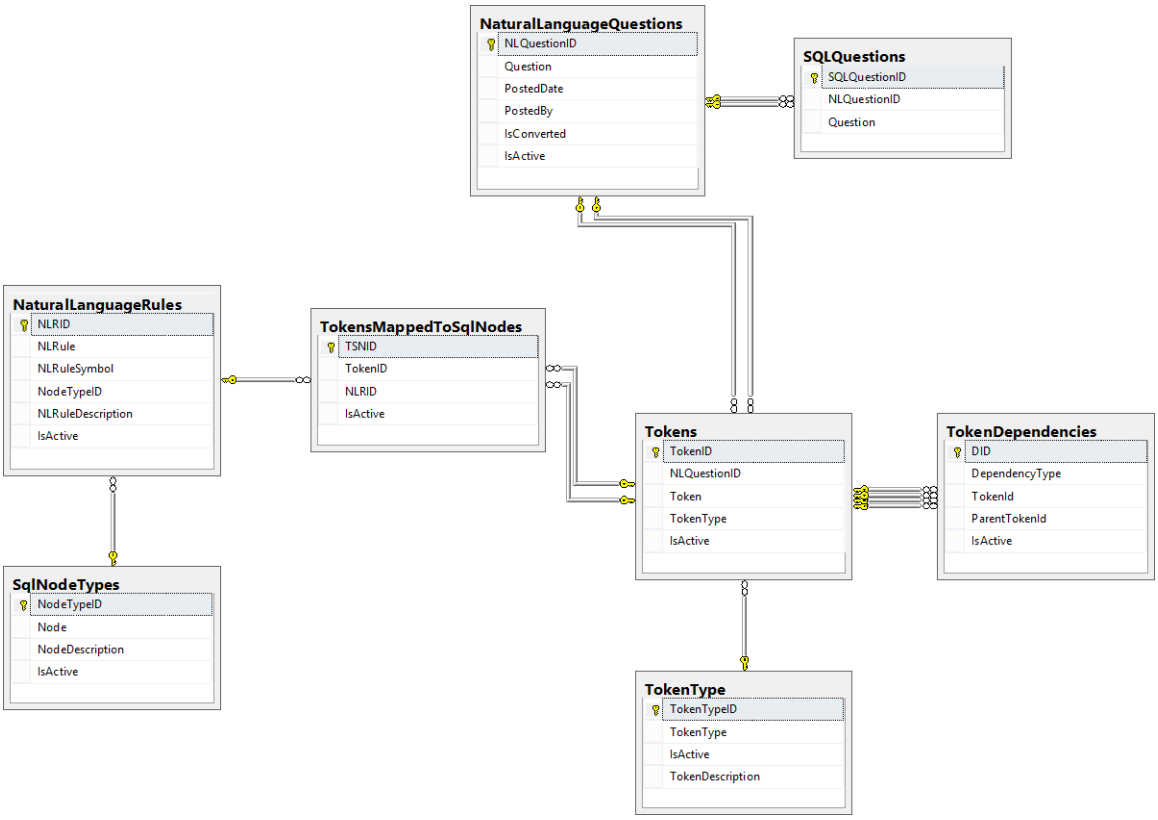


Figure 6.1 Database diagram of NLIDB

The table below lists out the tables in the NLIDB database and the data it stores.

Table in NLIDB database	The data stored
NaturalLanguageQuestions	Stores the natural language question and the details of the question (asked by who, when etc.)
NaturalLanguageRules	Stores the natural language rules defined for the different SQL node types. This is a master data table.
SqlNodeTypes	Stores a predefined set of tags which corresponds to elements or nodes in the T-SQL statement. SN Select Node ON Operator Node FN Function Node NNT Name Node Table VN Value Node QN Quantifier Node LN Logic Node NNV Name Node View NNC Name Node Column This is a master data table.
SQLQuestions	Stores the corresponding T-SQL statement for the question in natural language
TokenDependencies	Stores the dependencies between two tokens by recording the child and parent token
Tokens	Stores the extracted tokens for a given natural language question
TokensMappedToSqlNodes	Stores the tokens that were successfully mapped to SQL nodes through the natural language rules
TokenType	Stores the token types taken from the Penn Treebank tag set. This is a master data table

Table 4 The description and its role of the tables in the NLIDB database

6.4 Implementation of the NLIDB converter for Customer Relationship Index

For the implementation of the NLIDB converter for Customer Relationship Index the three tier architecture is adopted. The main advantage for specifically selecting the three tier architecture for this application is because the presentation layer, the business layer and the data access layer exist separately making it easy to maintain the application and apply a change if needed with relative ease. As this is a system based on research findings, frequent changes are anticipated therefore using an architecture that supports maintainability and easy application of changes is good practice.

The figure below shows the three tier architecture in the application development of the NLIDB converter for Customer Relationship Index.

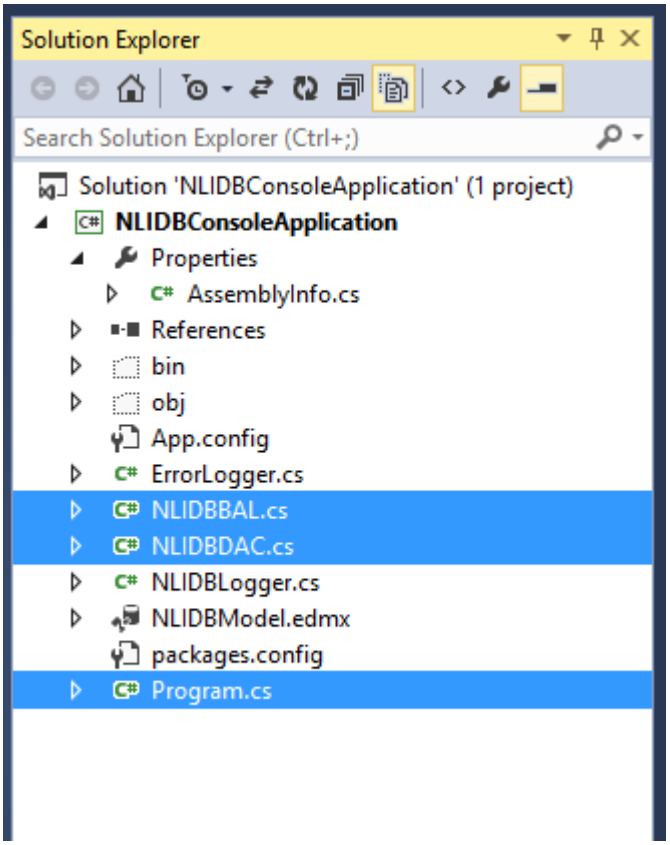


Figure 6.2 The three tier architecture in the NLIDB converter for Customer Relationship Index implementation

Class NLIDBDAC is the data access layer, together with the entity model it makes up the Data Access Component.

Class NLIDBBAL is the business layer. Class NLIDBBAL, NLIDBLogger, ErrorLogger and Program together makes up the business logic module. The execution calls to the Linguistic module which is made up of the Stanford CoreNLP toolkit and the Stanford Parser is also done by the business logic module through the Program class.

The presentation layer consists of the console that displays messages and collects inputs from the NLIDB converter for Customer Relationship Index user based on the execution calls made by the class Program.

Furthermore, the NLIDB converter for Customer Relationship Index uses Object Oriented Programming because it supports inheritance, polymorphism and encapsulation in order to make the most of the effort put in to development. Application of OOP will be further discussed at the relevant component level.

6.4.1 Class Diagram

The below diagram shows the class diagram of the NLIDB converter for Customer Relationship Index.

Figure 6.3 Class diagram of NLIDB converter for Customer Relationship Index

6.4.2 Implementation of the Data Access Component

The data access component consists mainly of the NLIDBDAC class which contains all the functions and methods for reading data from and writing data to the NLIDB database and the entity model which contains the conceptual and storage model of the NLIDB database and the mapping between the conceptual and storage model.

Data access component is interconnected with the business layer which manipulates the business logic before modifications to the NLIDB database is done.

6.4.2.1 Entity Framework and LINQ

When accessing the database through the entity framework all the tables and views in the database are treated as classes of the respective table or view in the NLIDB database. These classes enforce the datatypes and any constrains built in to the NLIDB database automatically maintaining the data integrity of the underlying database.

CRUD operations and data extraction is implemented using LINQ queries, which performs the required insert, update, delete or select operation in the underlying NLIDB database.

The figure below shows the variables and methods implemented in the data access component for CRUD operations and the extraction of data in the underlying NLIDB database.

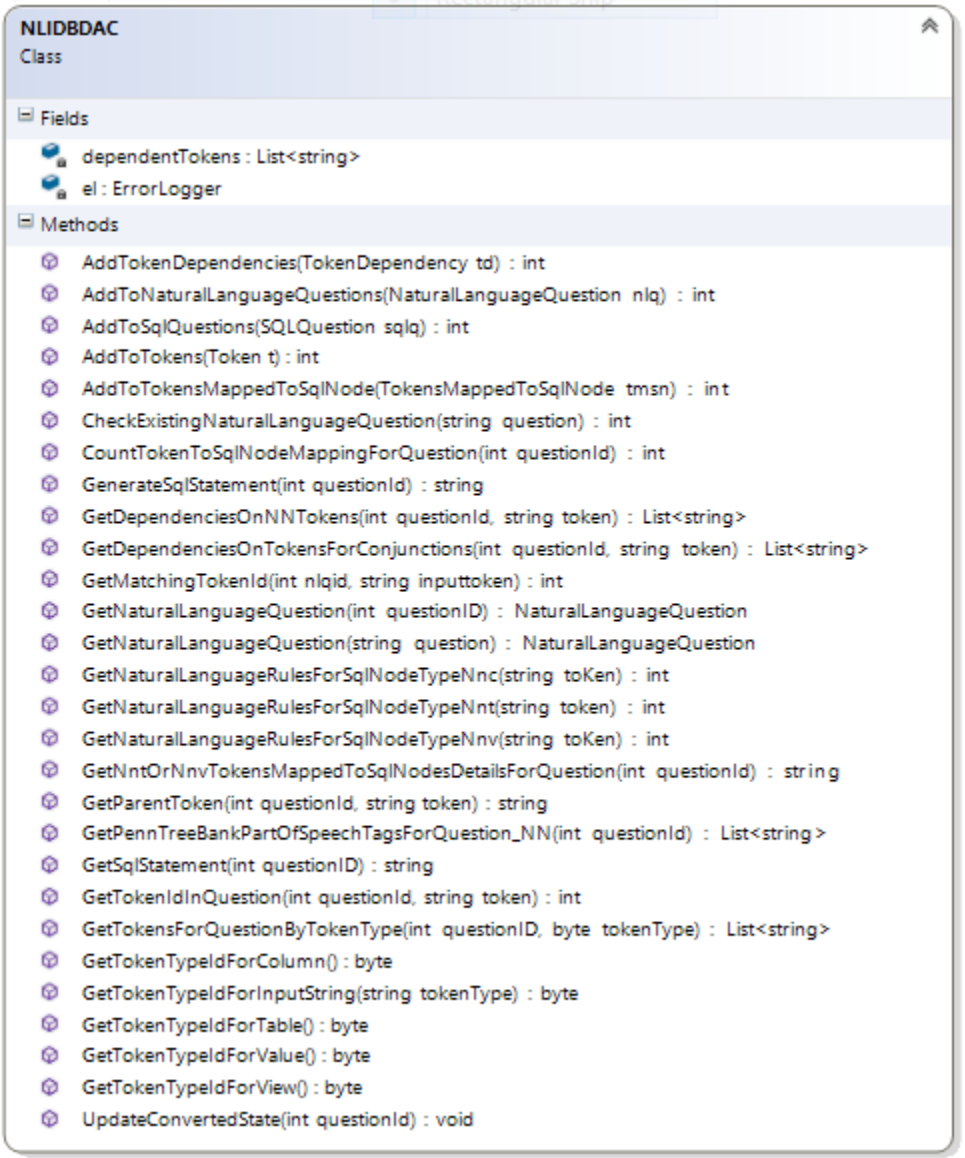


Figure 6.4 Variables and Methods in NLIDBDAC

6.4.3 Implementation of the Business Logic Module

The business logic module handles the core functionality of the NLIDB converter for Customer Relationship Index. Its main responsibility is to convert the input in natural language to its corresponding T-SQL statement. This module accomplishes it by

integrating the functionality of several sub modules inside the business logic module as well as by working with several other main components. The following sub modules discussed in detail in the forthcoming sections, collectively make up the business logic module.

6.4.3.1 NLIDB Converter

The source code for the NLIDB converter is located at the class Program (Appendix A). This is the most significant segment of code in the application. The class program calls the functions in the NLIDBBAL and NLIDBLogger classes as well as the linguistic component to manipulate the input string in to the desired T-SQL output.

The algorithm for converting an input string in natural language to its corresponding T-SQL statement is listed below.

Step 1 Prompt user to type the question in natural language

Step 2 If (question != null) -> go to next step

Else -> display error message - empty input string

Step 3 If question in natural language already exist in the database -> go to Step 4

Else -> go to Step 5

Step 4 If question is already converted to T-SQL -> show corresponding T-SQL statement

Else -> display warning message - please rephrase the question and try again as it was not converted to T-SQL

Step 5 Save new question in database

Step 6 Extract the sentences in the input

Step 7 Extract tokens in the sentences

Step 8 Foreach token

Get the token text and part of speech tag and save to database

Step 9 Extract token dependencies

Step 10 Foreach dependency

Save dependencies to database as parent and child token

Step 11 Extract nouns in the tokens (Nouns are identified based on the token type assigned by the Penn Treebank tags)

Step 12 Foreach noun

check them against the lexicon to identify tables or views and get the rule symbol which corresponds to the table/view name

Save table to database (this is a token mapped with a SQL node)

Step 13 Extract tokens depending on the selected noun token

Step 14 Foreach dependent token

check them against the lexicon to identify columns and get the rule symbol which corresponds to the column of the selected table

Save columns to database (this is a token mapped with a SQL node)

Step 15 Generate T-SQL statement based on tokens mapped to SQL nodes

The workflow representing the algorithm is displayed next. In this workflow swim lanes are introduced. Each swim lane shows processes completed by a given specific role in the system. For the NLIDB converter for Customer Relationship Index there are two roles, one is the NLIDB converter and the other one is the Customer Relationship Index user.

The figure on the next page illustrates the workflow in which the NLIDB converter for Customer Relationship Index converts the input in to the desired output.

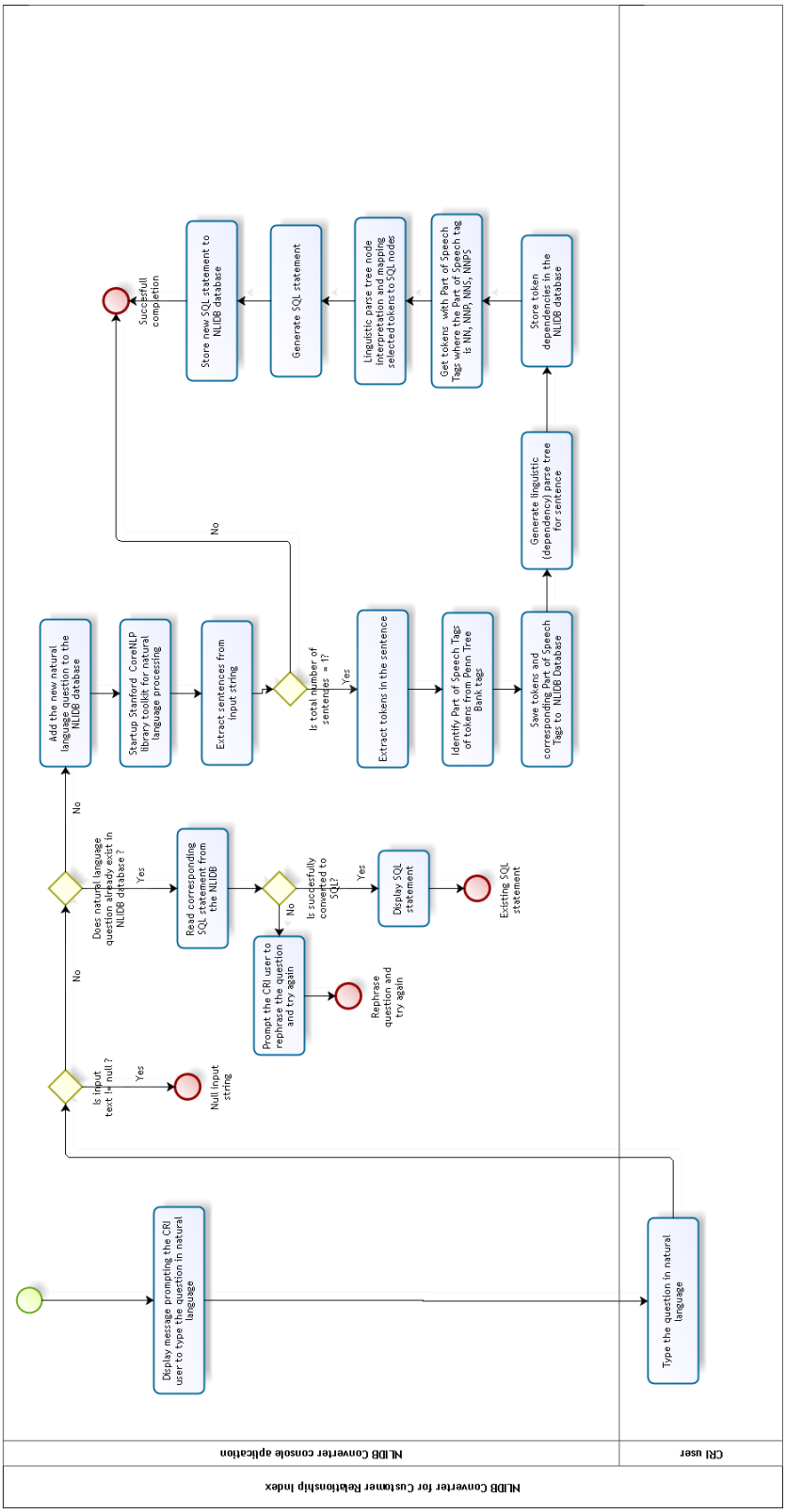


Figure 6.5 Flow chart - NLIDB converter for Customer Relationship Index

The pseudo code for the NLIDB converter is as follows,

```
DECLARE String Text
DECLARE String nnTokens[]
DECLARE Integer ruleIdNnt     = 0
DECLARE Integer ruleIdNnv     = 0
DECLARE Integer ruleIdNnc     = 0
DECLARE String dependencies   = NULL
DECLARE Integer tokenId       = 0
DECLARE Integer nnTokenId     = 0
DECLARE Integer parentTokenId = 0
DECLARE byte tokenTypeId      = 0
DECLARE int dependencyId      = 0
DECLARE sentencesCount        = 0

SET Text to READ Input String

IF Text ≠ NULL THEN
   IF Text exists in NLIDB
      IF Text is successfully converted to SQL Statement
         DISPLAY SQL Statement
      ELSE
         DISPLAY error message ‘Please rephrase and try again’
      END IF
   ELSE
      INSERT Text to NLIDB
      Startup Stanford CoreNLP library

      DECLARE String sentences []
      SET sentences [] to Extract sentences from Text

      IF sentences count = 1
         Extract tokens from sentences [1]

         FOR each token in tokens
            Identify part of speech tags of token from Penn
            tree bank tags
```

```
   INSERT token and part of speech tag to NLIDB
END LOOP

SET dependencies to Extract token dependencies in
sentence [1] using Stanford parser

DECALRE String stringSeparators[] ={‘)’}
DECLARE String dependencyList []

SET dependencyList [] to SPLIT dependency from
stringSeparators []

FOR each dependency in dependencies
   DECLARE String innerStringSeperators []
     = { ‘(’  ,  ’,’  , ’ -’ }
   DECLARE String tokenDependencies []

   SET tokenDependencies [] to SPLIT dependency
   from innerStringSeperators []

   IF tokenDependencies [1] = ’Root’
      SET parentTokenId to 0
      SET tokenId to tokenDependencies [3]
   ELSE
      SET parentTokenId to tokenDependencies [1]
      SET tokenId to tokenDependencies [3]
   END IF

   INSERT parentTokenId and tokenId to NLIDB
END LOOP

SET nnTokens [] to GET the tokens based on the Penn
tree bank tags and match them with SQL Node types

FOR each nnToken in nnTokens
   SET ruleIdNnt to GET the natural language
   rule id for name node table
```

```
   IF ruleIdNnt ≠ 0
      INSERT nnToken and ruleNnt to NLIDB
   END IF

   DECLARE Integer ruleIdNnv

   SET ruleIdNnv to GET the natural language rule id
   for name node view

   IF ruleIdNnv ≠ 0
      INSERT nnToken and ruleNnv to NLIDB
   END IF
END LOOP

DECLARE Integer countOfTokensMappedToSqlNodes

SET countOfTokensMappedToSqlNodes to GET the count
of tokens mapped to natural language rules

IF countOfTokensMappedToSqlNodes = 1
   DECLARE String token
   SET token to GET the NNT or NNV token

   DECLARE String dependentTokens []
   SET dependentTokens [] to GET the tokens depending
   on the NNT or NNV token

   FOR each dependentToken in dependentTokens
      SET ruleIdNnc to GET the natural language rule
      for name node view

      IF ruleIdNnc ≠ 0
         SET nnTokenId to GET Token ID by dependentToken
         INSERT ruledNnc and nnTokenId to NLIDB
      END IF
   END LOOP
```

```
            DECLARE String sqlStatement
            SET sqlStatement to Generate the SQL statement
            for the question in natural language

            INSERT sqlStatement to NLIDB
            DISPLAY sqlStatement
         ELSE IF countOfTokensMappedToSqlNodes = 0
            DISPLAY error message 'Natural rules not found'
         ELSE
            DISPLAY error message 'Not implemented for multiple
            tables or views'
         END IF
      ELSE
         DISPLAY error message 'Number of sentences
         is greater than 1'
      END IF
   END IF
ELSE
   DISPLAY error message 'Text is Empty'
END IF
```

The important source code segments for the NLIDB converter is explained in detail in the forthcoming section;

The required libraries to run the program are imported in the top section of the code through using statements. The significant libraries to note are the edu.stanford.nlp. * libraries.

The figure below shows the library imports for running the NLIDB converter program in the Program class.

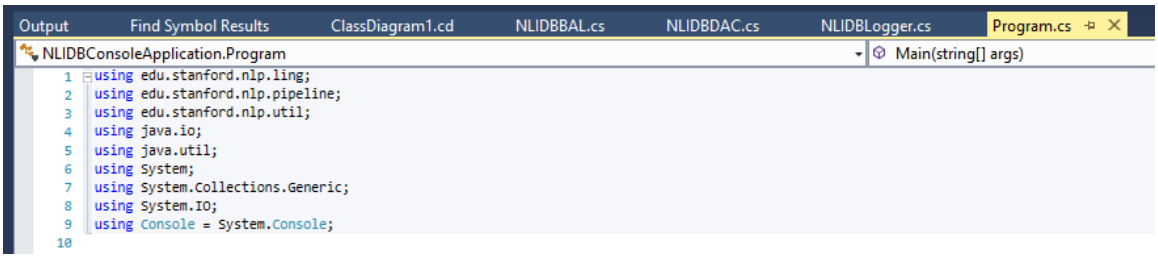


Figure 6.6 NLIDB Converter - Code segments - library imports

A message is displayed to the user asking him or her to type the question in natural language to which he or she responds with a question to be run against the Customer Relationship Index database.

The question in natural language is accepted from the user and is checked against the NLIDB database to see if it already exists in the system. If it's already there in the database and it is properly converted to T-SQL, then the T-SQL statement is extracted and displayed and if not, a warning message is displayed to the user saying that this particular question has been asked previously but it was not converted to T-SQL asking him or her to retry and try again.

The figure below shows the code segment for handling existing questions in the system.

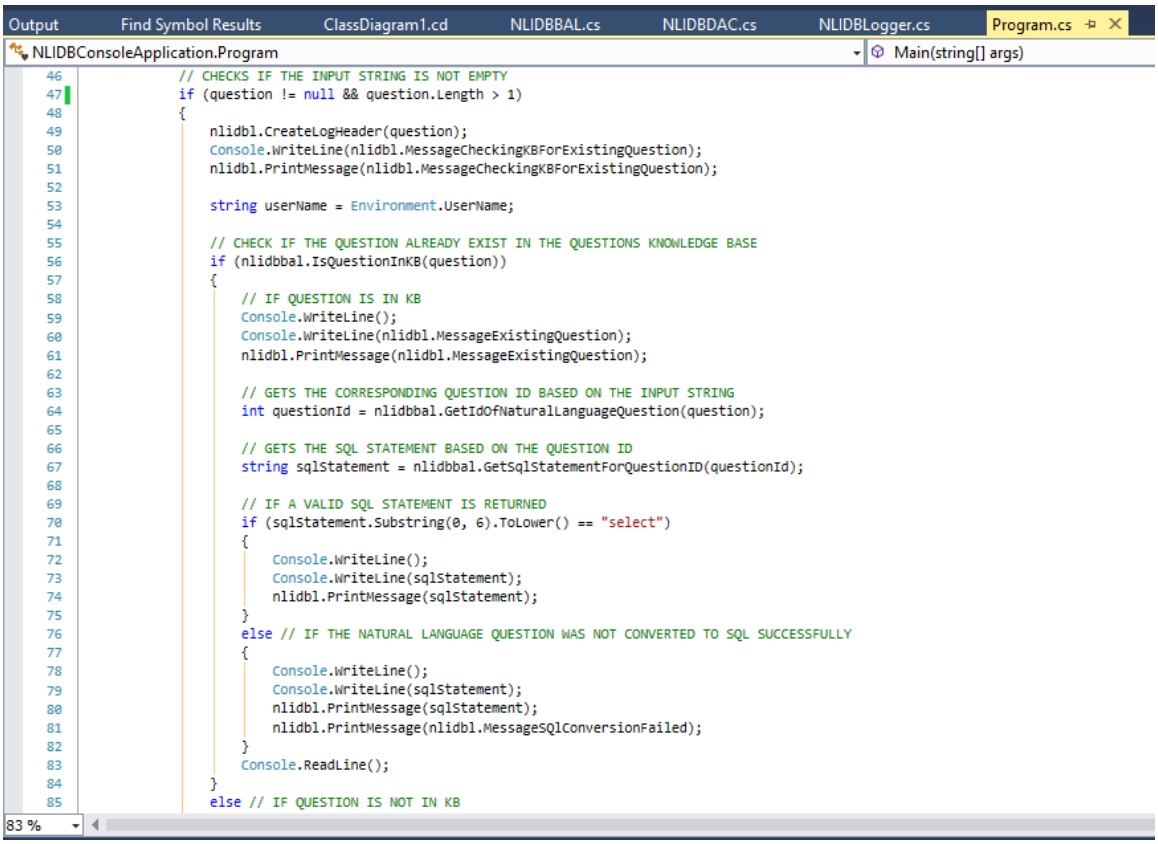


Figure 6.7 NLIDB Converter - Code segments - handling existing questions

If the natural language question asked by the user is not in the database, then the new question is saved to the database before further processing begins. In order to start the natural language processing and meaning extraction a Stanford CoreNLP object is constructed from a given set of properties under annotators. This method creates and configures the pipeline using the annotators given in the “annotators” property.

The figure below shows the code segment for configuring the Stanford CoreNLP toolkit for natural language processing and meaning extraction.

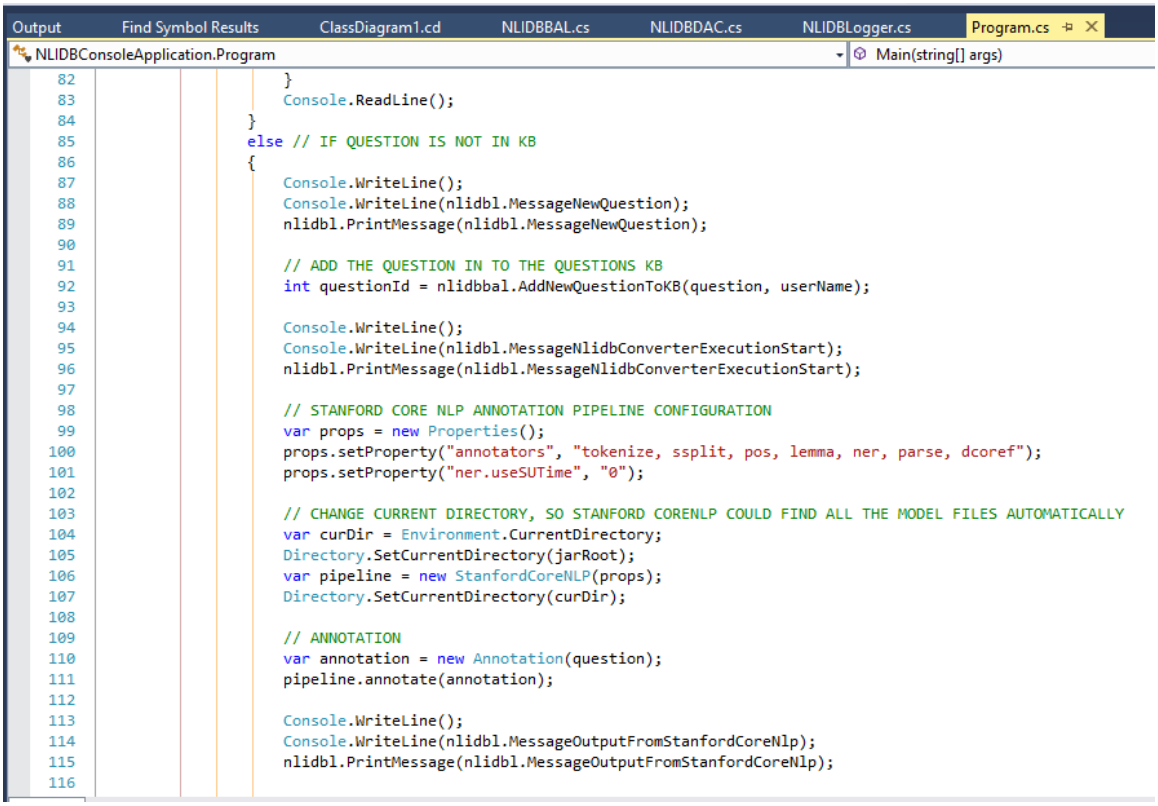


Figure 6.8 NLIDB Converter - Code segments - handling new questions and configuring Stanford CoreNLP

Once the pipeline successfully runs and the variable annotation in line 110 is assigned values through the CoreAnnotations.SentencesAnnotation().getClass() the system is able to extract the distinct sentences in the input question. Once the sentences are identified the program checks if its equal to the amount of allowed sentences and if the condition is met then the token details are extracted through CoreAnnotations.TokensAnnotation().getClass(). For each token extracted for the sentence then the token properties are extracted. Token properties consist of the token text extracted from CoreAnnotations.TextAnnotation().getClass() and the token type based on the Penn Treebank tag set extracted from CoreAnnotations.PartOfSpeechAnnotation().getClass().

The figure below shows the code segment for extracting sentences and the tokens in each of the sentences. However, since the maximum number of sentences is restricted to one, this loop for sentences runs only once.

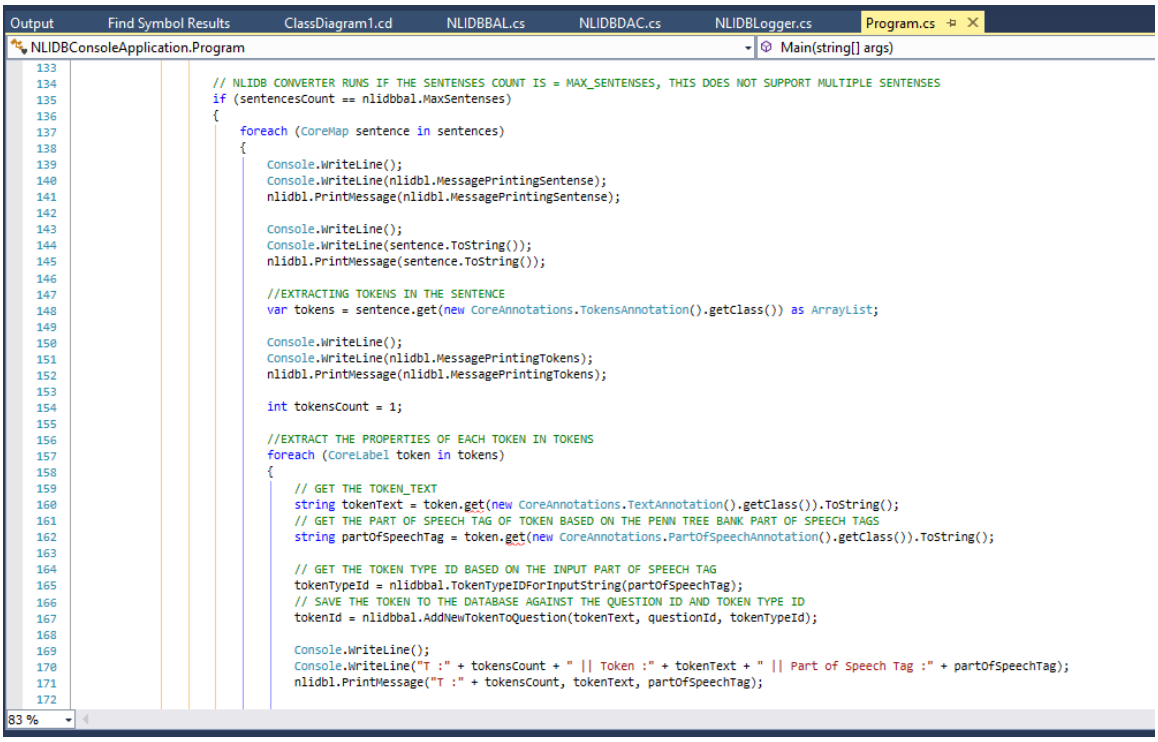


Figure 6.9 NLIDB Converter - Code segments - Extracting token details

For all tokens once the token details and the type of the token is saved to the database, the system starts to identify the dependencies of tokens in the input sentence.

Identification of dependencies between tokens is done to see if there are tokens that further describe a given particular token. The reason for this is that if a token is matched in the lexicon for a table name through SQL node mapping then the dependent tokens are matched in the lexicon for the column names in order to extract the column names for the table so that the T-SQL statement can be extracted accordingly.

The dependency extraction is done by the Stanford Parser. For this program, the dependencies known as Stanford dependencies are used. The Stanford dependencies provides a representation of grammatical relations between words in a sentence. The Stanford dependencies (SD) are triplets: name of the relation, governor and dependent.

For example, for the natural language question, “Show me the name of customers” the Stanford dependencies are extracted in the following format.

root(ROOT-0, Show-1)dep(Show-1, me-2)det(name-4, the-3)dep(Show-1, name-4)case(customers-6, of-5)nmod:of(name-4, customers-6)

This code segment shows how the dependencies are being extracted in to a usable form by the NLIDB converter for processing them in the next section.

The figure below shows the code segment for extracting and saving the dependencies between the tokens.

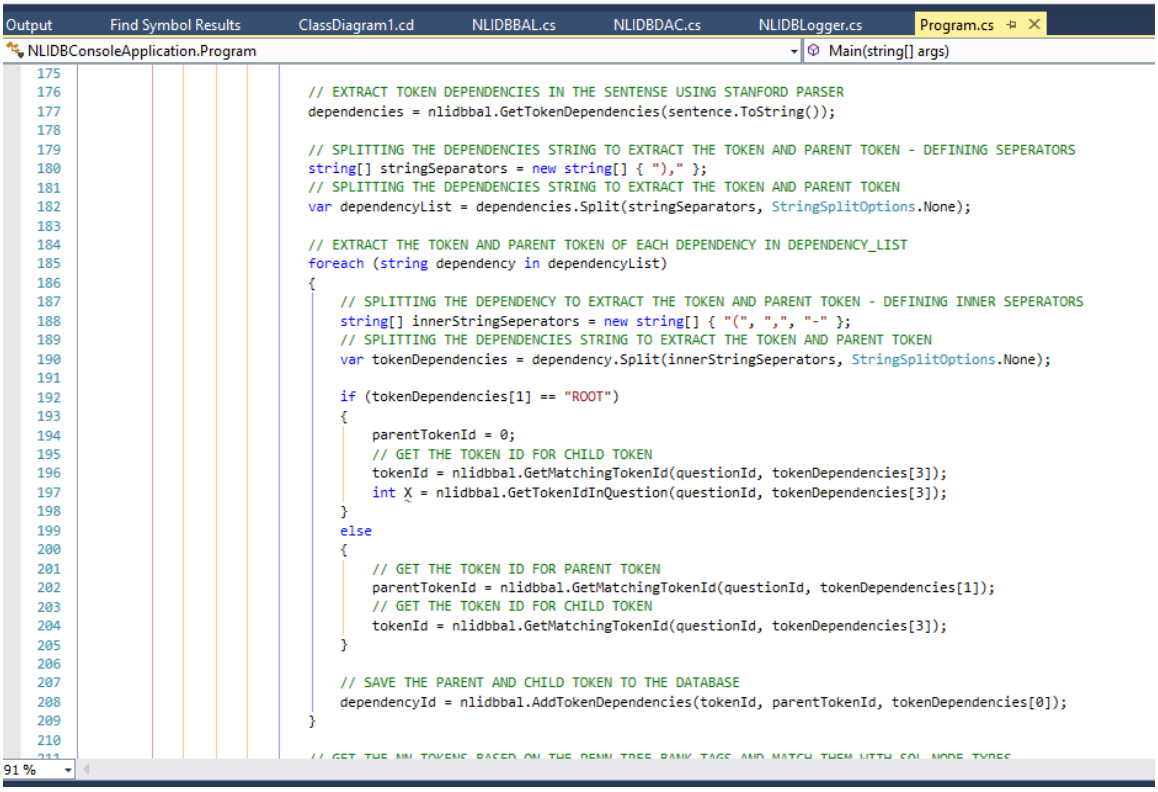


Figure 6.10 NLIDB Converter - Code segments - Extracting token details

According to the Penn Treebank tags for the tokens identified in the input string all tokens tagged with the token type NN% is extracted. (The tag types starting from NN are nouns in the Penn Treebank tags). These NN% tokens are highly likely to correspond to the tables, views or column names in the customer relationship index database. Therefore, the tokens

with the NN% type is extracted and checked against the lexicon to see if there are natural language rules to match them with. If such match is found for the rules mapped for the SQL node type NNT then the token is categorized as a table token and mapped with the actual name of the table. This is also true for views too.

Then the tokens mapped to tables and views are saved to the NLIDB database for later querying.

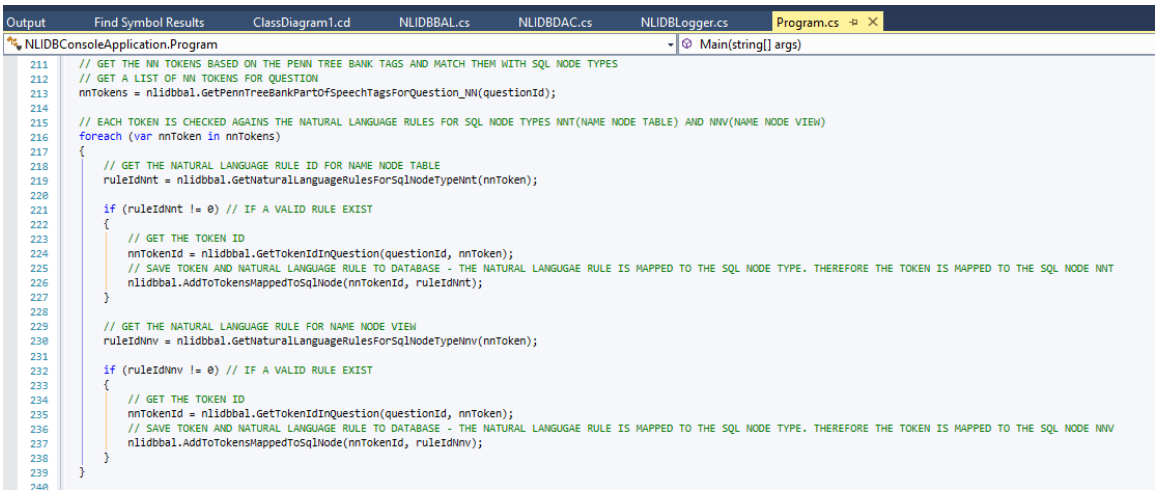


Figure 6.11 NLIDB Converter - Code segments - SQL node mapping for Tables and Views

Now the tokens mapped to the SQL nodes are extracted, since the system supports natural language to T-SQL conversion for only one table or view currently, the system checks if the count of nodes mapped to SQL nodes are equal to 1. If so, then the system extracts the dependent tokens for the initial token which was mapped with either a SQL table node (NNT) or a view node (NNV).

These dependent tokens are again run against the lexicon to find any natural language rules to match with the column names in the database. If such match is found for the rules mapped for the SQL node type NNC then the token is categorized as a column token and mapped with the actual name of the column.

Then the tokens mapped to the column names are saved to the NLIDB for later querying.

Now the system has a set of tables or view and quite possibly a set of column names mapped against SQL nodes. Depending on the nodes identified and using the actual name of the SQL element the system generates the T-SQL statement and displays it to the user.

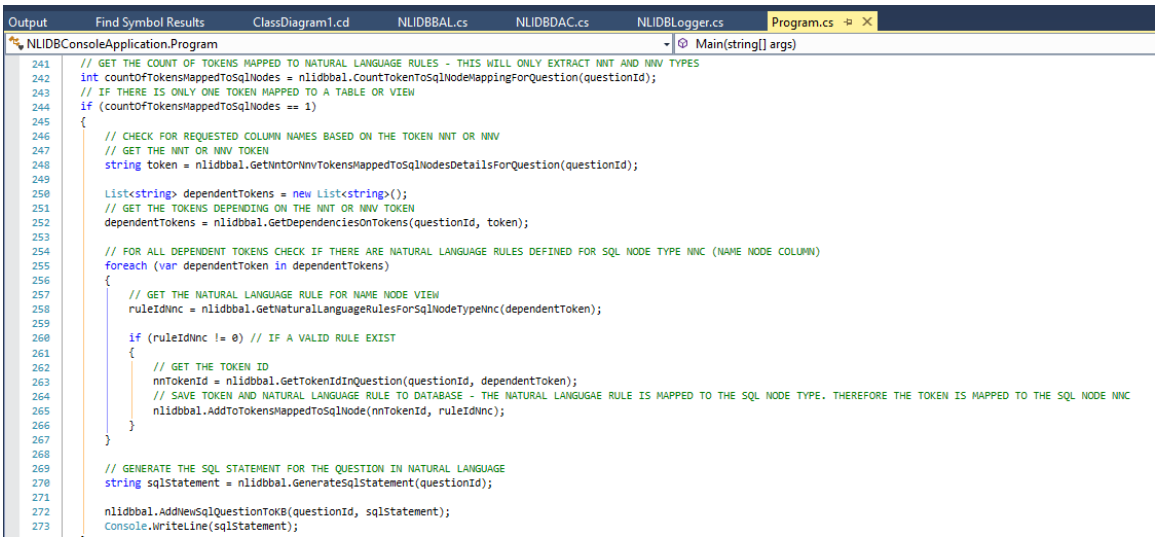


Figure 6.12 NLIDB Converter - Code segments - SQL node mapping for Columns and T-SQL statement generation

The following code segment shows how the SQL statement is generated based on the tokens mapped against the SQL node types.

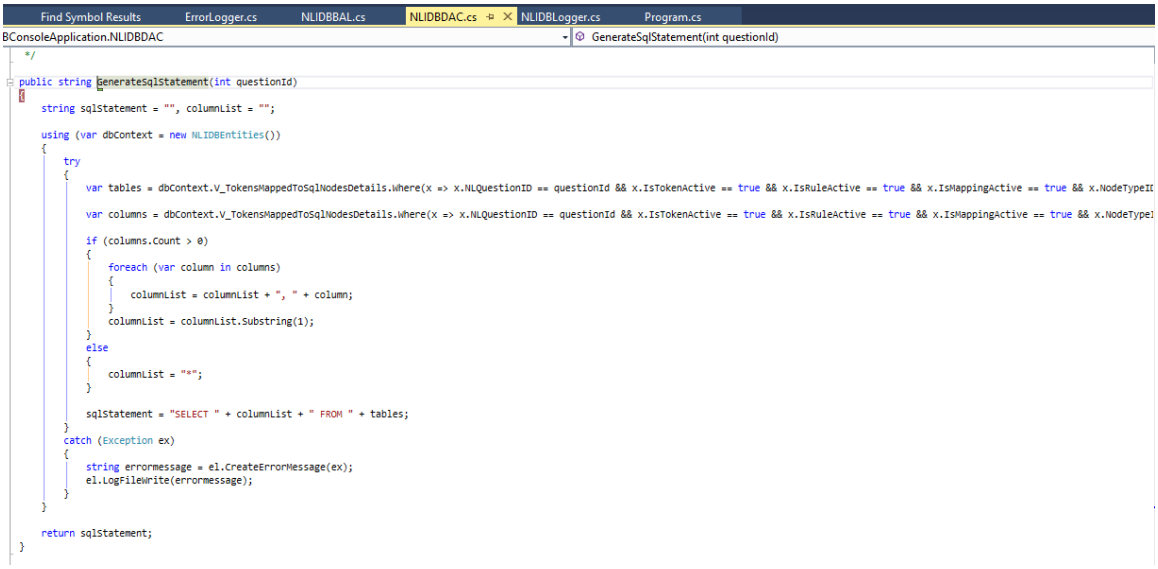


Figure 6.13 NLIDB Converter - Code segments - Generate SQL statement

6.4.3.2 **Business Layer**

Business layer consists of the class NLIDBBAL. The main functionality of this class is to accept the inputs from the class Program which runs the NLIDB converter program and apply the business rules defined in this class or the methods in this class to the input parameters and then pass it on to the NLIDBDAC class which is responsible for executing the relevant CRUD or select operation against the NLIDB database.

The business later creates an object of the NLIDBDAC and calls the methods in the NLIDBDAC through this object.

This class contains the control variables which is used in the Program class. E.g.: the maximum number of allowed sentences, the notification messages to be displayed to the user during the execution of the NLIDB converter etc.

The figure below shows the variables and methods in the NLIDBBAL for passing data to and fro from the data access layer to the presentation layer.

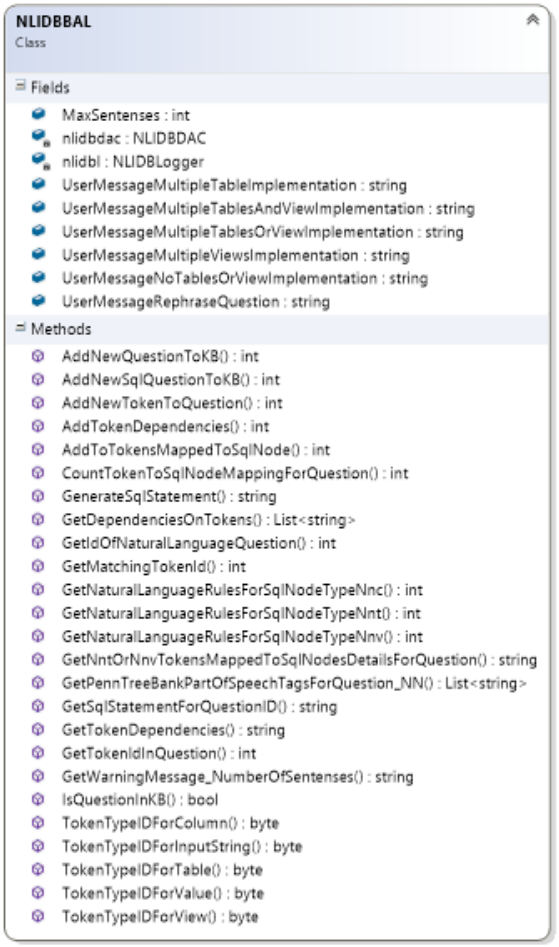


Figure 6.14 Variables and Methods in the NLIDBBAL

6.4.3.3 **NLIDB Logger**

The NLIDB logger is a class that defines a set of methods and functions to log all activities taking place in the NLIDB converter for Customer Relationship Index. The methods in this class uses the StringBuilder to create the message body and uses the FileStream and StreamWriter object for writing the log in to a note pad file in the specified location.

While implementing this class object oriented concepts such as polymorphism is used.

The image below shows the variables and methods in the NLIDBLogger class.

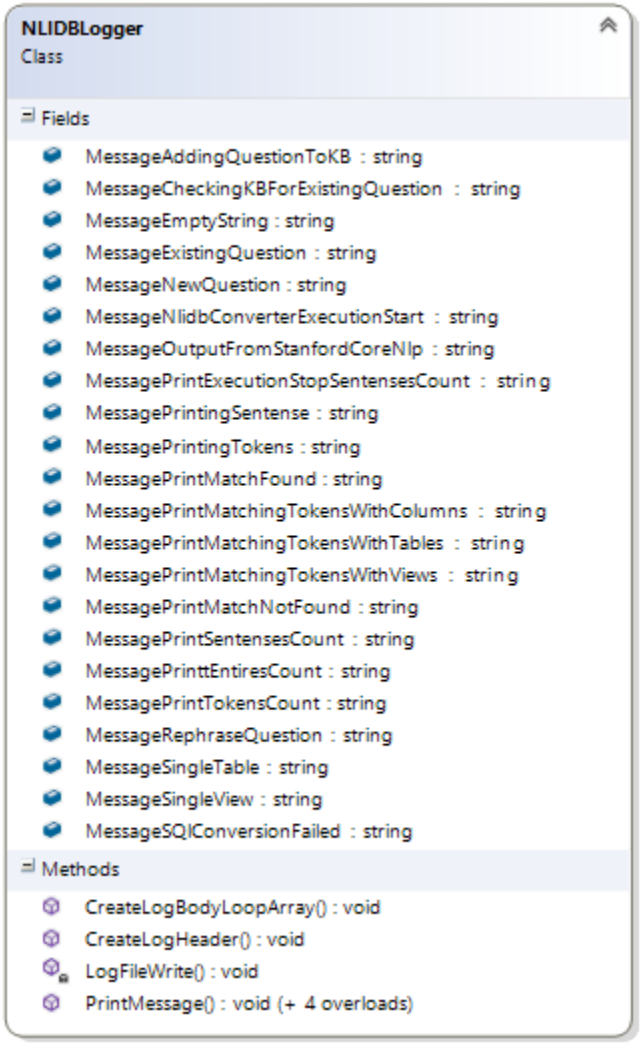


Figure 6.15 Variables and Methods in the NLIDBLogger

6.4.3.4 **Error Logger**

The Error logger class contains the functions and methods for logging all exceptions thrown during application execution. The methods in this class uses the StringBuilder to create the message body and the FileStream and StreamWriter object for writing the log in to a note pad file in the specified location.

The image below shows the variables and methods in the ErrorLogger class.

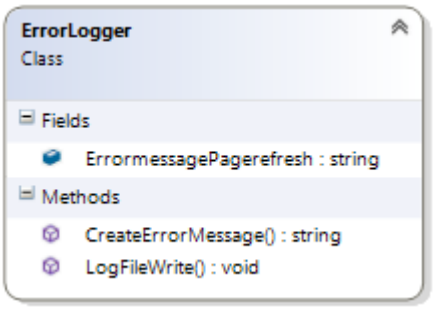


Figure 6.16 Variables and Methods in ErrorLogger

6.4.4 Implementation of the Linguistic Component

The linguistic component is made up of the Stanford CoreNLP and the Stanford Parser for .NET toolkit libraries. These two toolkit libraries are added to the project through the NuGet package manager.

To add the Stanford CoreNLP the following command must be run in the package manager console in the Visual Studio 2013

```
PM> Install-Package Stanford.NLP.CoreNLP
```

Figure 6.17 Adding the Stanford CoreNLP to the project

To add the Stanford Parser, the following command must be run in the package manager console in the Visual Studio 2013

```
PM> Install-Package Stanford.NLP.Parser
```

Figure 6.18Adding the Stanford Parser to the project

After running these two commands in the package manager console the DLL files needed to run the toolkits are installed and added to the project.

Next the models for both the Stanford CoreNLP and Stanford Parser must be downloaded and stored inside the projects root folder. Both Stanford CoreNLP and Stanford Parser accesses this model files when text annotation happens.

6.4.4.1 **Using Stanford CoreNLP toolkit libraries**

The original code provided by the Stanford team is modified to extract the text annotations in a format that is usable by the NLIDB converter for Customer Relationship Index to get the text properties like the image below.

The image below shows how the path for the model files are set in the code.

```
// PATH TO THE FOLDER WITH MODELS EXTRACTED FROM `stanford-corenlp-3.7.0-models.jar`
var jarRoot = @"H:\MSCIT-14-069\RESEARCH\NLIDBConsoleApplication\stanford-corenlp-full-2016-10-31\stanford-corenlp-3.7.0-models";
```

Figure 6.19 Code segment - Setting up the path for the Stanford CoreNLP models

The image below shows the code segment for configuring the annotators (Appendix B). The annotators used for this scenario are tokenize, ssplit, pos, lemma, ner, parse and decoref.

```
97
98              // STANFORD CORE NLP ANNOTATION PIPELINE CONFIGURATION
99              var props = new Properties();
100             props.setProperty("annotators", "tokenize, ssplit, pos, lemma, ner, parse, dcoref");
101             props.setProperty("ner.useSUTime", "0");
102
103             // CHANGE CURRENT DIRECTORY, SO STANFORD CORENLP COULD FIND ALL THE MODEL FILES AUTOMATICALLY
104             var curDir = Environment.CurrentDirectory;
105             Directory.SetCurrentDirectory(jarRoot);
106             var pipeline = new StanfordCoreNLP(props);
107             Directory.SetCurrentDirectory(curDir);
108
109             // ANNOTATION
110             var annotation = new Annotation(question);
111             pipeline.annotate(annotation);
112
113             Console.WriteLine();
114             Console.WriteLine(nlidbl.MessageOutputFromStanfordCoreNlp);
115             nlidbl.PrintMessage(nlidbl.MessageOutputFromStanfordCoreNlp);
116
117             // RESULT - PRETTY PRINT
118             using (var stream = new ByteArrayOutputStream())
119             {
120                 pipeline.prettyPrint(annotation, new PrintWriter(stream));
121                 Console.WriteLine();
122                 Console.WriteLine(stream.toString());
123                 nlidbl.PrintMessage(stream.toString());
124                 stream.close();
125                 Console.WriteLine("=============================================================");
126             }
```

Figure 6.20 Code segment - Configuring annotators in Stanford CoreNLP

The code segment below shows how the variable annotation splits the sentence in to sentences and assign the sentences in to an array list.

```
127
128             //EXTRACTING SENTENCES FROM THE INPUT STRRING
129             ArrayList sentences = annotation.get(new CoreAnnotations.SentencesAnnotation().getClass()) as ArrayList;
130
```

Figure 6.21 Code segment - Sentence splitting in Stanford CoreNLP

The code segment below shows how the variable sentence (this is one element in the array list sentences) is split to tokens.

```
147                 //EXTRACTING TOKENS IN THE SENTENCE
148                 var tokens = sentence.get(new CoreAnnotations.TokensAnnotation().getClass()) as ArrayList;
149
```

Figure 6.22 Code segment - Tokenizing in Stanford CoreNLP

The code segment below shows how the token properties are extracted.

```
159                     // GET THE TOKEN_TEXT
160                     string tokenText = token.get(new CoreAnnotations.TextAnnotation().getClass()).ToString();
161                     // GET THE PART OF SPEECH TAG OF TOKEN BASED ON THE PENN TREE BANK PART OF SPEECH TAGS
162                     string partOfSpeechTag = token.get(new CoreAnnotations.PartOfSpeechAnnotation().getClass()).ToString();
```

Figure 6.23 Code segment - Extracting token properties in Stanford CoreNLP

6.4.4.2 **Using the Stanford Parser for .NET**

The original code provided by the Stanford team is modified to extract the token dependencies within a sentence in a format that is usable by the NLIDB converter for Customer Relationship Index. The code segment below shows how the dependencies are extracted from the Stanford Parser for .NET.

```
public string GetTokenDependencies(string sentence)
{
    string dependencies = null;

    // PATH TO MODELS EXTRACTED FROM `stanford-parser-3.7.0-models.jar`
    var jarRoot = @"H:\MSCIT-14-069\RESEARCH\NLIDBConsoleApplication\stanford-corenlp-full-2016-10-31\stanford-corenlp-3.7.0-models\";
    var modelsDirectory = jarRoot + @"\edu\stanford\nlp\models";

    // LOADING ENGLISH PCFG PARSER FROM FILE
    var lp = LexicalizedParser.loadModel(modelsDirectory + @"\lexparser\englishPCFG.ser.gz");

    // THIS OPTION SHOWS LOADING AND USING AN EXPLICIT TOKENIZER
    var tokenizerFactory = PTBTokenizer.factory(new CoreLabelTokenFactory(), "");
    var sentenceReader = new StringReader(sentence);
    var words = tokenizerFactory.getTokenizer(sentenceReader).tokenize();
    sentenceReader.close();
    var tree = lp.apply(words);

    // EXTRACT DEPENDENCIES FROM LEXICAL TREE
    var tlp = new PennTreebankLanguagePack();
    var gsf = tlp.grammaticalStructureFactory();
    var gs = gsf.newGrammaticalStructure(tree);
    var tdl = gs.typedDependenciesCollapsed();

    System.Console.WriteLine("\n{0}\n", tdl);
    nlidbl.PrintMessage(tdl.ToString());

    // EXTRACT COLLAPSED DEPENDENCIES FROM PARSED TREE
    var tp = new TreePrint("penn,typedDependenciesCollapsed");
    tp.printTree(tree);
    nlidbl.PrintMessage(tree.toString());

    dependencies = tdl.ToString();

    return dependencies;
}
```

Figure 6.24 Code Segment - Extracting dependencies in Stanford Parser

These two implementations make up the linguistic component of the NLIDB converter for Customer Relationship Index. The code is placed in the Program class and the NLIDBBAL class.

6.4.5 Implementation of the User Interfaces

The user interface to the NLIDB converter for Customer Relationship Index is the windows console. Messages are displayed and input is collected from the user through the windows console.

6.4.5.1 Displaying messages

The code segment for displaying messages to the user is,

```
Console.WriteLine(Message_To_Be_Displayed);
```

6.4.5.2 Collecting input

The code segment for collecting input from the user is, in this case the question in natural language.

```
var question = Console.ReadLine().Trim();
```

6.4.6 Assumptions and Constraints during Implementation

6.4.6.1 Assumptions

When naming the tables, views and column names the software team who developed the Customer Relationship Index used nouns so that its sufficient to check the lexicon for rules defined under tables, views and columns for only NN% typed tokens.

6.4.6.2 Constraints

1. The NLIDB converter for Customer Relationship Index can convert natural language question to its corresponding T-SQL when there is only one token that has been mapped to a table or view. If there are multiple token to SQL node matches for tables or views the system stops execution and displays a warning message.
2. The system only generates T-SQL statements in "SELECT ------ FROM -------" form.
3. The conversion from natural language to the corresponding T-SQL form depends on the number of rules defined in the lexicon under each category.

6.5 Summary

The current chapter discusses the implementation of the NLIDB converter for Customer Relationship Index. The implementation of each component is discussed in detail providing algorithms, flow charts and code segments. Next chapter discusses the performance of NLIDB converter for Customer Relationship Index when it is put through the initial evaluation.

Chapter 7

7 NLIDB converter for Customer Relationship in Practice

7.1 Introduction

The previous chapter presented the details of implementation of the NLIDB converter for Customer Relationship Index by presenting the algorithms, pseudo code and relevant code segments for the benefit of the reader. This chapter presents the reader details of how the NLIDB converter for Customer Relationship Index performed when it was put through the initial evaluation with the software team of Brandix Apparel Solution LTD – Essentials. Furthermore, this chapter discusses inputs, outputs, results and the process to arrive at the outputs in detail for test scenarios used by the software team during their evaluation of the system.

7.2 Initial Evaluation of the NLIDB converter for Customer Relationship Index

After its implementation the NLIDB converter for Customer Relationship Index was planned to be released to the users of the system for them evaluate it in order to identify if there are any critical issues in the system before the system is subjected to a formal evaluation. Out of the two categories of users defined for this system under the Approach section in Chapter 4, first, the NLIDB converter for Customer Relationship Index was released to the software team of Brandix Apparel Solution LTD – Essentials as they are the proprietary owners of the Customer Relationship Index web application. Since this team designed the schema of the Customer Relationship Index database, they are the most suitable party to test and review the NLIDB converter for Customer Relationship Index initially.

The NLIDB converter for Customer Relationship Index was installed on each of the three software engineers who worked in the Customer Relationship Index system and were asked to type in questions in natural language (English) to see if the system generated the expected results. The software engineers team has review the system for two weeks after

it was installed in their laptops and no critical issue was identified that resulted in a system failure or seizure.

For the benefit of the reader and to give insight as to how the NLIDB converter for Customer Relationship Index works, a few scenarios tested by the software team of Brandix Apparel Solution LTD – Essentials have been included in this chapter with screenshots of the input, output and the process in between.

Before the scenarios can be discussed, a sound understanding of the schema of the table that the scenarios were built around is needed by the reader to fully comprehend the matter. Hence, the next section presents the schema of the table in the Customer Relationship Index database that was used for the test scenarios included in this document, allowing the reader to understand the scenarios better and to verify that the responses returned by the NLIDB converter for Customer Relationship Index is accurate.

The figure below shows the schema of the table containing the details of the customer in the Customer Relationship Index database.

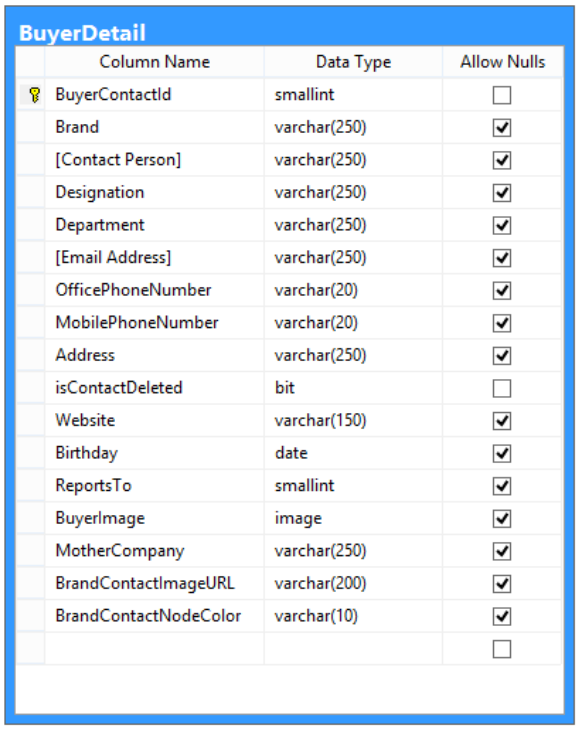
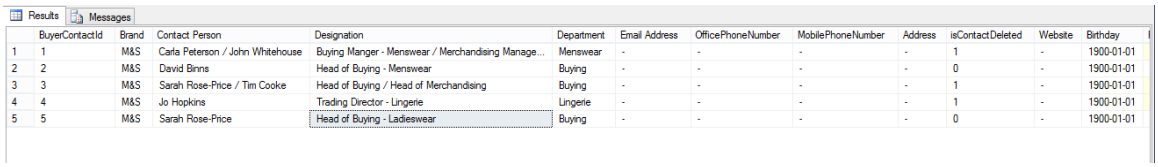


Figure 7.1 Schema of the table containing customer details

The figure below shows the data that the BuyerDetail table holds.



Results | Messages

	BuyerContactId	Brand	Contact Person	Designation	Department	Email Address	OfficePhoneNumber	MobilePhoneNumber	Address	isContactDeleted	Website	Birthday
1	1	M&S	Carla Peterson / John Whitehouse	Buying Manger - Menswear / Merchandising Manage...	Menswear	-	-	-	-	1	-	1900-01-01
2	2	M&S	David Binns	Head of Buying - Menswear	Buying	-	-	-	-	0	-	1900-01-01
3	3	M&S	Sarah Rose-Price / Tim Cooke	Head of Buying / Head of Merchandising	Buying	-	-	-	-	1	-	1900-01-01
4	4	M&S	Jo Hopkins	Trading Director - Lingerie	Lingerie	-	-	-	-	1	-	1900-01-01
5	5	M&S	Sarah Rose-Price	Head of Buying - Ladieswear	Buying	-	-	-	-	0	-	1900-01-01

Figure 7.2 Screenshot of the details in the table BuyerDetail

The following table explains each column and the data it holds in the BuyerDetail table.

Column Name	Column Description
BuyerContactId	The auto generated Id of each contact
Brand	The name of the customer brand; e.g. VS, M&S, CK etc.
Contact Person	The name of the contact person
Designation	The designation of the contact person
Department	The department in which the contact person is from
Email Address	The email address of the contact person
OfficePhoneNumber	The land line number
MobilePhoneNumber	The mobile phone number
Address	Address where the contact person is located at
isContactDeleted	If the contact person is currently active in the system or not
Website	The website of the contact person if he/she has one
Birthday	The birthday of the contact person
ReportsTo	The auto generated Id of the manager
BuyerImage	The binaries of the image for the contact person
MotherCompany	The mother company of the contact person
BrandContactImageURL	The location of the contact person's picture in the HD of the server

BrandContactNodeColor	The color in hexadecimal to be applied in the organization structure for the contact person

Table 5 Schema explanation of BuyerDetail

7.3 Scenario 1

7.3.1 Scenario Description

This scenario tests how the application responds when a user input is submitted that contains matching natural language rules in the lexicon for both the table and the columns allowing successful token to SQL node matching for both table and columns.

7.3.2 Input String

Show me the name and designation of Customers

The figure below shows the input submitted to the NLIDB converter for Customer Relationship Index for scenario 1.

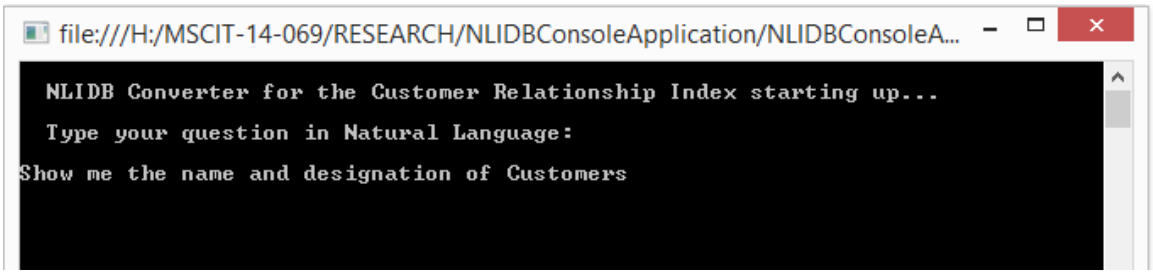


Figure 7.3 Initial Evaluation – Scenario 1 – Inputs

7.3.3 Arriving at the output

1. Using the Penn Treebank tags the tokens in the input string is categorized.

Token ID	Token	Token Type	Token Description
1	Show	NN	noun, common, singular or mass
2	me	PRP	pronoun, personal
3	the	DT	determiner
4	name	NN	noun, common, singular or mass
5	and	CC	conjunction, coordinating

6	designation	NN	noun, common, singular or mass
7	of	IN	preposition or conjunction, subordinating
8	Customers	NNS	noun, common, plural

Table 6 Tokens in Scenario 1

2. The NN% typed tokens are checked against the lexicon to see if any of these tokens match with rules defined for the SQL node type NNT – Name Node Table to isolate a table to extract data from, in the Customer Relationship Index database.

Token Id 1, 4, 6 and 8 is selected and checked against the lexicon for NNT rules.

NLRID	NLRule	NL Rule Symbol	Node	Node Description
10	Administrator	Administrators	NNT	Name Node Table
11	Administrators	Administrators	NNT	Name Node Table
12	Admin	Administrators	NNT	Name Node Table
13	Admin_Users	Administrators	NNT	Name Node Table
14	Admn	Administrators	NNT	Name Node Table
15	User	AllowedUsers	NNT	Name Node Table
16	Users	AllowedUsers	NNT	Name Node Table
17	Authorized_Users	AllowedUsers	NNT	Name Node Table
18	Allowed_Users	AllowedUsers	NNT	Name Node Table
19	CRI_Users	AllowedUsers	NNT	Name Node Table
20	Brand	BuyerDetail	NNT	Name Node Table
21	Brands	BuyerDetail	NNT	Name Node Table
22	Brand_Contacts	BuyerDetail	NNT	Name Node Table
23	Customer	BuyerDetail	NNT	Name Node Table
24	Customers	BuyerDetail	NNT	Name Node Table
25	Buyer	BuyerDetail	NNT	Name Node Table
26	Buyers	BuyerDetail	NNT	Name Node Table

Table 7 Natural language rules for NNT in scenario 1

Natural language rule Id (NLRID) 24 has a match for the token “Customers” implying that it refers to the table BuyerDetail in the Customer Relationship Index database.

3. Using the dependencies extracted for the tokens in the input string, the system identifies tokens that depend on the NNT token which are most likely to be the columns requested by the user.

DID	Dependency Type	Token	Parent Token
1	[root	Show	NULL
2	nsubj	me	name
3	det	the	name
4	xcomp	name	Show
5	cc	and	name
6	conj:and	designation	name
7	case	of	Customers
8	nmod:of	Customers	name

Table 8 Token dependencies for tokens in scenario 1

Dependency ID (DID) 6 and 8 tells us that tokens Customers, name and designation is interrelated.

4. The new tokens selected through dependencies are checked against the lexicon to see if any of these tokens match with rules defined for the SQL node type NNC – Name Node Column, to see if these tokens correspond to any columns in the Customer Relationship Index database.

NLRID	NLRule	NL Rule Symbol	Node	Node Description
36	Name	[Contact Person]	NNC	Name Node Column
37	Names	[Contact Person]	NNC	Name Node Column
38	Calling_Name	[Contact Person]	NNC	Name Node Column
1002	Designation	[Designation]	NNC	Name Node Column
1003	Designations	[Designation]	NNC	Name Node Column

1004	Title	[Designation]	NNC	Name Node Column
1005	Titles	[Designation]	NNC	Name Node Column
1007	Department	[Department]	NNC	Name Node Column
1008	Departments	[Department]	NNC	Name Node Column
1009	Section	[Department]	NNC	Name Node Column
1010	Sections	[Department]	NNC	Name Node Column
1012	Mobile	[MobilePhoneNumber]	NNC	Name Node Column
1013	Phone	[MobilePhoneNumber]	NNC	Name Node Column
1014	Phone_Number	[MobilePhoneNumber]	NNC	Name Node Column

Table 9 Natural language rules for NNC in scenario 1

Natural language rule Id 36 and 1002 has a match for the tokens “Name” and “Designation” which corresponds to the columns “[Contact Person]” and “[Designation]” in the Customer Relationship Index database.

5. Now that the token to SQL node mapping is completed the T-SQL statement is generated.

NLRID	NLRule	Token	NL Rule Symbol	Node
24	Customers	Customers	BuyerDetail	NNT
36	Name	name	[Contact Person]	NNC
1002	Designation	designation	[Designation]	NNC

Table 10 Tokens mapped to SQL nodes in scenario 1

Based on this results set the T-SQL output is generated.

7.3.4 Output

SELECT [Contact Person], [Designation] FROM BuyerDetail

The figure below shows the output returned from the NLIDB converter for Customer Relationship Index for scenario 1.

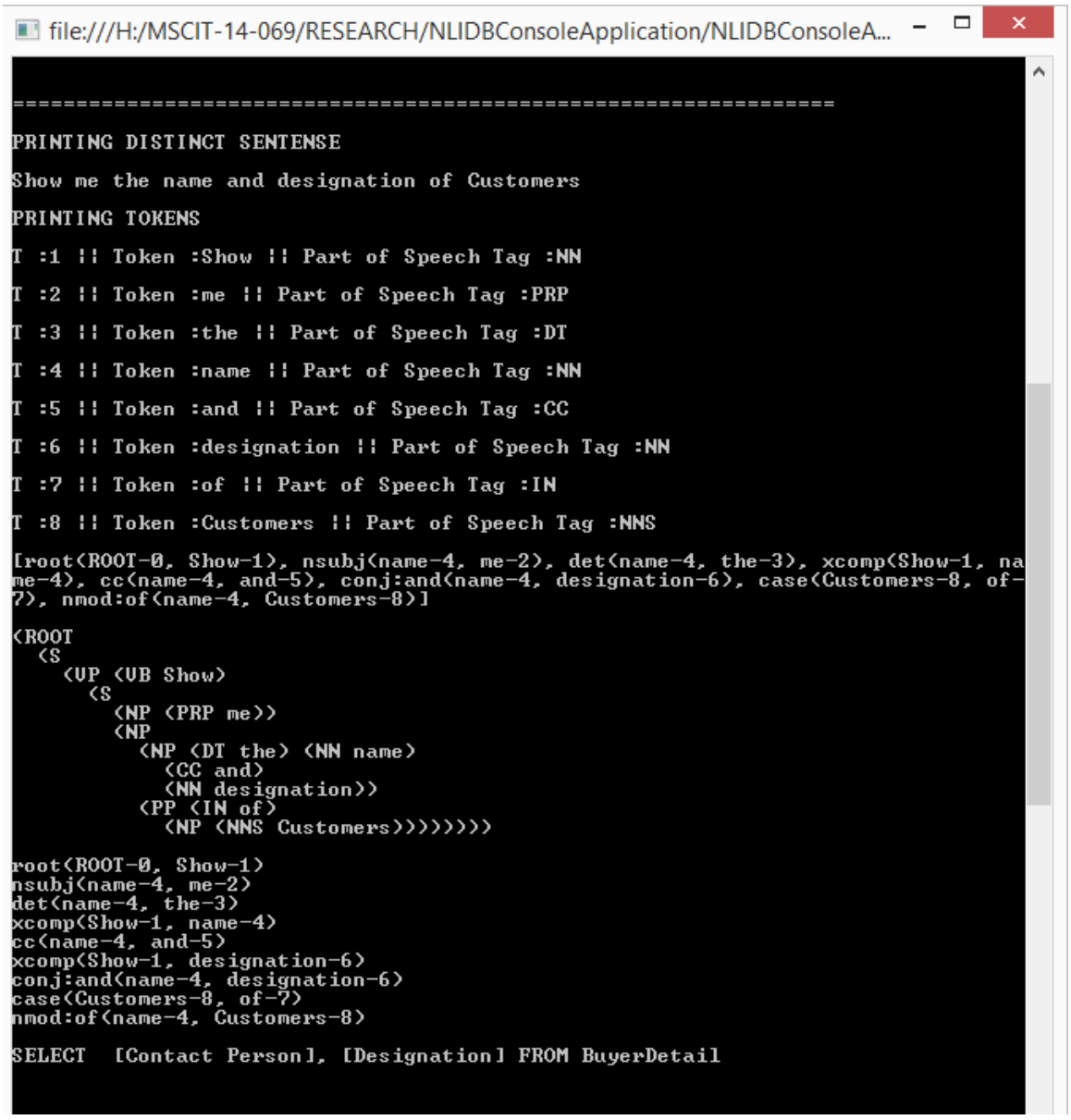


Figure 7.4 Initial Evaluation -Scenario 1 - Outputs

7.3.5 Results

The figure below shows the results extracted when the T-SQL statement is run against the Customer Relationship Index database.

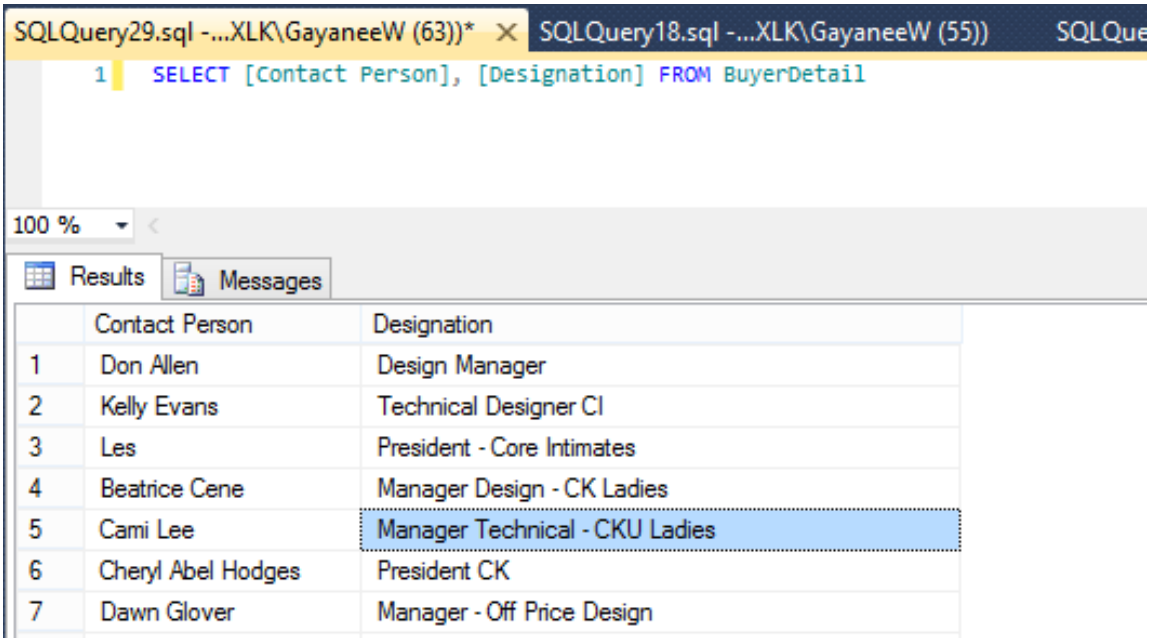


Figure 7.5 Initial Evaluation - Scenario 1 – Results

7.4 Scenario 2

7.4.1 Scenario Description

This scenario tests how the application responds when a user input is submitted that contains matching natural language rules in the lexicon for the table but not the columns allowing successful token to SQL node matching for tables only.

7.4.2 Input String

Show me details of the Brands

The figure below shows the input submitted to the NLIDB converter for Customer Relationship Index for Scenario 2.

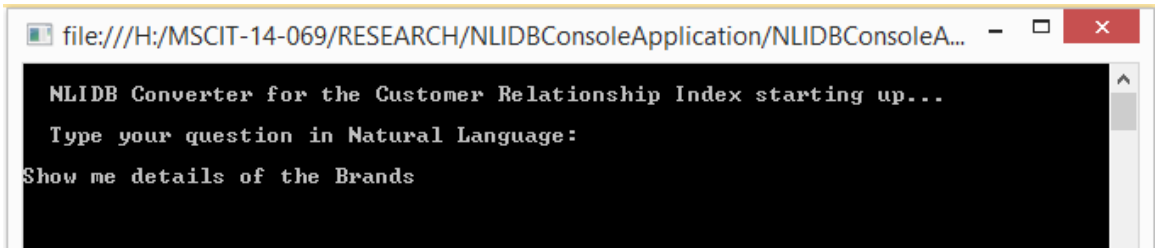


Figure 7.6 Initial Evaluation - Scenario 2 - Inputs

7.4.3 Arriving at the output

1. Using the Penn Treebank tags the tokens in the input string is categorized.

Token ID	Token	Token Type	Token Description
14	Brands	NNS	noun, common, plural
11	details	NNS	noun, common, plural
10	me	PRP	pronoun, personal
12	of	IN	preposition or conjunction, subordinating
9	Show	NN	noun, common, singular or mass
13	the	DT	determiner

Table 11 Tokens in Scenario 2

2. The NN% typed tokens are checked against the lexicon to see if any of these tokens match with rules defined for the SQL node type NNT – Name Node Table to isolate a table to extract data from, in the Customer Relationship Index database.

Token Id 9, 11, and 14 is selected and checked against the lexicon for NNT rules.

NLRID	NLRule	NL Rule Symbol	Node	Node Description
10	Administrator	Administrators	NNT	Name Node Table
11	Administrators	Administrators	NNT	Name Node Table
12	Admin	Administrators	NNT	Name Node Table
13	Admin_Users	Administrators	NNT	Name Node Table
14	Admn	Administrators	NNT	Name Node Table
15	User	AllowedUsers	NNT	Name Node Table
16	Users	AllowedUsers	NNT	Name Node Table
17	Authorized_Users	AllowedUsers	NNT	Name Node Table
18	Allowed_Users	AllowedUsers	NNT	Name Node Table
19	CRI_Users	AllowedUsers	NNT	Name Node Table
20	Brand	BuyerDetail	NNT	Name Node Table
21	Brands	BuyerDetail	NNT	Name Node Table
22	Brand_Contacts	BuyerDetail	NNT	Name Node Table
23	Customer	BuyerDetail	NNT	Name Node Table
24	Customers	BuyerDetail	NNT	Name Node Table
25	Buyer	BuyerDetail	NNT	Name Node Table
26	Buyers	BuyerDetail	NNT	Name Node Table

Table 12 Natural language rules for NNT in scenario 2

Natural language rule Id (NLRID) 21 has a match for the token “Brands” implying that it refers to the table BuyerDetail in the Customer Relationship Index database.

3. Using the dependencies extracted for the tokens in the input string, the system identifies tokens that depend on the NNT token which are most likely to be the columns requested by the user.

DID	DependencyType	Token	ParentToken
9	[root	Show	NULL
10	nsubj	me	details
11	xcomp	details	Show
12	case	of	Brands
13	det	the	Brands
14	nmod:of	Brands	details

Table 13 Token dependencies for tokens in scenario 2

Dependency ID (DID) 10,11,12,13 and 14 tells us that tokens Brands, details, of, the, Show and me are interrelated.

4. The new tokens selected through dependencies are checked against the lexicon to see if any of these tokens match with rules defined for the SQL node type NNC – Name Node Column, to see if these tokens correspond to any columns in the Customer Relationship Index database.

NLRID	NLRule	NL Rule Symbol	Node	NodeDescription
36	Name	[Contact Person]	NNC	Name Node Column
37	Names	[Contact Person]	NNC	Name Node Column
38	Calling_Name	[Contact Person]	NNC	Name Node Column
1002	Designation	[Designation]	NNC	Name Node Column
1003	Designations	[Designation]	NNC	Name Node Column
1004	Title	[Designation]	NNC	Name Node Column
1005	Titles	[Designation]	NNC	Name Node Column
1007	Department	[Department]	NNC	Name Node Column
1008	Departments	[Department]	NNC	Name Node Column

1009	Section	[Department]	NNC	Name Node Column
1010	Sections	[Department]	NNC	Name Node Column
1012	Mobile	[MobilePhoneNumber]	NNC	Name Node Column
1013	Phone	[MobilePhoneNumber]	NNC	Name Node Column
1014	Phone_Number	[MobilePhoneNumber]	NNC	Name Node Column

Table 14 Natural language rules for NNC in scenario 2

There are no natural language rule Ids for the dependent tokens in the lexicon.

5. Now that the token to SQL node mapping is completed the T-SQL statement is generated.

NLRID	**NLRule**	**Token**	**NL Rule Symbol**	**Node**
21	Brands	Brands	BuyerDetail	NNT

Table 15 Tokens mapped to SQL nodes in scenario 1

Based on this results set the T-SQL output is generated.

7.4.4 Output

SELECT * FROM BuyerDetail

The figure below shows the output returned from the NLIDB converter for Customer Relationship Index for scenario 2.

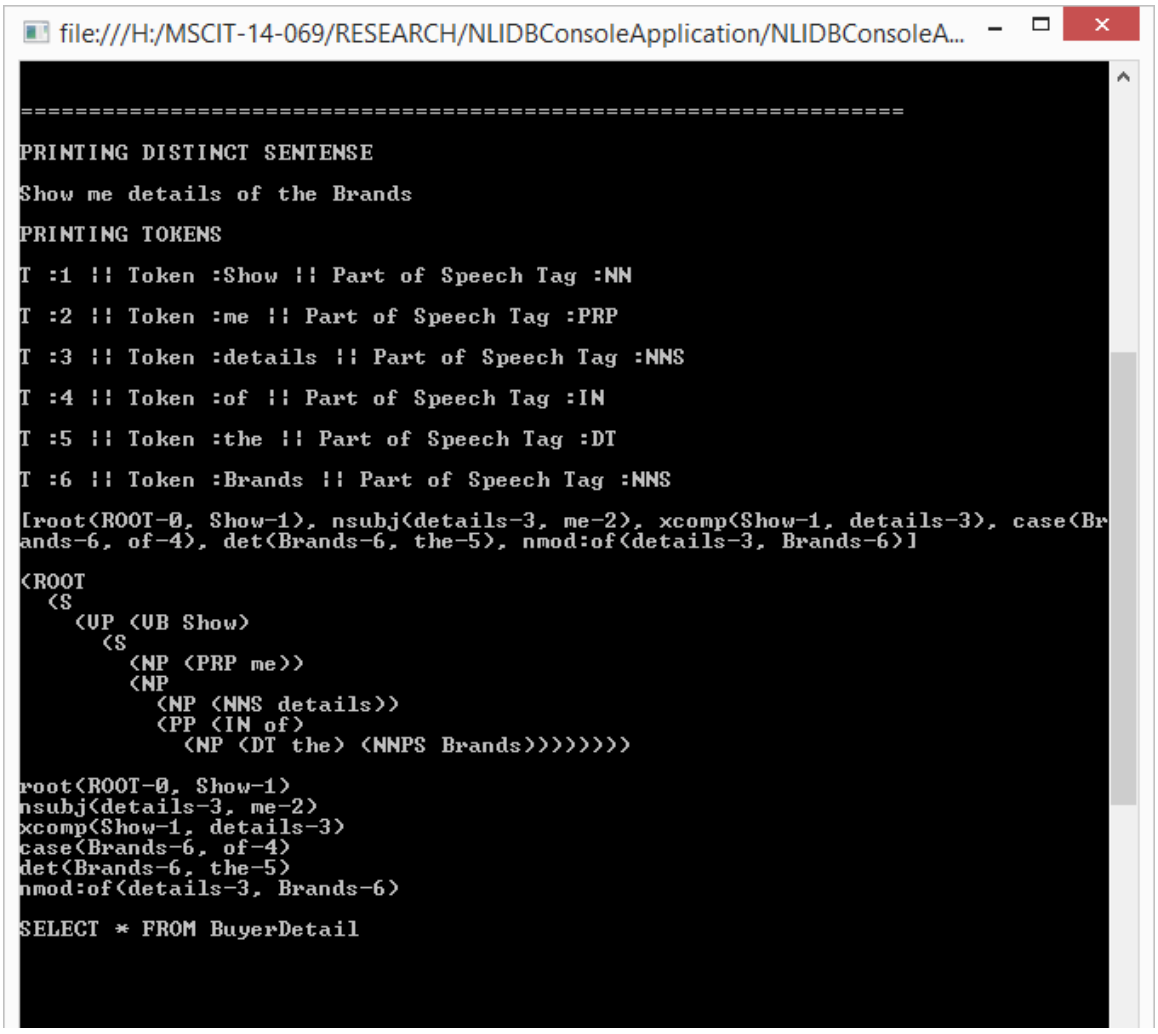


Figure 7.7 Initial Evaluation - Scenario 2 – Outputs

7.4.5 Results

The figure below shows the results extracted when the T-SQL statement is run against the Customer Relationship Index database.

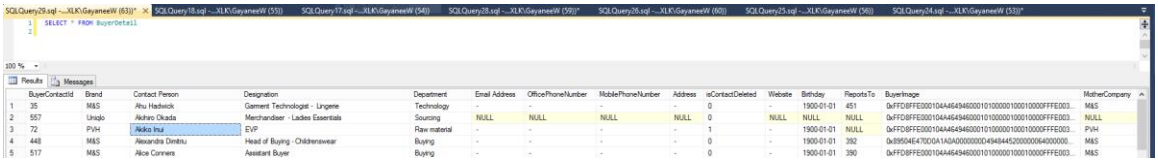


Figure 7.8 - Initial Evaluation - Scenario 2 – Results

7.5 Scenario 3

7.5.1 Scenario Description

This scenario tests how the application responds when a user input is submitted that does not contain matching natural language rules in the lexicon for neither table nor column.

7.5.2 Input String

Show me the details of the people I export garments to

The figure below shows the input submitted to the NLIDB converter for Customer Relationship Index for scenario 3.

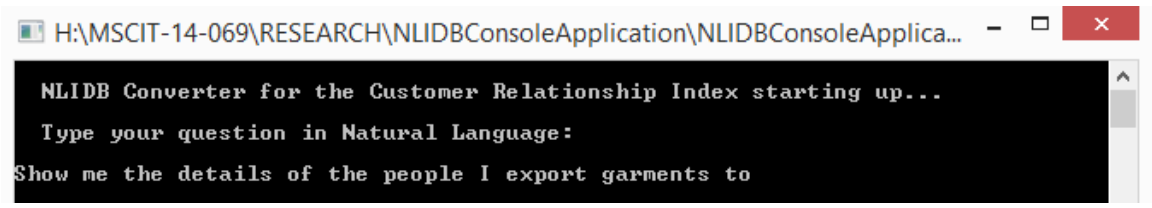


Figure 7.9 Initial Evaluation - Scenario 3 – Inputs

7.5.3 Arriving at the output

1. Using the Penn Treebank tags the tokens in the input string is categorized.

Token ID	Token	TokenType	TokenDescription
18	details	NNS	noun, common, plural
23	export	VBP	verb, present tense, not 3rd person singular
24	garments	NNS	noun, common, plural
22	I	PRP	pronoun, personal
16	me	PRP	pronoun, personal
19	of	IN	preposition or conjunction, subordinating
21	people	NNS	noun, common, plural
15	Show	NN	noun, common, singular or mass
17	the	DT	determiner
25	to	TO	to as preposition or infinitive marker

Table 16 Tokens in Scenario 3

2. The NN% typed tokens are checked against the lexicon to see if any of these tokens match with rules defined for the SQL node type NNT – Name Node Table to isolate a table to extract data from, in the Customer Relationship Index database.

Token Id 18, 21 and 24 is selected and checked against the lexicon for NNT rules.

NLRID	NLRule	NL Rule Symbol	Node	NodeDescription
10	Administrator	Administrators	NNT	Name Node Table
11	Administrators	Administrators	NNT	Name Node Table
12	Admin	Administrators	NNT	Name Node Table
13	Admin_Users	Administrators	NNT	Name Node Table
14	Admn	Administrators	NNT	Name Node Table
15	User	AllowedUsers	NNT	Name Node Table
16	Users	AllowedUsers	NNT	Name Node Table
17	Authorized_Users	AllowedUsers	NNT	Name Node Table

18	Allowed_Users	AllowedUsers	NNT	Name Node Table
19	CRI_Users	AllowedUsers	NNT	Name Node Table
20	Brand	BuyerDetail	NNT	Name Node Table
21	Brands	BuyerDetail	NNT	Name Node Table
22	Brand_Contacts	BuyerDetail	NNT	Name Node Table
23	Customer	BuyerDetail	NNT	Name Node Table
24	Customers	BuyerDetail	NNT	Name Node Table
25	Buyer	BuyerDetail	NNT	Name Node Table
26	Buyers	BuyerDetail	NNT	Name Node Table

Table 17 Natural language rules for NNT in scenario 3

There are no natural language rule Ids for the NN% tokens in the lexicon. This means that the question is not clear enough for the NLIDB converter for Customer Relationship Index to translate the question in to its T-SQL form.

7.6 Output

The NLIBD converter for Customer Relationship Index informs the user “MATCHING NATURAL LANGUAGE RULES NOT FOUND FOR INPUT TOKENS UNDER TABLES OR VIEWS” and stops further execution.

The figure below shows the output returned from the NLIDB converter for Customer Relationship Index for scenario 3.

Figure 7.10 Initial Evaluation - Scenario 3 – Outputs

7.7 Summary

Chapter 7 presented and discussed details of how the NLIDB converter for Customer Relationship Index performed when it was released to be evaluated by the software team of Brandix Apparel Solution LTD – Essentials. Next chapter presents the details of the applications formal evaluation. The formal evaluation includes details of the test cases, participants, feedback, data collection and data analysis.

Chapter 8

8 Evaluation

8.1 Introduction

In Chapter 7 we evaluated how the NLIDB converter for Customer Relationship Index works in the practical environment using a few scenarios and explained its behavior and responses. This chapter formally evaluates the NLIDB converter for the Customer Relationship Index by discussing the testing setup, the participants of the tests, the test cases and emphasizes the results of the tests and explains how the data collection was carried out to fine tune the system and identify any improvements to be made.

8.2 Participants

For the evaluation of the NLIDB converter for Customer Relationship Index, a set of fifteen users from multiple backgrounds were selected based on their experience, training, fluency in English and the education level and were divided in to four groups.

The different backgrounds and the participant teams falling in to these four groups are described in the forthcoming section.

G1: Brandix Apparel Solution LTD – Essentials – Software Team

This team consists of highly skilled Software Engineers and Database Architects with an experience of minimum 4 years in the current industry. This team is the proprietary owners of the Customer Relationship Index web application.

The main reason in associating this team to test the system is because, since they designed and developed the Customer Relationship Index web application and the underlying database they have a greater depth of knowledge of the database schema. Hence making it easy for them to verify and validate the results.

Five members of the software team was selected for this evaluation.

G2: Brandix Apparel Solution LTD – Essentials – Software Support Team

This team consists of highly skilled and trained support personnel's who come from software development and information systems backgrounds. They facilitate and assist in the system support functionality providing for the day to day needs of the users of Brandix Apparel Solution LTD – Essentials by handling support tickets within 8 hours of notification. Since these users frequently work with almost all of the associates in a daily basis they are well exposed to many types of issues that users face daily when working with systems and therefore can give good insight to the functionality and usability aspects of the system.

Three members of the software support team was selected for this evaluation.

G3: Customer Relationship Index User

The Customer Relationship Index user is typically a manager or a senior manager of Brandix Apparel Solution LTD – Essentials who works in the merchandising function which constantly deals and collaborates with the customers. During the development of the Customer Relationship Index these users have been involved with it providing the business requirements and the data to be fed in to the system. Therefore, these users have a vast knowledge and experience in both working with the customers and the Customer Relationship Index web application. This team knows the potential natural language questions that would be run against the NLIDB converter for Customer Relationship Index and the type of information that needs to be extracted from it allowing us to see how accurate and efficient the system is once it's deployed in the live environment.

Five current users of the Customer Relationship Index web application are selected for this evaluation.

G4: Non Customer Relationship Index User

This group of users are the average merchandisers of Brandix Apparel Solution LTD – Essentials who do not user the Customer Relationship Index but has a sound understanding of the merchandising and marketing functions. This team is specifically selected in order

to see what type of natural language questions a user would normally try to run against a natural language interface and see how the NLIDB converter for Customer Relationship Index responds to it.

Two merchandisers were selected for this evaluation.

8.3 Test Cases

The test cases used to test the NLIDB converter for Customer Relationship Index was specifically designed in such a way to capture all possible types of natural language queries that are going to be run against the system. The assumption that we make here is that if the system passes the test for one test scenario group, then it should run successfully for natural language queries which fall under that particular test scenario group, provided that required natural language rules are properly defined in the lexicon for the tables, views and columns used in that query.

The reason behind this type of scenario based test cases is that this testing is based on extracting the linguistic meaning of a sentence. The way people express their thought is different, their fluency in English is different. Therefore, coming up with all possible options to test is not feasible.

For each test case a natural language query representing a group of query types has been selected. And the system was tested against these selected natural language queries which represents a group of query types.

The test groups and test cases used for the NLIDB converter for Customer Relationship Index is presented next.

Test Group 1:

The tokens in the natural language query cannot be mapped against any of the SQL node types.

Test Case No	**1**	
Test Case	Test system functionality when input statement doesn't contain any tokens that can be matched and mapped to a SQL node type (NNT, NNV or NNC) through lexicon.	
Priority	1	
Prerequisites	User should be logged in to the system. Console application should be opened.	
Inputs	Show me the details of the people I export garments to	
Steps	**Step**	**Expected Result**
	Go to console application	Console application should be up and running. "Type your question in Natural Language:" statement should be displayed.
	Type statement "Show me the details of the people I export garments to"	Typed letters should be visible
	Press Enter	User should get an acknowledgement from the system indicating that it accepted the input
	Check the response from the application	User should get a message stating that the token mapping to SQL nodes were not successful.
Actual Results (Output)	An error statement displayed as "MATCHING NATURAL LANGUAGE RULES NOT FOUND FOR INPUT TOKENS UNDER TABLES OR VIEWS"	
Status	Passed	

Table 18 Test case 1

Test Group 2:

The tokens in the natural language query is mapped against only one NNT SQL node.

Test Case No	**2**	
Test Case	Test system functionality when input statement contain only one token that can be mapped and matched to a SQL node of type NNT through the lexicon.	
Priority	2	
Prerequisites	User should have been logged in to the system. Console application should be opened.	
Inputs	Show me details of the Brands	
Steps	**Step**	**Expected Result**
	Go to console application	Console application should be up and running. "Type your question in Natural Language:" statement should be displayed.
	Type statement "Show me details of the Brands"	Typed letters should be visible
	Press Enter	User should get an acknowledgement from the system indicating that it accepted the input
	Check the response from the application	User should get a T-SQL statement.
Actual Results (Output)	The T-SQL statement "SELECT * FROM BuyerDetail" is displayed.	
Status	Passed	

Table 19 Test case 2

Test Group 3:

The tokens in the natural language query is mapped against only one NNT SQL node and only one NNC SQL node of the same table.

Test Case No	**3**	
Test Case	Test system functionality when input statement contain only one token that can be mapped and matched to a SQL node of type NNT and another token that can be matched and mapped to a NNC typed token of the same table through the lexicon.	
Priority	2	
Prerequisites	User should have been logged in to the system. Console application should be opened.	
Inputs	Show me the name of customers	
Steps	Step	Expected Result
	Go to console application	Console application should be up and running. "Type your question in Natural Language:" statement should be displayed.
	Type statement "Show me the name of customers"	Typed letters should be visible
	Press Enter	User should get an acknowledgement from the system indicating that it accepted the input
	Check the response from the application	User should get a T-SQL statement.
Actual Results (Output)	The T-SQL statement "SELECT [Contact Person] FROM BuyerDetail" is displayed.	
Status	Passed	

Table 20 Test case 3

Test Group 4:

The tokens in the natural language query is mapped against only one NNT SQL node and two NNC SQL nodes of the same table.

Test Case No	**4**	
Test Case	Test system functionality when input statement contain only one token that can be mapped and matched to a SQL node of type NNT and two tokens that can be matched and mapped to a NNC typed tokens of the same table through the lexicon.	
Priority	2	
Prerequisites	User should have been logged in to the system. Console application should be opened.	
Inputs	Show me the name and designation of Customers	
Steps	Step	Expected Result
	Go to console application	Console application should be up and running. "Type your question in Natural Language:" statement should be displayed.
	Type statement "Show me the name and designation of Customers"	Typed letters should be visible
	Press Enter	User should get an acknowledgement from the system indicating that it accepted the input
	Check the response from the application	User should get a T-SQL statement.
Actual Results (Output)	The T-SQL statement "SELECT [Contact Person], [Designation] FROM BuyerDetail" is displayed.	
Status	Passed	

Table 21 Test case 4

Test Group 5:

The tokens in the natural language query is mapped against only one NNT SQL node and three NNC SQL nodes of the same table.

Test Case No	**5**	
Test Case	Test system functionality when input statement contain only one token that can be mapped and matched to a SQL node of type NNT and three tokens that can be matched and mapped to a NNC typed tokens of the same table through the lexicon.	
Priority	2	
Prerequisites	User should have been logged in to the system. Console application should be opened.	
Inputs	Show me the name, designation and department of the customers	
Steps	Step	Expected Result
	Go to console application	Console application should be up and running. "Type your question in Natural Language:" statement should be displayed.
	Type statement "Show me the name, designation and department of the customers"	Typed letters should be visible.
	Press Enter	User should get an acknowledgement from the system indicating that it accepted the input.
	Check the response from the application	User should get a T-SQL statement.
Actual Results (Output)	The T-SQL statement "SELECT [Contact Person], [Designation], [Department] FROM BuyerDetail" is displayed.	
Status	Passed	

Table 22 Test case 5

Test Group 6:

Test the system functionality when the user tries to input a blank statement.

Test Case No	**6**	
Test Case	Test system functionality when input statement is blank	
Priority	1	
Prerequisites	User should have been logged in to the system. Console application should be opened.	
Input		
Steps	**Step**	**Expected Result**
	Go to console application	Console application should be up and running. “Type your question in Natural Language:” statement should be displayed.
	Press Enter without typing anything	User should get an acknowledgement from the system indicating that it accepted the input.
	Check the response from the application	User should get an error message.
Actual Results	An error statement displayed as “THE INPUT STRING IS EMPTY. THIS IS NOT ALLOWED. PLEASE TRY AGAIN.”	
Status	Passed	

Table 23 Test case 6

Test Group 7:

The tokens in the natural language query is mapped against more than one NNT SQL node.

Test Case No	**7**	
Test Case	Test system functionality when input statement contains more than one token that can be mapped and matched to a SQL node of type NNT through the lexicon.	
Priority	1	
Prerequisites	User should have been logged in to the system. Console application should be opened.	
Inputs	Show me the details of customers and brands	
Steps	**Step**	**Expected Result**
	Go to console application	Console application should be up and running. "Type your question in Natural Language:" statement should be displayed.
	Type statement "Show me the details of customers and brands"	Typed letters should be visible
	Press Enter	User should get an acknowledgement from the system indicating that it accepted the input
	Check the response from the application	User should get an error message.
Actual Results (Output)	An error statement displayed as "CRI'S NLIDB CONVERTER NEEDS TO BE IMPLEMENTED FOR MULTIPLE TABLES OR VIEWS: D"	
Status	Passed	

Table 24 Test case 7

Test Group 8:

The input in natural language contains more than one sentence.

Test Case No	**8**	
Test Case	Test system functionality when the input statement contains more than one sentence.	
Priority	1	
Prerequisites	User should have been logged in to the system. Console application should be opened.	
Steps	**Step**	**Expected Result**
	Go to console application	Console application should be up and running. "Type your question in Natural Language:" statement should be displayed.
	Type statement "Show me the details of brands. Show me the name of customers"	Typed letters should be visible
	Press Enter	User should get an acknowledgement from the system indicating that it accepted the input
	Check the response from the application	User should get an error message.
Actual Results (Output)	An error statement displayed as "YOUR QUESTION CONTAINS 2 DISTINCT SENTENSES. PLEASE TRY A MORE SIMPLIFIED QUESTION."	
Status	Passed	

Table 25 Test case 8

8.4 Testing Setup

The selected fifteen participants were invited to come to an introductory session which provided a brief overview of the objectives of the session, the NLIDB converter for Customer Relationship Index and about natural language query processing in general. Then the users were taken through the new system in order to provide a user training before they can conduct the tests in the system themselves.

After the user training was completed, the NLIDB converter for Customer Relationship Index was installed in all the users' machines (desktops and laptops). In order for the users to get used to the system two days were given for them to test and trial with the system. Any natural language inputs and any generated results during this period were not considered for the evaluation.

Natural language questions submitted to the system and the results generated for these inputs after this grace period of two days was considered for the evaluation. Two weeks were given to the users to test the system and submit their queries in natural language.

Once the test data is collected, the software team of Brandix Apparel Solution LTD – Essentials was asked to provide the T-SQL statements for the natural language questions asked against the NLIDB converter for Customer Relationship Index by the test users. The T-SQL statements generated by the NLIDB converter for Customer Relationship Index was then reviewed against the T-SQL statements provided by the software team of Brandix Apparel Solution LTD – Essentials.

8.5 Data Collection

The data collection for the Testing of NLIDB converter for Customer Relationship Index was carried out in 2 stages.

Stage 1: Test Data

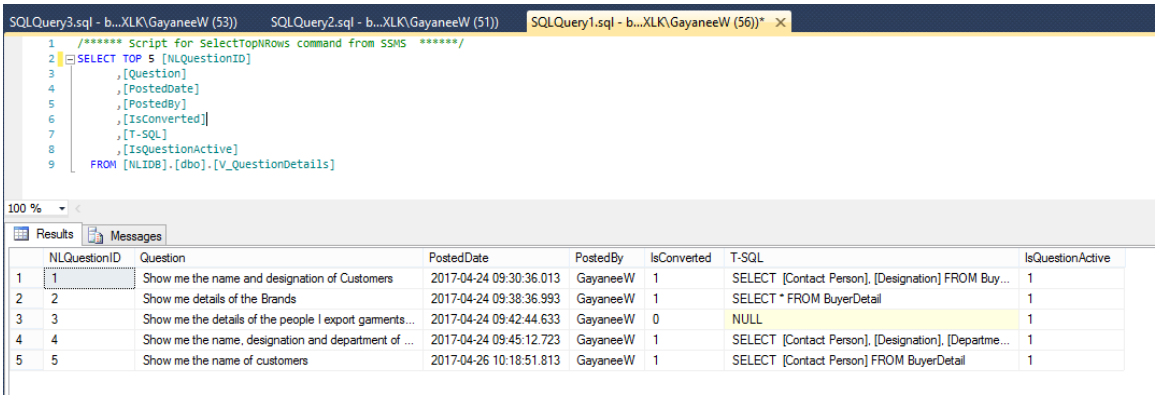
When a user posts a question against the NLIDB converter for Customer Relationship Index, the details of the question, for example the question itself, who asked the question,

when and at what time the question was asked is saved in a table ([NLIDB].[dbo].[NaturalLanguageQuestions]) by the system automatically.

Once the natural language question is converted to the T-SQL statement, the T-SQL statement is saved in a table ([NLIDB].[dbo].[SQLQuestions]) by the system as well.

By performing a left join on [NLIDB].[dbo].[NaturalLanguageQuestions] and [NLIDB].[dbo].[SQLQuestions] a data set like the following can be collected.

The figure below shows such five records that are extracted from the NLIDB converter for Customer Relationship Index.



	NLQuestionID	Question	PostedDate	PostedBy	IsConverted	T-SQL	IsQuestionActive
1	1	Show me the name and designation of Customers	2017-04-24 09:30:36.013	GayaneeW	1	SELECT [Contact Person], [Designation] FROM Buy...	1
2	2	Show me details of the Brands	2017-04-24 09:38:36.993	GayaneeW	1	SELECT * FROM BuyerDetail	1
3	3	Show me the details of the people I export garments...	2017-04-24 09:42:44.633	GayaneeW	0	NULL	1
4	4	Show me the name, designation and department of ...	2017-04-24 09:45:12.723	GayaneeW	1	SELECT [Contact Person], [Designation], [Departme...	1
5	5	Show me the name of customers	2017-04-26 10:18:51.813	GayaneeW	1	SELECT [Contact Person] FROM BuyerDetail	1

Figure 8.1 Data Collection in NLIDB converter for Customer Relationship Index

Stage 2: User Feedback

The user feedback was collected from each user group that participated in testing the NLIDB converter for Customer Relationship Index. In order to collect the feedback from each user, one of two questionnaires were submitted to the users depending on which user group they belonged to.

One questionnaire was specifically designed for G1: Brandix Apparel Solution LTD – Essentials – Software Team and G2: Brandix Apparel Solution LTD – Essentials – Software Support Team which focuses more on the technical aspects of the system and another questionnaire for G3: Customer Relationship Index User and G4: Non Customer Relationship Index Users which focuses more on usability of the system. (Appendix C)

8.6 Data Analysis

Data analysis of the NLIDB converter for Customer Relationship Index was carried out using the data, information and feedback collected during the data collection stage discussed under point 8.5.

The test data shed light that the test user groups had executed 121 natural language queries against the NLIDB converter for Customer Relationship Index during the test phase. The original question in natural language and the NLIDB converter for Customer Relationship Index's response which is the generated SQL statement was then passed to the database architect of the Customer Relationship Index in Brandix Apparel Solution LTD – Essentials to get her conformation on the accuracy of the system generated SQL statements.

The results returned are as follows,

Total Questions Posted	Correctly Converted	Incorrectly Converted	Skipped
121	97	15	9

Table 26: Summary of Evaluation Results

To illustrate the results furthermore,

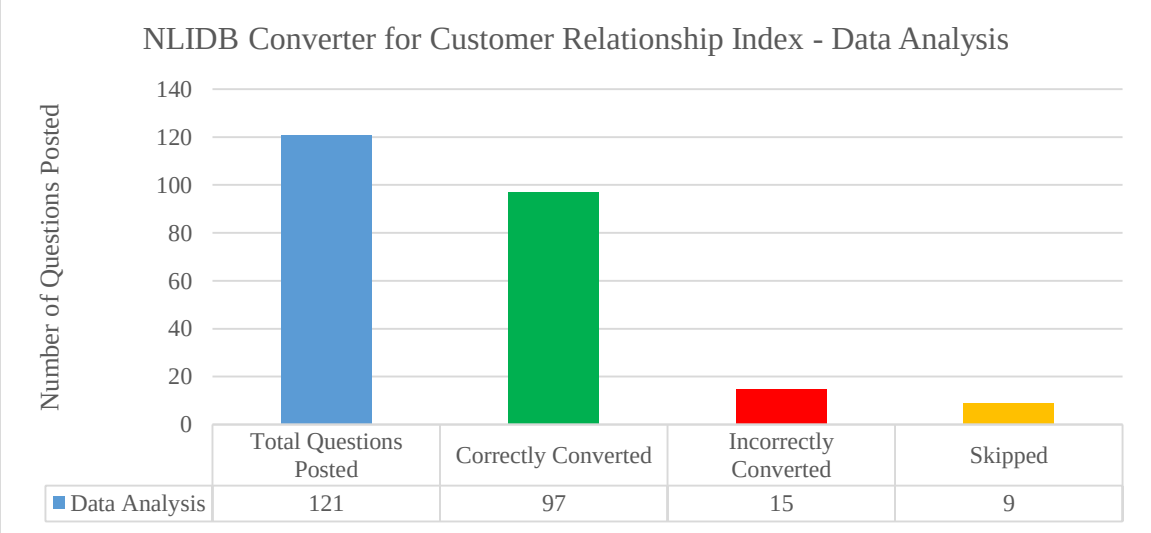


Figure 8.2 Summary of Evaluation Results

Additionally, the evaluation of the algorithm used in converting natural language to T-SQL statements and the usability of the NLIDB converter for Customer Relationship Index was analyzed using the collected data.

8.6.1 Algorithm Analysis

By considering the test results (inputs, expected output and the actual output) of the testing carried out under the provided test groups in this document mentioned under point 8.3 the following observations were made.

1. The NLIDB converter for Customer Relationship Index can successfully convert natural language questions that are translated to "SELECT * FROM TABLE_X" form. (Test Group 2)
2. The NLIDB converter for Customer Relationship Index can successfully convert natural language questions that are translated to "SELECT COLUMN_1 FROM TABLE_X" form. (Test Group 3)
3. The NLIDB converter for Customer Relationship Index can successfully convert natural language questions that are translated to "SELECT COLUMN_1, COLUMN_2 FROM TABLE_X" form. (Test Group 4)
4. The NLIDB converter for Customer Relationship Index can successfully convert natural language questions that are translated to "SELECT COLUMN_1, COLUMN_2, COLUMN_3 FROM TABLE_X" form. (Test Group 5)

   This implies that the following format of SQL statements "SELECT COLUMN_1, Column_2, ..., COLUMN_N FROM TABLE_X", can be supported by the NLIDB converter for Customer Relationship Index.
5. If there are no matching natural language rules defined in the lexicon the NLDIB converter for Customer Relationship Index displays an error message. (Test Group 1)
6. If an invalid input is submitted to the NLIDB converter for Customer Relationship Index, then the system doesn't accept the input and displays an error message. (Test Group 6)
7. The algorithm cannot translate natural language to T-SQL when multiple tables are involved. (Test Group 7)

8. The algorithm can't and will not accept inputs when multiple sentences are submitted as the input natural language query. (Test Group 8)
9. The algorithm cannot support WHERE clauses in the T-SQL statement.
10. The algorithm cannot support aggregate functions.
11. The algorithm cannot support join statements as it fails when multiple tables are mentioned. (Test Group 7)

8.6.2 Usability and Performance Analysis

Based on the feedback collected from the two questionnaires the following observations were made.

In the "popular feedback" rating, 1 stands for Poor and 5 stands for Excellent.

Questionnaire: G1 and G2

ID	Criteria	Popular feedback
1	How would you rate the installation process?	4
2	How satisfied are you with the overall design of the system?	4
3	How satisfied are you about the response time of the system?	5
4	How satisfied are you with the results returned from the system?	4
5	How satisfied are you with the execution of the NLIDB converter for CRI?	4
6	Did you run in to any critical issues during execution?	No
7	Did you run in to any bugs during execution?	No
8	Are the system responses / feedback clear?	Yes

Table 27 Feedback summary questionnaire: G1 and G2

Questionnaire: G3 and G4

ID	Criteria	Popular feedback
1	How would you rate the ease of access of the system?	4
2	How satisfied are you with the user interface?	3
3	How satisfied are you with the input mechanism of the natural language questions?	3
4	How satisfied are you about the response time of the system?	5
5	How satisfied are you with the overall performance of the system?	4
6	Did you run in to any critical issues during execution?	No
7	Did you run in to any bugs during execution?	No
8	Are the system responses / feedback clear?	Yes

Table 28 Feedback summary questionnaire: G3 and G4

Based on the user feedback it is evident that the test participants are satisfied with the overall performance, response time, system responses and the accuracy of the system. However, the results indicate that they are less satisfied with the user interface of the NLIDB converter for Customer Relationship Index and the input mechanism used in this.

8.7 Summary

This chapter presents the details of the critical evaluation of the NLIDB converter for Customer Relationship Index by describing the participants involved in testing, the test set up, the test criteria or the test cases used and how data was collected after the testing was complete. Furthermore, this chapter presents an analysis of the data collected during the testing process. The next chapter will discuss the conclusion and future work with regard to the NLIDB converter for Customer Relationship Index.

Chapter 9

9 Conclusion and Future Work

9.1 Introduction

This chapter concludes the research presented in this thesis. We recall our hypothesis as "Successful implementation of Natural Language Interface to Database for the Customer Relationship Index can bridge the gap between the languages used by humans and the computers, allowing the system users to retrieve the required information they need directly from the database without seeking assistance or depending on the software team of Brandix Apparel Solution LTD – Essentials". It is evident from the results presented in Chapter 8, that a NLIDB implementation to the Customer Relationship Index web application can in fact bridge the gap between the languages used by humans and the computers, allowing the users of the Customer Relationship Index to extract information successfully without having to seek for assistance from the software team of Brandix Apparel Solution LTD – Essentials. In more than 80% of the instances the NLIDB converter for Customer Relationship Index returned the accurate T-SQL statement for the input query in natural language submitted by the user. This chapter presents the overall conclusion of the research, the objective-wise conclusions, achievements and limitations and finally suggests the future work for this project.

9.2 Overall Conclusion

The evaluation reveals and confirms that the average user of the Customer Relationship Index has benefited from an implementation of a natural language interface to the Customer Relationship Index database during their attempt at trying to satisfy the daily informational needs that must be met by the Customer Relationship Index system without the assistance of the software team of Brandix Apparel Solution LTD – Essentials with an approximate success rate of more than 80%. Different test participant groups yielded different test results ranging from 70% to 90% depending on their knowledge of the kind of information that can be extracted from the Customer Relationship Index and the underlying database schema. The test participant group G4 which consisted of the "Non Customer Relationship

Index User" yielded the lowest accuracy percentage. This is expectable as they have no prior knowledge as to the extent of which informational requests that the Customer Relationship Index database can cater to. The test participants group G3 is a subset of the average user group for the Customer Relationship Index system and this team, without any knowledge of the underlying database schema was able to get a success rate of about 80% during the evaluation. The highest success rate was recorded by the G1 team which was made up of the Brandix Apparel Solution LTD – Essentials – Software Team. This was also expected as they knew the database schema well, they were able to come up with a variety of natural language queries which returned accurate results.

In occasions when the natural language question failed to be converted to its corresponding T-SQL statement, the following scenarios were observed.

1. A significant input token had a typo error and this resulted in the failure as the token was unrecognizable therefore it couldn't be matched against the lexicon.
2. The lexicon didn't contain the required entries for the NNT, NNV or NNC tokens.
3. Differences in the speaking, writing vocabulary and language skills of the participants. Since the NLIDB converter for Customer Relationship Index extracts meaning using the tokens of the sentence submitted in natural language, language skills of a person and how they express their thoughts and ideas is a very important, decisive and a crucial factor contributing to the accuracy of the system.
4. The schema of the Customer Relationship Index and the overall design of the database is critical and this directly affects the results produced by the NLIDB converter for Customer Relationship Index.

Based on the test results and the feedback received from the participants, it proves that the implementation of a natural language interface for the Customer Relationship Index is successful at solving the problem at hand and the natural language to T-SQL converter application developed to prove this hypothesis is successful at generating accurate responses efficiently.

9.3 Objective-wise Conclusions

This document presented the objectives of carrying out this research in Chapter 1. They were,

1. To critically study and do and in depth exploration of the natural language interfaces to databases.
2. To critically study the issues faced when implementing a natural language interface to databases.
3. To identify and evaluate the technologies used for implementing a natural language interface to databases.
4. To implement a prototype of a natural language interface to the Customer Relationship Index based on research findings.
5. To evaluate the accuracy and the performance of the implemented solution.

Objective 1 has been successfully achieved by conducting a comprehensive literature survey in the domain of natural language interfaces to databases which is presented in the Chapter 2 of this document. The literature survey was conducted by using key researches based on the latest developments on the domain of natural language processing, researches based on comparisons done on available natural language interface to database techniques discussing the pros and cons of each technique and researches based on how different natural language interface to database techniques can be applied for different database managements systems under a variety of scenarios discussing the results each implementation produced.

Objective 2 has also been supported and achieved through Chapter 2 as each paper the literature review is based on presents and critically reviews the problems faced when implementing a natural language interface to a database. Some of the most common issues these systems face is, sometimes natural language can be vague, the meaning of each word can differ based on the context it is used, ideas and thoughts expressed by each individual is based on that person’s level of education and his or her speaking and writing vocabulary due to these facts implementing natural language interfaces to databases are still a vast area of research.

Objective 3 has been achieved and supported through Chapter 3 as it discusses and presents each of the technologies used to implement the proposed solution giving details and the justification as to why each technology was chosen on instances where multiple options exist to choose from.

The achievement of objective 4 is evident from the contents of the following chapters, Approach (Chapter 4), Design (Chapter 5), Implementation (Chapter 6) and NLIDB converter for Customer Relationship Index in Practice (Chapter 7). These chapters present the approach taken in designing the solution, the architecture of the proposed solution giving in depth design details of each module presenting its responsibilities and functionalities and discusses how each module is implemented using algorithms, workflows, pseudocode and actual code segments. Finally, Chapter 7 shows how the application works in the based on different types of inputs and discusses the outputs and results generated by the system.

Objective 5 has been achieved through the evaluation chapter (Chapter 9) of this thesis where it systematically reviews and studies the output and the results returned by the NLIDB converter for Customer Relationship Index. The test strategies used in the evaluation of NLIDB converter for Customer Relationship Index predominantly consisted of the testing process described in detail under the evaluation chapter as well as white box testing, black box testing and integration testing. The testing process was a methodical review consisting of 4 test user groups coming from various backgrounds comprising of 15 participants who are professional experienced software engineers, support engineers, typical users of the Customer Relationship Index and typical merchandisers testing 8 types of test groups.

Based on the responses provided by the test participants and the evaluation of results it is evident that the typical Customer Relationship Index user benefits from an implementation of a natural language interface for the Customer Relationship Index database as the average response accuracy is more than 80% for the NLIDB converter for Customer Relationship Index.

9.4 Achievements

1. The NLIDB converter is able to successfully convert queries in natural language in to T-SQL statements with an average accuracy rate which is above 80% for the Customer Relationship Index Database.
2. It successfully supports the T-SQL statement format “SELECT * FROM TABLE_X” and “SELECT COLUMN_1, ..., COLUMN_N FROM TABLE_X”.
3. The system is able to produce results within 10-20 seconds.

9.5 Limitations and Future work

Based on the analysis of the results extracted and the feedback that was collected during the evaluation of the NLIDB converter for Customer Relationship Index the following suggestions are proposed to improve and overcome the limitations faced.

1. Extend the system to support the where clause, as the current conversions to T-SQL only support “SELECT * FROM TABLE_X” or “SELECT COLUMN_1, ..., COLUMN_N FROM TABLE_X”.

   This will enable users to post queries like “Show me the address of the customer Bennadict Cumberbatch”.
2. Extend the system to support aggregate functions as the system doesn’t currently support converting to T-SQL statements that uses AVG, COUNT, SUM, MAX or MIN. This will enable the users to post queries like “How many customers do I work with?” against the NLIDB converter for Customer Relationship Index.
3. Extend the system to join multiple tables to a minimum of either two or three tables in order to gather information through a JOIN clause.

   An implementation of the JOIN clause can enable the users of the Customer Relationship Index post queries like “I want to see the locations of my administrators” against the NLIDB converter for Customer Relationship Index which is not currently supported by the system.

9.6 Summary

This chapter presented and concluded the research findings of implementing a natural language interface for Customer Relationship Index database with an average success rate above 80% in the NLIDB converter for Customer Relationship Index. The overall achievements and objective-wise achievements are presented in detail in this section. Furthermore, the achievements, limitations and future work is presented in summarized form separately. In conclusion it is proven that a natural language interface for the Customer Relationship Index does benefit the typical Customer Relationship Index user and that the implementation of the NLIDB converter for Customer Relationship Index has been successful.

10 References

[1] M. P. Bhopal, “Natural language Interface for Database: A Brief review,” *IJCSI*, p. 600, 2011.

[2] B. Hemerelain and H. Belbachir, “Semantic Analysis of Natural Language Queries for an Object Oriented Database,” *J. Softw. Eng. Appl.*, vol. 3, no. 11, pp. 1047–1053, 2010.

[3] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch, “Natural language interfaces to databases–an introduction,” *Nat. Lang. Eng.*, vol. 1, no. 1, pp. 29–81, 1995.

[4] A.-M. Popescu, O. Etzioni, and H. Kautz, “Towards a theory of natural language interfaces to databases,” in *Proceedings of the 8th international conference on Intelligent user interfaces*, 2003, pp. 149–157.

[5] “Patent US5495604 - Method and apparatus for the modeling and query of database structures using ... - Google Patents.” [Online]. Available: https://www.google.com/patents/US5495604. [Accessed: 16-Dec-2016].

[6] “Academic paper: The Lunar Science Natural Language Information System: Final Report.” [Online]. Available: https://www.researchgate.net/publication/247926251_The_Lunar_Science_Natural_Language_Information_System_Final_Report. [Accessed: 29-Apr-2017].

[7] D. Waltz and B. Goodman, “Planes: A Data Base Question-answering System,” *SIGART Bull*, no. 61, pp. 24–24, Feb. 1977.

[8] G. G. Hendrix, E. D. Sacerdoti, D. Sagalowicz, and J. Slocum, “Developing a natural language interface to complex data,” *ACM Trans. Database Syst. TODS*, vol. 3, no. 2, pp. 105–147, 1978.

[9] R. Alexander, P. Rukshan, and S. Mahesan, “Natural Language Web Interface for Database (NLWIDB),” *ArXiv Prepr. ArXiv13083830*, 2013.

[10] B. B. Huang, G. Zhang, and P. C. Y. Sheu, “A Natural Language Database Interface Based on a Probabilistic Context Free Grammar,” in *IEEE International Workshop on Semantic Computing and Systems*, 2008, pp. 155–162.

[11] M. J. Minock, “A STEP towards realizing Codd’s vision of rendezvous with the casual user,” in *Proceedings of the 33rd international conference on Very large data bases*, 2007, pp. 1358–1361.

[12] E. F. Codd, *Seven Steps to Rendezvous with the Casual User*. IBM Corporation, 1974.

[13] D. H. D. Warren and F. C. N. Pereira, “An Efficient Easily Adaptable System for Interpreting Natural Language Queries,” *Comput Linguist*, vol. 8, no. 3–4, pp. 110–122, Jul. 1982.

[14] D. L. Waltz, “An English language question answering system for a large relational database,” *Commun. ACM*, vol. 21, no. 7, pp. 526–539, Jul. 1978.

[15] B. H. Thompson and F. B. Thompson, “Introducing ask, a simple knowledgeable system,” presented at the Proceedings of the first conference on Applied natural language processing, 1983, pp. 17–24.

[16] P. Resnik, “Access to Multiple Underlying Systems in Janus,” BBN SYSTEMS AND TECHNOLOGIES CORP CAMBRIDGE MA, BBN-7142, Sep. 1989.

[17] M. Templeton and J. Burger, “Problems in Natural-language Interface to DBMS with Examples from EUFID,” in *Proceedings of the First Conference on Applied Natural Language Processing*, Stroudsburg, PA, USA, 1983, pp. 3–16.

[18] C. D. Hafner, “Interaction of Knowledge Sources in a Portable Natural Language Interface,” in *Proceedings of the 10th International Conference on Computational Linguistics and 22Nd Annual Meeting on Association for Computational Linguistics*, Stroudsburg, PA, USA, 1984, pp. 57–60.

[19] B. J. Grosz, “TEAM: a transportable natural-language interface system,” presented at the Proceedings of the first conference on Applied natural language processing, 1983, pp. 39–45.

[20] C. Reviewer-Lee, “Book review: Subsymbolic Natural Language Processing: An Integrated Model of Scripts, Lexicon, and Memory by Risto Miikkulainen (Bradford Books, MIT Press 1993),” *ACM SIGART Bull.*, vol. 6, no. 4, pp. 19–21, Oct. 1995.

[21] E. Charniak, *Statistical Language Learning*. MIT Press, 1994.

[22] K. W. Church and R. L. Mercer, “Introduction to the special issue on computational linguistics using large corpora,” *Comput. Linguist.*, vol. 19, no. 1, pp. 1–24, Mar. 1993.

[23] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, “Building a large annotated corpus of English: the penn treebank,” *Comput. Linguist.*, vol. 19, no. 2, pp. 313–330, Jun. 1993.

[24] R. G. Reilly, Ed., *Connectionist approaches to natural language processing*. Hove: Lawrence Erlbaum Assoc, 1992.

[25] L. Shastri, “A model of rapid memory formation in the hippocampal system,” in *Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society*, 1997, pp. 680–685.

APPENDIX A – Class Program – NLIDB Converter

```
using edu.stanford.nlp.ling;
using edu.stanford.nlp.pipeline;
using edu.stanford.nlp.util;
using java.io;
using java.util;
using System;
using System.Collections.Generic;
using System.IO;
using Console = System.Console;

namespace NLIDBConsoleApplication {
 internal class Program {
  public static void Main(string[] args) {
   NLIDBLogger nlidbl = new NLIDBLogger();
   NLIDBBAL nlidbbal = new NLIDBBAL();

   List < string > nnTokens = new List < string > ();

   int ruleIdNnt = 0;
   int ruleIdNnv = 0;
   int ruleIdNnc = 0;

   string dependencies = null;

   int tokenId = 0;
   int nnTokenId = 0;
   int parentTokenId = 0;
   byte tokenTypeId = 0;
   int dependencyId = 0;
   int sentencesCount = 0;

   // PATH TO THE FOLDER WITH MODELS EXTRACTED FROM `stanford-corenlp-3.7.0-models.jar`
   var jarRoot = @ "H:\MSCIT-14-069\RESEARCH\NLIDBConsoleApplication\stanford-corenlp-
full-2016-10-31\stanford-corenlp-3.7.0-models";

   Console.WriteLine("\n  NLIDB Converter for the Customer Relationship Index starting
up...\n");

   // TEXT FOR PROCESSING
   Console.WriteLine("  Type your question in Natural Language:\n");

   var question = Console.ReadLine().Trim();

   Console.WriteLine();

   // CHECKS IF THE INPUT STRING IS NOT EMPTY
   if (question != null && question.Length > 1) {
    nlidbl.CreateLogHeader(question);
    Console.WriteLine(nlidbl.MessageCheckingKBForExistingQuestion);
    nlidbl.PrintMessage(nlidbl.MessageCheckingKBForExistingQuestion);

    string userName = Environment.UserName;

    // CHECK IF THE QUESTION ALREADY EXIST IN THE QUESTIONS KNOWLEDGE BASE
    if (nlidbbal.IsQuestionInKB(question)) {
     // IF QUESTION IS IN KB
     Console.WriteLine();
     Console.WriteLine(nlidbl.MessageExistingQuestion);
```

```
     nlidbl.PrintMessage(nlidbl.MessageExistingQuestion);

     // GETS THE CORRESPONDING QUESTION ID BASED ON THE INPUT STRING
     int questionId = nlidbbal.GetIdOfNaturalLanguageQuestion(question);

     // GETS THE SQL STATEMENT BASED ON THE QUESTION ID
     string sqlStatement = nlidbbal.GetSqlStatementForQuestionID(questionId);

     // IF A VALID SQL STATEMENT IS RETURNED
     if (sqlStatement.Substring(0, 6).ToLower() == "select") {
      Console.WriteLine();
      Console.WriteLine(sqlStatement);
      nlidbl.PrintMessage(sqlStatement);
     } else // IF THE NATURAL LANGUAGE QUESTION WAS NOT CONVERTED TO SQL
SUCCESSFULLY
     {
      Console.WriteLine();
      Console.WriteLine(sqlStatement);
      nlidbl.PrintMessage(sqlStatement);
      nlidbl.PrintMessage(nlidbl.MessageSQlConversionFailed);
     }
     Console.ReadLine();
    } else // IF QUESTION IS NOT IN KB
    {
     Console.WriteLine();
     Console.WriteLine(nlidbl.MessageNewQuestion);
     nlidbl.PrintMessage(nlidbl.MessageNewQuestion);

     // ADD THE QUESTION IN TO THE QUESTIONS KB
     int questionId = nlidbbal.AddNewQuestionToKB(question, userName);

     Console.WriteLine();
     Console.WriteLine(nlidbl.MessageNlidbConverterExecutionStart);
     nlidbl.PrintMessage(nlidbl.MessageNlidbConverterExecutionStart);

     // STANFORD CORE NLP ANNOTATION PIPELINE CONFIGURATION
     var props = new Properties();
     props.setProperty("annotators", "tokenize, ssplit, pos, lemma, ner, parse,
dcoref");
     props.setProperty("ner.useSUTime", "0");

     // CHANGE CURRENT DIRECTORY, SO STANFORD CORENLP COULD FIND ALL     THE MODEL
FILES AUTOMATICALLY
     var curDir = Environment.CurrentDirectory;
     Directory.SetCurrentDirectory(jarRoot);
     var pipeline = new StanfordCoreNLP(props);
     Directory.SetCurrentDirectory(curDir);

     // ANNOTATION
     var annotation = new Annotation(question);
     pipeline.annotate(annotation);

     Console.WriteLine();
     Console.WriteLine(nlidbl.MessageOutputFromStanfordCoreNlp);
     nlidbl.PrintMessage(nlidbl.MessageOutputFromStanfordCoreNlp);

     // RESULT - PRETTY PRINT
     using(var stream = new ByteArrayOutputStream()) {
      pipeline.prettyPrint(annotation, new PrintWriter(stream));
      Console.WriteLine();
      Console.WriteLine(stream.toString());
```

```
      nlidbl.PrintMessage(stream.toString());
      stream.close();
      Console.WriteLine("================================");
     }

     //EXTRACTING SENTENCES FROM THE INPUT STRRING
     ArrayList sentences = annotation.get(new
CoreAnnotations.SentencesAnnotation().getClass()) as ArrayList;

     // GETTING THE COUNT OF SENTENSES IN THE INPUT TEXT
     sentencesCount = sentences.size();

     // NLIDB CONVERTER RUNS IF THE SENTENSES COUNT IS = MAX_SENTENSES,
THIS DOES NOT SUPPORT MULTIPLE SENTENSES
     if (sentencesCount == nlidbbal.MaxSentenses) {
      foreach(CoreMap sentence in sentences) {
       Console.WriteLine();
       Console.WriteLine(nlidbl.MessagePrintingSentense);
       nlidbl.PrintMessage(nlidbl.MessagePrintingSentense);

       Console.WriteLine();
       Console.WriteLine(sentence.ToString());
       nlidbl.PrintMessage(sentence.ToString());

       //EXTRACTING TOKENS IN THE SENTENCE
       var tokens = sentence.get(new CoreAnnotations.TokensAnnotation().getClass()) as
ArrayList;

       Console.WriteLine();
       Console.WriteLine(nlidbl.MessagePrintingTokens);
       nlidbl.PrintMessage(nlidbl.MessagePrintingTokens);

       int tokensCount = 1;

       //EXTRACT THE PROPERTIES OF EACH TOKEN IN TOKENS
       foreach(CoreLabel token in tokens) {
        // GET THE TOKEN_TEXT
        string tokenText = token.get(new
CoreAnnotations.TextAnnotation().getClass()).ToString();
        // GET THE PART OF SPEECH TAG OF TOKEN BASED ON THE PENN TREE BANK PART OF
SPEECH TAGS
        string partOfSpeechTag = token.get(new
CoreAnnotations.PartOfSpeechAnnotation().getClass()).ToString();

        // GET THE TOKEN TYPE ID BASED ON THE INPUT PART OF SPEECH TAG
        tokenTypeId = nlidbbal.TokenTypeIDForInputString(partOfSpeechTag);
        // SAVE THE TOKEN TO THE DATABASE AGAINST THE QUESTION ID AND TOKEN TYPE ID
        tokenId = nlidbbal.AddNewTokenToQuestion(tokenText, questionId, tokenTypeId);

        Console.WriteLine();
        Console.WriteLine("T :" + tokensCount + " || Token :" + tokenText + " || Part
of Speech Tag :" + partOfSpeechTag);
        nlidbl.PrintMessage("T :" + tokensCount, tokenText, partOfSpeechTag);

        tokensCount = tokensCount + 1;
       }

       // EXTRACT TOKEN DEPENDENCIES IN THE SENTENSE USING STANFORD PARSER
       dependencies = nlidbbal.GetTokenDependencies(sentence.ToString());
```

```
        // SPLITTING THE DEPENDENCIES STRING TO EXTRACT THE TOKEN AND PARENT TOKEN -
DEFINING SEPERATORS
        string[] stringSeparators = new string[] {
         "),"
        };
        // SPLITTING THE DEPENDENCIES STRING TO EXTRACT THE TOKEN AND PARENT TOKEN
        var dependencyList = dependencies.Split(stringSeparators,
StringSplitOptions.None);

        // EXTRACT THE TOKEN AND PARENT TOKEN OF EACH DEPENDENCY IN DEPENDENCY_LIST
        foreach(string dependency in dependencyList) {
         // SPLITTING THE DEPENDENCY TO EXTRACT THE TOKEN AND PARENT TOKEN - DEFINING
INNER SEPERATORS
         string[] innerStringSeperators = new string[] {
          "(",
          ",",
          "-"
         };
         // SPLITTING THE DEPENDENCIES STRING TO EXTRACT THE TOKEN AND PARENT TOKEN
         var tokenDependencies = dependency.Split(innerStringSeperators,
StringSplitOptions.None);

         if (tokenDependencies[1] == "ROOT") {
          parentTokenId = 0;
          // GET THE TOKEN ID FOR CHILD TOKEN
          tokenId = nlidbbal.GetMatchingTokenId(questionId, tokenDependencies[3]);
          int X = nlidbbal.GetTokenIdInQuestion(questionId, tokenDependencies[3]);
         } else {
          // GET THE TOKEN ID FOR PARENT TOKEN
          parentTokenId = nlidbbal.GetMatchingTokenId(questionId, tokenDependencies[1]);
          // GET THE TOKEN ID FOR CHILD TOKEN
          tokenId = nlidbbal.GetMatchingTokenId(questionId, tokenDependencies[3]);
         }

         // SAVE THE PARENT AND CHILD TOKEN TO THE DATABASE
         dependencyId = nlidbbal.AddTokenDependencies(tokenId, parentTokenId,
tokenDependencies[0]);
        }

        // GET THE NN TOKENS BASED ON THE PENN TREE BANK TAGS AND MATCH THEM WITH SQL
NODE TYPES
        // GET A LIST OF NN TOKENS FOR QUESTION
        nnTokens = nlidbbal.GetPennTreeBankPartOfSpeechTagsForQuestion_NN(questionId);

        // EACH TOKEN IS CHECKED AGAINS THE NATURAL LANGUAGE RULES FOR SQL NODE TYPES
NNT(NAME NODE TABLE) AND NNV(NAME NODE VIEW)
        foreach(var nnToken in nnTokens) {
         // GET THE NATURAL LANGUAGE RULE ID FOR NAME NODE TABLE
         ruleIdNnt = nlidbbal.GetNaturalLanguageRulesForSqlNodeTypeNnt(nnToken);

         if (ruleIdNnt != 0) // IF A VALID RULE EXIST
         {
          // GET THE TOKEN ID
          nnTokenId = nlidbbal.GetTokenIdInQuestion(questionId, nnToken);
          // SAVE TOKEN AND NATURAL LANGUAGE RULE TO DATABASE - THE NATURAL LANGUGAE
RULE IS MAPPED TO THE SQL NODE TYPE. THEREFORE THE TOKEN IS MAPPED TO THE SQL NODE NNT
          nlidbbal.AddToTokensMappedToSqlNode(nnTokenId, ruleIdNnt);
         }

         // GET THE NATURAL LANGUAGE RULE FOR NAME NODE VIEW
         ruleIdNnv = nlidbbal.GetNaturalLanguageRulesForSqlNodeTypeNnv(nnToken);
```

```
        if (ruleIdNnv != 0) // IF A VALID RULE EXIST
        {
         // GET THE TOKEN ID
         nnTokenId = nlidbbal.GetTokenIdInQuestion(questionId, nnToken);
         // SAVE TOKEN AND NATURAL LANGUAGE RULE TO DATABASE - THE NATURAL LANGUGAE
RULE IS MAPPED TO THE SQL NODE TYPE. THEREFORE THE TOKEN IS MAPPED TO THE SQL NODE NNV
         nlidbbal.AddToTokensMappedToSqlNode(nnTokenId, ruleIdNnv);
        }
       }

       // GET THE COUNT OF TOKENS MAPPED TO NATURAL LANGUAGE RULES - THIS WILL ONLY
EXTRACT NNT AND NNV TYPES
       int countOfTokensMappedToSqlNodes =
nlidbbal.CountTokenToSqlNodeMappingForQuestion(questionId);
       // IF THERE IS ONLY ONE TOKEN MAPPED TO A TABLE OR VIEW
       if (countOfTokensMappedToSqlNodes == 1) {
        // CHECK FOR REQUESTED COLUMN NAMES BASED ON THE TOKEN NNT OR NNV
        // GET THE NNT OR NNV TOKEN
        string token =
nlidbbal.GetNntOrNnvTokensMappedToSqlNodesDetailsForQuestion(questionId);

        List < string > dependentTokens = new List < string > ();
        // GET THE TOKENS DEPENDING ON THE NNT OR NNV TOKEN
        dependentTokens = nlidbbal.GetDependenciesOnTokens(questionId, token);

        // FOR ALL DEPENDENT TOKENS CHECK IF THERE ARE NATURAL LANGUAGE RULES DEFINED
FOR SQL NODE TYPE NNC (NAME NODE COLUMN)
        foreach(var dependentToken in dependentTokens) {
         // GET THE NATURAL LANGUAGE RULE FOR NAME NODE VIEW
         ruleIdNnc = nlidbbal.GetNaturalLanguageRulesForSqlNodeTypeNnc(dependentToken);

         if (ruleIdNnc != 0) // IF A VALID RULE EXIST
         {
          // GET THE TOKEN ID
          nnTokenId = nlidbbal.GetTokenIdInQuestion(questionId, dependentToken);
          // SAVE TOKEN AND NATURAL LANGUAGE RULE TO DATABASE - THE NATURAL LANGUGAE
RULE IS MAPPED TO THE SQL NODE TYPE. THEREFORE THE TOKEN IS MAPPED TO THE SQL NODE NNC
          nlidbbal.AddToTokensMappedToSqlNode(nnTokenId, ruleIdNnc);
         }
        }

        // GENERATE THE SQL STATEMENT FOR THE QUESTION IN NATURAL LANGUAGE
        string sqlStatement = nlidbbal.GenerateSqlStatement(questionId);

        nlidbbal.AddNewSqlQuestionToKB(questionId, sqlStatement);
        Console.WriteLine(sqlStatement);
       } else if (countOfTokensMappedToSqlNodes == 0) // THERE ARE NO TOKENS MAPPED TO
TABLES OR VIEWS
       {
        Console.WriteLine(nlidbbal.UserMessageNoTablesOrViewImplementation);
        nlidbl.PrintMessage(nlidbbal.UserMessageNoTablesOrViewImplementation);
       } else // THERE ARE MULTIPLE TOKENS MAPPED TO TABLES OR VIEWS
       {
        Console.WriteLine(nlidbbal.UserMessageMultipleTablesOrViewImplementation);
        nlidbl.PrintMessage(nlidbbal.UserMessageMultipleTablesOrViewImplementation);
       }
      }
     } else // THE NUMBER OF SENTENCES HAS EXCEEDED THE PREDEFINED MAX_SENTENSES
     {
```

```
      string warningMessage =
nlidbbal.GetWarningMessage_NumberOfSentenses(sentencesCount);
      Console.WriteLine(warningMessage);
      nlidbl.PrintMessage(warningMessage);
     }
    }
   } else // INPUT STRING IS EMPTY
   {
    Console.WriteLine(nlidbl.MessageEmptyString);
    nlidbl.PrintMessage(nlidbl.MessageEmptyString);
   }

   Console.ReadLine();
  }
 }
}
```

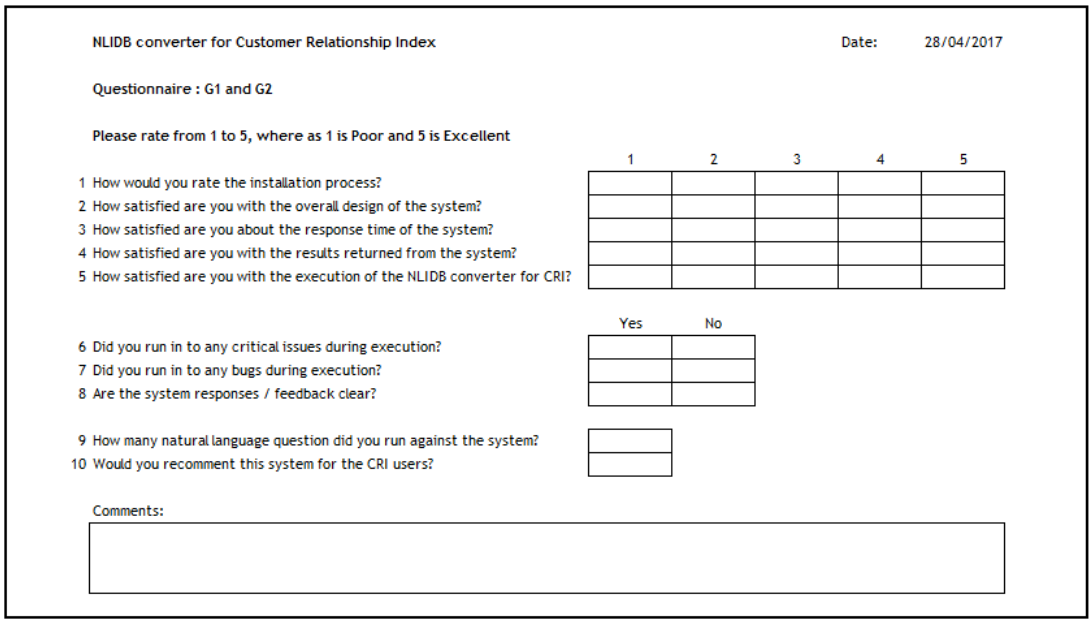
APPENDIX B – Stanford CoreNLP Annotators

Property name	Annotator class name	Description
tokenize	TokenizerAnnotator	Tokenizes the text.
ssplit	WordsToSentencesAnnotator	Splits a sequence of tokens into sentences.
pos	POSTaggerAnnotator	Labels tokens with their POS tag.
lemma	MorphaAnnotator	Generates the word lemmas for all tokens in the corpus.
ner	NERClassifierCombiner	Recognizes named (PERSON, LOCATION, ORGANIZATION, MISC), numerical (MONEY, NUMBER, ORDINAL, PERCENT), and temporal (DATE, TIME, DURATION, SET) entities.
parse	ParserAnnotator	Provides full syntactic analysis, using both the constituent and the dependency representations. The constituent-based output is saved in TreeAnnotation. The system generate three dependency-based outputs, as follows: basic, uncollapsed dependencies, saved in BasicDependenciesAnnotation; collapsed dependencies saved in CollapsedDependenciesAnnotation; and collapsed dependencies with processed coordinations, in

		CollapsedCCProcessedDependenciesAnnotation.
depparse	DependencyParseAnnotator	Provides a fast syntactic dependency parser. The system three dependency-based outputs, as follows: basic, uncollapsed dependencies, saved in BasicDependenciesAnnotation; collapsed dependencies saved in CollapsedDependenciesAnnotation; and collapsed dependencies with processed coordinations, in CollapsedCCProcessedDependenciesAnnotation. Most users of our parser will prefer the latter representation. For details about the dependency software, see this page. For more details about dependency parsing in general, see this page.
dcoref	DeterministicCorefAnnotator	Implements both pronominal and nominal coreference resolution. The entire coreference graph (with head words of mentions as nodes) is saved in CorefChainAnnotation.

APPENDIX C – Questionnaire

The questionnaire submitted to G1: Brandix Apparel Solution LTD – Essentials – Software Team and G2: Brandix Apparel Solution LTD – Essentials – Software Support Team



NLIDB converter for Customer Relationship Index Date: 28/04/2017

Questionnaire : G1 and G2

Please rate from 1 to 5, where as 1 is Poor and 5 is Excellent

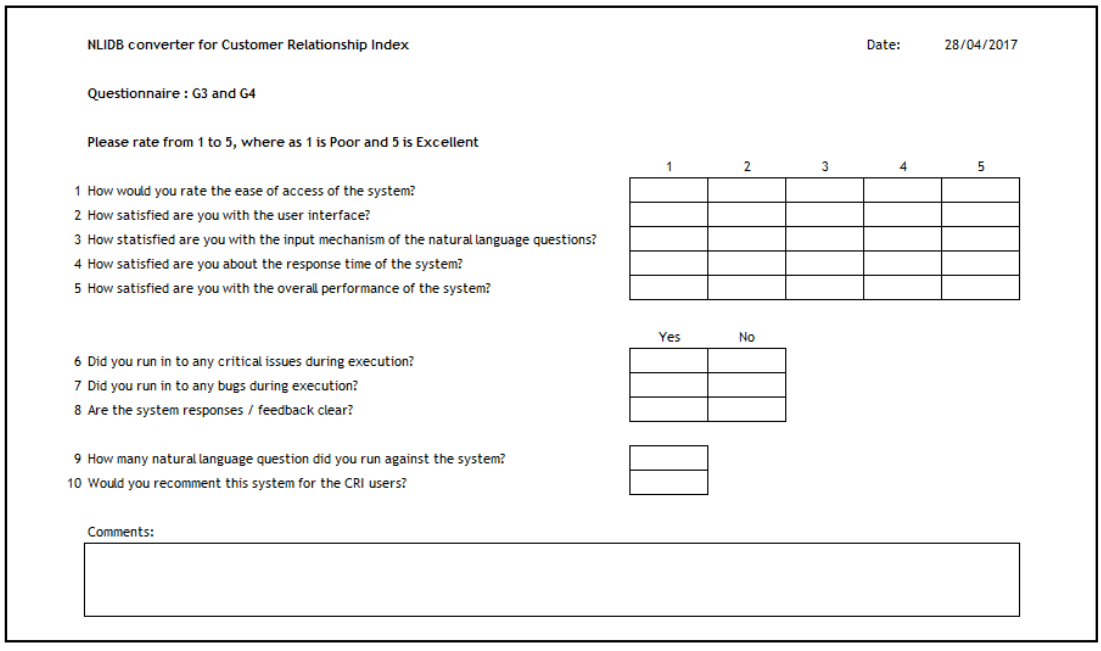
	1	2	3	4	5
1 How would you rate the installation process?					
2 How satisfied are you with the overall design of the system?					
3 How satisfied are you about the response time of the system?					
4 How satisfied are you with the results returned from the system?					
5 How satisfied are you with the execution of the NLIDB converter for CRI?					

	Yes	No
6 Did you run in to any critical issues during execution?		
7 Did you run in to any bugs during execution?		
8 Are the system responses / feedback clear?		

9 How many natural language question did you run against the system?	
10 Would you recomment this system for the CRI users?	

Comments:

The questionnaire submitted to G3: Customer Relationship Index User and G4: Non Customer Relationship Index Users.



NLIDB converter for Customer Relationship Index Date: 28/04/2017

Questionnaire : G3 and G4

Please rate from 1 to 5, where as 1 is Poor and 5 is Excellent

	1	2	3	4	5
1 How would you rate the ease of access of the system?					
2 How satisfied are you with the user interface?					
3 How statisfied are you with the input mechanism of the natural language questions?					
4 How satisfied are you about the response time of the system?					
5 How satisfied are you with the overall performance of the system?					

	Yes	No
6 Did you run in to any critical issues during execution?		
7 Did you run in to any bugs during execution?		
8 Are the system responses / feedback clear?		

9 How many natural language question did you run against the system?	
10 Would you recomment this system for the CRI users?	

Comments: