Addis Ababa University

College of Natural Science

Accessing Databases Using Amharic Natural Language

Beniyam Legesse

A Thesis Submitted to the Department of Computer Science in Partial Fulfilment for the Degree of Master of Science in Computer Science

Addis Ababa, Ethiopia

October 2020

Addis Ababa University

College of Natural Science

Beniyam Legesse

Advisor: Yaregal Assabie (Ph.D.)

This is to certify that the thesis prepared by Beniyam Legesse, titled: *Accessing Databases Using Amharic Natural Language* and submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science complies with the regulation of the university and meets the accepted standards with respect to originality and quantity.

Signed by Examining Committee:

Name	Signature	Date
1. Advisor: Dr.Yaregal Assabie	_____	
2. Examiner: Dr.Solomon Atnafu	_____	
3. Examiner: Dr.Minale Ashagre	_____	

Abstract

Nowadays, day to day activities of human beings is highly dependent on the information distributed in every part of the world. One major source of the information, which is the collection of related data, is the database. To extract the information from the database, it is required to formulate a structured query language which is understood by the database engine. The SQL query is not known by everyone in the world as it requires studying and remembering its syntax and semantics. Only professionals who study the SQL can formulate the query to access the database. On the other hand, human beings communicate with each other using natural language. It would be easier to access the content of the database using that natural language which in turn contributes to the field of natural language interface to the database. Since in many private and public organizations, peoples are performing day to day activities in Amharic language and many of them are not skilled in formulating structured query language, it would be better if there is a mechanism by which the users can directly extract information from the database using the Amharic language.

This research accepts questions that are written in Amharic natural language and converts to its equivalent structured query language. A dataset which consists of an input word that is tagged with the appropriate output variable is prepared. Features which represent the Amharic questions are identified and given to the classifier for training purpose. Stemmer, Morphological analyzer, and pre-processor prepare the input question in the format required by the classifier. To identify appropriate query elements, Query Element Identifier uses the dictionary which is prepared by applying the concept of semantic free grammar. The query constructor constructs the required SQL query using these identified query elements. A prototype called Amharic Database querying system is developed to demonstrate the idea raised by this research. Testers from different departments having different mother tongue language test the performance of the system.

Keywords: Natural language interface to the database, Stemmer, Morphological analyzer, pre-processor, Query Element Identifier, Semantic free grammar, Query constructor

Acknowledgment

My GOD and your mother, **Saint Virgin Mary**, I don't have an equivalent word to thank you for what you have done to me. Thank you very much for giving me such a chance and wisdom to accomplish this research.

Next to this, it is a great pleasure for me to acknowledge the assistance and contribution done by many individuals to this research. My first thank goes to my advisor, Dr.Yaragal Assabie who showed me a direction on how to finish the research and for his continuous feedback. Your comment, encourage and commitments are unbelievable. I would like to thank the support, encouragement, and patience of my family. My friends especially, Tsiyon Belayneh, it would be difficult for me to reach to this stage without your continuous advice and assistance. Thank you very much.

I would like to repeat my thanks to the Employees of Ethiopian and Shipping Logistics Service Enterprise. Especially IT department employees who gave me information and materials related to the database that is used for the prototype development of this research. My many thanks go to Employees of Ethiopian and Shipping Logistic service enterprise who involved in testing the performance of the ADQS system apart from their work.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Acronyms and Abbreviations

ADQS	Amharic Database Querying System
ASMA	Amharic Stems Morphological Analyzer
ANSI	American National Standard Institute
DBMS	Database Management System
DDL	Data Definition Language
DML	Data Manipulation language
FAM	File Access Manager
IDA	Intelligent Data Access
INLAND	Informal Natural Language Access to Navy Data
ISO	International Standards Organization
LADDER	Language Access to Distributed Data with Error Recovery
NALIX	Natural Language Interface to XML
NLIDB	Natural Language Interface To The Database
NLP	Natural Language Processing
RDMS	Relational Databases Management system
SEQUEL	Structured English Query Language
SES	Shipping Expert system
SGR	Semantic Grammar Rules
SQL	Structured Query Language
SVM	Support Vector Machine
WASP	Word Alignment based Semantic Parsing

Chapter One: Introduction

This chapter deals with the introduction of the natural language interface to database. The situations or reasons that motivate the Authors to write this Master's thesis will come next. Also, the gaps that this Master's research is going to fill, and the objective will be described. An overview of the methodology will be presented. The scope and limitation of the research and finally the application of the result will be described in sequence.

1.1 Background

Nowadays, day to day activities of human beings is highly dependent on the information distributed in every part of the world. Users are searching for information from various sources. One of the major sources of information that every application dependant on is the database. A database is a collection of related data. The data represents known facts which have an implicit meaning [1]. Thus, every database is a collection of coherent (related) data that represents certain aspects of the real world. It is designed, built, and populated to address a specific situation in the real world.

To get information from the database, knowledge of database languages is required. The languages can be Procedural or Structural query language. A procedural query language is consisting of a set of operations that take one or two relations as input and produce a new relation as an output. Relational Algebra can be taken as an example of Procedural language. The algebra operations enable a user to retrieve a specific request on a relational model. The operations that produce a new relationship can be further manipulated using operations of the relational algebra. The sequence of the relational algebra that produces new relation forms a relational algebra expression [1].

On the other hand, SQL is a query language that is standardized by the American national standard institute for most commercial relational database management system. The Users execute queries they are also called SQL statements to pull or modify the requested information from the database. SQL supports data definition and data manipulation language.

The data definition language is consisting of the structured query language that can be used to define the database schema. It is used to create and modify the structure of the database object in the database. This includes queries like create, drop, alter, truncate, etc. The data

manipulation language is the SQL command that deals with the manipulation of data present in the database and this includes most of the SQL statements. It is used to access, modify, or retrieve the data from the database. An example of a DML query includes select, update, and delete.

The natural language also called human language is a language used by humans naturally to communicate with each other. It is expressed as a text by the alphabet or letters of the human language. Natural language interface to the database 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, Amharic) [2]. If users want to access information from the database, they are required to know structured query language/SQL queries, and they are required to know the physical structure of the database. The natural language interface to the database minimizes these difficulties and enables the users to extract information from the databases using natural languages.

In recent times, there are rising demands for non-expert users to query a relational database in more natural languages. An advancement of Natural language Interface to the database which is an important step towards the development of an intelligent database attracts the attention of many researchers. Many types of research [3] have been done on Natural language interfaces to the database for different natural languages and satisfactory results have been achieved.

1.2 Motivation

The authors were motivated to write this Master's proposal because the problem currently exists in Ethiopia which has more than 100 million populations [5]. In Ethiopia, many people speak the Amharic language and perform the daily activities using this language.

The need for the Amharic language for this Master's research is because the Amharic language is a working language of the federal government of Ethiopia and many people in private and public organizations are using this language to perform many tasks in the office. Additionally, there is a growing demand for developing applications using the Amharic language interface. Many applications have been developed in Ethiopia whose front and back ends involve Amharic content.

In many private and public organizations, many peoples do not know how to formulate SQL to directly access the database. Many organizations are automating their manual work and several databases have been designed with Amharic content. Lots of information might be asked using Amharic questions. One of such questions written in the Amharic language that might be raised by the naïve users could be "በእያንዳንዱ የስራ ክፍል ውስጥ ያለውን ከፍተኛ ደሞዝ አሳየኝ/ b*äaəyanədanədu yäsəra kəfələ wəsət̀ə yaläwənə käfətäġa dämozə asayägə*" which can be translated as for each department, give me the highest salary.

In many private and public organizations, people are performing day to day activities in Amharic language and many of them are not skilled in formulating structured query language. Thus it would be better if there is a mechanism by which the users can directly extract information from the database using the Amharic language. Thus, the authors were motivated to provide a better alternative so that the person who knows only Amharic natural language can access the database.

1.3 Statement of the problem

Despite its Importance, writing structured query language creates an additional burden on non-English speakers to fetch some information directly from the database. In the first place, knowing the English language is advantageous to learn SQL itself. This is because the SQL syntax itself is developed using English language characters. Secondly, the users have to study the syntax and semantics of structured query language to write efficient queries. Additionally, they are required to know the physical structure and construction of the database. This might not be enough to become an expert on the database. It may need more understanding and practice through various exercises.

Many researchers have studied and applied various techniques to minimize the above steps. The researchers [5, 6, 7, 8, 9, 10] have developed a system that converts relevant questions written in English Natural language into structured query language. Some of them used rule-based techniques and some other used machine learning to model the system. Other researchers [11, 12, 13, 14] applied the same concept to different languages other than English. Since the analysis and conversion process is language-dependent for different languages, it is impossible to equivalently implement those methods on the Amharic Language.

Accordingly, Tihitina Petros [15] has studied on Amharic language interface to the database and satisfactory result has been achieved. The research uses a set of rules to convert the Amharic statements into corresponding English SQL. The system accepts simple relevant Amharic statements and converts into structured query language. However, the research was limited to use a certain set of rules which restrict the users only to input pre-defined words. When the users want to use the system, first they have to select the database that they want to extract information from it. Then, they have to use the table names and attributes defined in that list. The users are involved in the selection of the table names, attributes, and conditions. Additionally, it only considered simple select query of Data Manipulation Languages (DML) queries and other DML queries like delete, update, complex queries which involve like group by, Order by, aggregate function, and so on are not considered.

As compared to other languages, the Natural language interface to the database for the Amharic language lags behind other languages. In other languages, the Natural language interface to the database has been developed by many researchers using various techniques and a good result has been achieved [8, 9, 10]. In Amharic language Interface to the database, up to the Authors' best knowledge, very few researches [15] have been done so far. Thus this research aims to fill the gaps observed in the above researches [15] in such a way that the users can express the questions using free Amharic text. The system will take care of identifying the table names, attributes, and relationships from the Amharic questions itself. Other types of queries that are not covered in the previous work will also be considered in this research. The research is specifically designed for converting questions written in the Amharic language to its equivalent SQL query using Machine learning. The benefit from this proposal is not only used for naïve users but it can also be extended for other technical users. This is because it is easy to handle negation, quantification type of questions, and some complex ideas using natural languages than structured query languages [12].

1.4 Objective

General Objective

The general objective of this Master's research is to develop an Amharic Natural language interface for accessing databases.

Specific Objectives

Thus the specific objective of this Master's research is to

- Review related works on Natural language interface to the database, artificial intelligence, human-computer interaction
- Collect Amharic relevant questions which has a connection with the database
- Design a system architecture
- Develop algorithms that will convert relevant Amharic statements into SQL
- Develop a prototype
- Test the performance of the system

1.5 Methodology

Since this Master's research emphasizes systematic, testable, and communicable methods, the research will be carried out following the design science research principle. Design Science Research creates and evaluates IT artefacts intended to solve the identified organizational problems [16].

Literature Review

For the effective development of this Master's research, a review of related work that has been done on the Natural language interface to the database by other researchers is very essential. Thus, review of related work will be carried out first for both Amharic and other languages.

Data collection

Amharic relevant questions from Amharic speakers and books will be collected and analyzed to train the classifier.

Prototype Development

For the demonstration of this Master's research, a prototype will be developed using suitable programming language. Appropriate Machine learning algorithm will be selected to train the classifier.

Evaluation

After the prototype development, the prototype will be tested by both naïve and IT users. The result generated by the system will be compared with the expected output produced by the normal SQL query. Its performance will be expressed in terms of accuracy. From the evaluation, the result and discussion will be presented.

1.6 Scope and Limitation

This Master's research is only prepared to convert Amharic written questions into structured query language by accepting Amharic text as input. Other languages are behind the scope of this research. The DDL queries, Joins, and sub queries are not handled in the research. The research will not make an Analysis of the values that are stored in the table. The users are expected to write a complete sentence that has a complete meaning. Spelling checking is also out of the scope of the research.

1.7 Application of the result

The result from this Master's research will allow many naïve users to easily extract information from the database using Amharic natural language without the help of any other supplementary resource. The naïve users don't need to know the English language, the structured query language, and the physical structure of the database. The only thing that they require for this research is the Amharic natural language.

Equivalently the research also helps other educated users to access the database using natural language. Sometimes it is very difficult to remember and memorize all syntax of the structured query language. Additionally, even if you are familiar with SQL, some queries

are somehow complex to write it. The users know what they want to extract from the database, but expressing those ideas in SQL might be difficult. This research will avoid these difficulties and makes communication with the database to be more convenient and easy.

This research can also serve as an interface for other NLP applications so that they can automatically access the database using Amharic natural languages without knowing the internal structure of the database.

1.8 Organization of the Rest of the Thesis

The remaining part of the thesis is organized in such a way to include the literature review, related works, Design and implementation, Experiment, and finally conclusion and recommendation. The background knowledge that is required for a better understanding of the natural language interface to the database will be presented in the literature review. The related work will discuss a survey of scholarly sources that are related to Accessing database contents using different natural languages. The algorithm and any design-related issues that are used for the prototype development will be presented in the Design and implementation. The experiment that has been conducted to test the performance of the thesis will be discussed in the experiment. Finally, the summary and future works will be the last chapter of this thesis.

Chapter Two: Literature Review

In this chapter, a survey of scholarly sources that are related to Database accessing using Amharic natural language will be presented. These sources include books, published and unpublished researches, journals, manuals, documents taken from the internet, surveys. These provide an overview of current knowledge; identify relevant theories, methods, and gaps in the existing research [17].

2.1 Natural Language Interface to the Database

As the name implies, the natural language interface to the database has three major components that are integrated. These are the database, the natural language, and finally their interface. A database is a collection of related data. By data, we mean known facts that can be recorded and that have implicit meaning [1]. This database comes into use by the database management system. The database management system is the software that can be used to manage this interrelated data by storing in the databases and retrieving it from the database. Thus Database management system is a collection of interrelated data and a set of programs that are used to access these data. The database management system is designed to manage and maintain large bodies of information [18].

The second component is the natural language which is a language that has developed naturally in use as contrasted with an artificial language. It is evolved naturally in humans through use and repetition without conscious planning or remediation. To use this natural language systematically, the processing concept comes next. Thus Natural language processing is an integral area of computer science in which machine learning and computational linguistics are broadly used. The main objective of this natural language processing is to make computers understand, interpret, and manipulate human language. The task that is performed by the natural language processing is to take chunks of information as input in the form of voice or text or both and manipulate them as per the algorithm inside the computer. This field is mainly concerned with making the human and computer interaction easy but efficient [19].

The last component is the interface that exists between the natural language and the database which enables accessing information from the database using the natural language. The

purpose of natural language interfaces is to allow users to compose questions in natural language and to receive responses under the form of tables or short answers [20]. The database system is designed in such a way that its content is stored in a structured manner and its content is accessed using artificial languages. The natural language is spoken naturally. There is no way of accessing the content of the database directly using this natural language. Thus the interface is needed to fill this gap. The above all three components are integrated and lead to a natural language interface to the database.

The main reason that NLIDB required is to directly access the content of the database using the natural language. This is because to access the information stored in the database, knowledge of Structured Query Language is needed. Only those users who know SQL languages can access the data or information. An end-user normally doesn't know SQL [18, 21, 22]. In the NLIDB, the users naturally write the questions, the NLIDB understands the question, accesses the content of the database, and returns the response to the users.

2.1.1 History of Natural language Interface to the Database

The history of the Natural language Interface to the database was begun in 1961 when Baseball was invented [23]. Baseball is a computer program that answers questions phrased in ordinary English about stored data. The program reads the question from punched cards. It was implemented to answer questions about baseball games which were played in the American League during one season. The data that is stored in the Baseball are the month, day, place, teams, and scores for each game in the American League for one year.

The next NLIDB application is Lunar which was developed in 1972 to allow lunar scientists to ask questions, compute averages and ratios, and make selective listings based on information in a chemical analysis table [24]. The lunar scientist's natural language information system is a prototype computer system that allows English language access to a large database of lunar sample information. It uses two database files; one is a 13,000 line table of chemical and age analysis of the Apollo 11, and the second, the key phase index to those reports. Apollo 11 was the spaceflight that first landed humans on the Moon.

In 1975, the next NLIDB application called Planes was developed as a prototype to access the information from a large database of aircraft flight and maintenance. The model was

presented for processing English request for information from a relational database [25]. The database is comprised of detailed flight and maintenance data and data summaries. It is organized by month, aircraft, and type of data (scheduled maintenance, unscheduled maintenance, flights).

The NLIDB application called LADDER [26] was developed in 1978. It was developed to make large, distributed databases directly available to decision-makers. The user provides a question about the information base in English; LADDER applies all the necessary information concerning the vocabulary and syntax of the question, the names of specific fields, how they are formatted, how they are structured into files, and even where the files are physically located, to provide an answer. It consists of three major components that provide levels of buffering of the user from the underlying DBMSs. The first component, INLAND, accepts questions in a restricted subset of natural language. The second component, IDA breaks down a query against the entire very large database into a sequence of queries against individual files. The third component, called FAM relies on a locally stored model showing where files are located throughout the disk.

Consequently, several NLIDB based research has been conducted with different approaches and efficiencies. Especially in the mid of 1980s, Natural language interface to the database had become a very popular research area, and numerous research prototypes were implemented [18]. The CHAT-8 [27] provides a strategy for planning a query so that it can be efficiently executed by the elementary deductive mechanism which was developed using Prolog programming language. Dialogic [28] parses English sentences and translates them into logical forms. Diagram [29] is a grammar used in an artificial intelligence system for interpreting dialogue. TELI [30] which is an English-Language Interface that answers English questions about tabular (first normal-form) data files and runs on a Symbolic Lisp Machine. Janus [31] is a natural language interface that translates intentional logic expressions representing the meaning of a request into executable code for each application program. These all are research prototypes that are developed in the mid of 1980s.

Coming to some of the recent NLIDB applications [32], NALIX which is a generic interactive natural language query interface to an XML database was developed in 2006. The application accepts an arbitrary English language sentence as query input then the query

is translated into an XQuery expression that can be evaluated against an XML database after reformulation. The application used an off-the-shelf natural language parser and interactively guides the user to pose queries that the system can understand by providing meaningful feedback and helpful rephrasing suggestions. Three main steps which are Parse Tree Classification, Parse tree Validation, Parse tree translation are involved in the translation of natural language queries into corresponding XQuery expressions [33].

The PRECISE [34] is another recent NLIDB application which uses a statistical parser as a "plugin". The application targets the database in the form of a relational database using SQL as the query language. PRECISE takes a lexicon and a parser as input. Then, given an English question, PRECISE maps it to one (or more) corresponding SQL queries. The WASP [35] is also another recent NLIDB application. It uses a statistical approach to semantic parsing for constructing a complete, formal meaning representation of a sentence. The semantic parser was learned by a set of sentences annotated with a correct formal query language.

2.1.2 Components of NLIDB

Many of the computing scientists divide the component of natural language interface to the databases into Linguistic and Database components [21]. The linguistic component is responsible to translate the natural language into a formal query and then to generate a natural language response based on the result which comes from the database search. This occurs usually at the beginning and end of the process.

The database component performs traditional Database Management functions [21]. It takes the query translated by the linguistics component and executes the query. The natural language generator takes the formal response as its input and inspects the parse tree to generate adequate natural language response. Natural language database systems make use of syntactic knowledge and knowledge about the actual database to properly relate natural language input to the structure and contents of that database. Syntactic knowledge usually resides in the linguistic component of the system, in particular in the syntax analyzer whereas knowledge about the actual database resides to some extent in the semantic data model used.

2.1.3 Advantages of NLIDB

The natural language interface to the database allows users to communicate with the database using the natural language. By doing so, the NLIDB provides the following advantages [18].

One of the advantages that NLIDB provides is the no need requirements of Artificial Language. Since artificial languages like SQL are difficult to learn and master at least by non-computer specialists, the NLIDB avoids the requirement to learn artificial languages. The users are expected to use the native language to communicate with the database. This will also remove the time that will be spent in learning a formal database query language.

The next advantage is the simplest and Easiness to use of NLIDB in the real life. The users are not expected to remember the syntax and semantics of the formal query as they type the query using native languages. They can express the ideas to the computer in the same way as they express to other human beings.

NLIDB is also better for some questions. It has been argued that there is some kind of questions (e.g. questions involving negation, or quantification) that can be easily expressed in natural language, but that seems difficult (or at least tedious) to express using graphical or form-based interfaces.

NLIDB also provides a fault tolerance capability. Most of NLIDB systems provide some tolerances to minor grammatical errors, while in a computer system; most of the time, the lexicon should be the same as defined, the syntax should correctly follow certain rules, and any errors will cause the input automatically be rejected by the system

Additionally, NLIDB is easy to use for multiple database tables. Queries that involve multiple database tables like “list the names of the employees whose salary is greater than 10,000 dollars”, are difficult to form in the graphical user interface as compared to the natural language interface.

2.1.4 Disadvantages of NLIDB

In addition to its advantages', there are some drawbacks that the NLIDB provides. To mention a few of them, the following are described [21]. One disadvantage of NLIDB is it creates false Expectation. Users can be misled by an NLIDB system's ability to process a natural language. Sometimes they may assume that the system is intelligent. Therefore, rather than asking precise questions from a database, they may be tempted to ask questions that involve complex ideas, certain judgments, reasoning capabilities, etc., which an NLIDB system cannot be relied upon.

Conceptual failures are also another disadvantage of NLIDB. The NLIDB accepts the user question and generates the output back to the user based on the query transformed from the user. The query is transformed based on the logic that the NLIDB understands the question. The output might not be conceptually correct. It is often the case that the system does not provide any explanation of what causes the system to fail conceptually.

2.1.5 Approaches of NLIDB

Since the NLIDB is a hot research area that attracts the intension of many researchers due to the implicit ambiguity that language possesses [20], various results with various approaches are achieved by different researchers. In this section techniques used by the different researchers will be described. The first approach that is used by the NLIDB is a Symbolic Approach; it is also called a rule-based approach. Among the approaches of NLIDB, the natural language interface to the database has been dominated by the symbolic approach for several decades [36]. This is because the natural languages itself are expressed with words which are symbols that stand for the objects and concepts in the real world. In this approach, language knowledge is explicitly encoded in rules or other forms of representation [20, 34]. The NLIDB tries to capture the meaning of the language based on these rules.

The approach to query analysis is based on the development and extension of the existing rules so that the NLIDB doesn't require a massive training corpus. To prepare hand-crafted rules, the rule-based approach requires skilled experts. The linguist or knowledge engineers are required to manually encode each rule of the NLIDB.

The next approach is the Empirical approach it is also called the corpus-based approach. Empirical approaches are based on statistical analysis as well as other data-driven analyses, of raw data which is in the form of text corpora [21]. A corpus is a collection of machine-readable text. In this approach, the corpora are annotated to study several linguistics phenomena. The approach has been around since NLP began in the early 1950s. By collecting such a corpus and computing the appropriate statistics, corpus-based representation offers an alternative to traditional knowledge representation for a broad class of applications.

Several researchers in corpus-based computational linguistics believe that the size of available annotated corpora is the current limiting factor in creating accurate corpus trained natural language processing system [37]. If this is the case, the cycle of automatically annotating a corpus, manually correcting it, and retraining the automatic annotator on the larger corpus could provide a fast mechanism for providing very large annotated corpora.

The last approach is called the connectionist approach which is carried using a neural network. The connectionist approaches to NLIDB make use of large networks of simple computational units that communicate utilizing simple quantitative signals. It uses interconnected simple processors called neurons to form a simplified model of the structures in the biological nervous system. Since human language capabilities are based on neural networks in the brain, Artificial Neural Networks (also called connectionist network) provides an essential starting point for modeling language processing [20].

Neural network approaches are essentially an extension of the empirical methods which involve a mathematically based assessment of complex inter-relationships within the system. In this approach, instead of symbols, the approach is based on distributed representations that correspond to statistical regularities in language

2.1.6 Techniques used in NLIDB

In the selected approach as described in the above section, the researchers used appropriate techniques to develop the natural language interface to the database. One of the techniques that are used in NLIDB is called pattern matching. The pattern matching is an algorithmic task that finds pre-determined patterns among sequences of raw data or processed tokens. In

contrast to pattern recognition, this task can only make exactly matched from an existing database and won't discover new patterns. In these techniques, a perceived sequence of tokens is checked for the presence of the constituents of some pattern. The patterns generally have the form of either sequences or tree structures. Many NLIDBs was based on pattern-matching techniques to answer the user's questions. As an example, consider the Table 2.1 which shows a sample database table [18].

Table 2.1 Sample database

Countries table		
Country	Capital	Language
France	Paris	French
Italy	Rome	Italian
India	Delhi	Hindi

A primitive pattern-matching system might use the following rules as

Pattern: …. "Capital" …. <Country>

Action: Report Capital and Country of a row where Country=<Country>

Pattern: …. "Capital" …. <Country>

Action: Report Capital and Country of a row where Country=<Country> of each row

Then if the user asked a question like "What is the capital city of France?" the system will report "Paris" using the first pattern rule. The same rule can be applied to answer questions, "kindly show me the capital city of France?", "give me the Capital of France?", "Could you please show me the capital of France?" etc.

The main advantage of the pattern-matching approach is its simplicity i.e. no elaborate parsing and interpretation modules are needed, and the systems are easy to implement [39]. Also, pattern-matching systems often manage to come up with some reasonable answer, even if the input is out of the range of sentences the patterns were designed to handle.

The second technique is called Syntax-base. The syntax is the study of the principles and processes by which sentences are constructed. It also refers to the rules and principles that govern the construction of a given sentence. In syntax-based systems, the user's question is parsed (i.e. analyzed syntactically) and the resulting parse tree is directly mapped to expression in some database query language. 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 [18].

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, how words can be grouped to form a phrase, how phrases can be grouped to form more complex phrases until a complete sentence is built. Having this information, the semantic meanings to certain production rules (or nodes in a parse tree) can be mapped.

The third technique that the NLIDB uses is the semantic grammar-based technique. Semantic analysis is the process of relating syntactic structures, from the levels of phrases, clauses, sentences, and paragraphs to the level of the writing as a whole, to their language-independent meanings. 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. Instead of smaller structures, the semantic grammar approach also provides a special way for assigning a name to a certain node in the tree, thus resulting in less ambiguity compared to the syntax-based techniques [21].

The semantic grammar characterizes the statements in terms of the concepts as opposed to the traditional syntactic structure. The language can be described by a set of grammar rules which can characterize each concept or relationship. As an example, consider concepts of measurements [38]. The concept of measurement requires a quantity to be measured and something to measure for the quantity like '20 students'. If a concept of measurement is applied in other situations, at least these two things are present. In Electrical Environment as an example, a measurement is typically expressed by giving the quantity followed by a preposition followed by the thing that specifies where to measure as "voltage across C2", "current through R1". These phrasing can be captured by the grammar rule as

<Measurement>: =< measurable/quantity> <preposition> <part>

This concept of measurement can be used in other sentences as one part of the sentence as "what is the voltage across C2", "Is the current through transistor R1 equals 2A".

Semantic grammars have two advantages over traditional syntactic grammars. They allow semantic constraints to be used to make predictions during the parsing process, and they provide a useful characterization of those sentences which the system should try to handle [38]. The main drawback of the 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.

The last technique is known as Intermediate Representation Language. It is very difficult to directly translate a sentence parsed by syntax-based techniques into general query language. Thus Intermediate representation of a language that maps the sentence into a logical query language is needed. The intermediate logical query expresses the meaning of the user's question in terms of high-level world concepts, which are independent of the database structure. The logical query is then translated to an expression in the database's query language, and evaluated against the database [21]. Figure 2.1 shows the possible architecture of the Intermediate representation system.

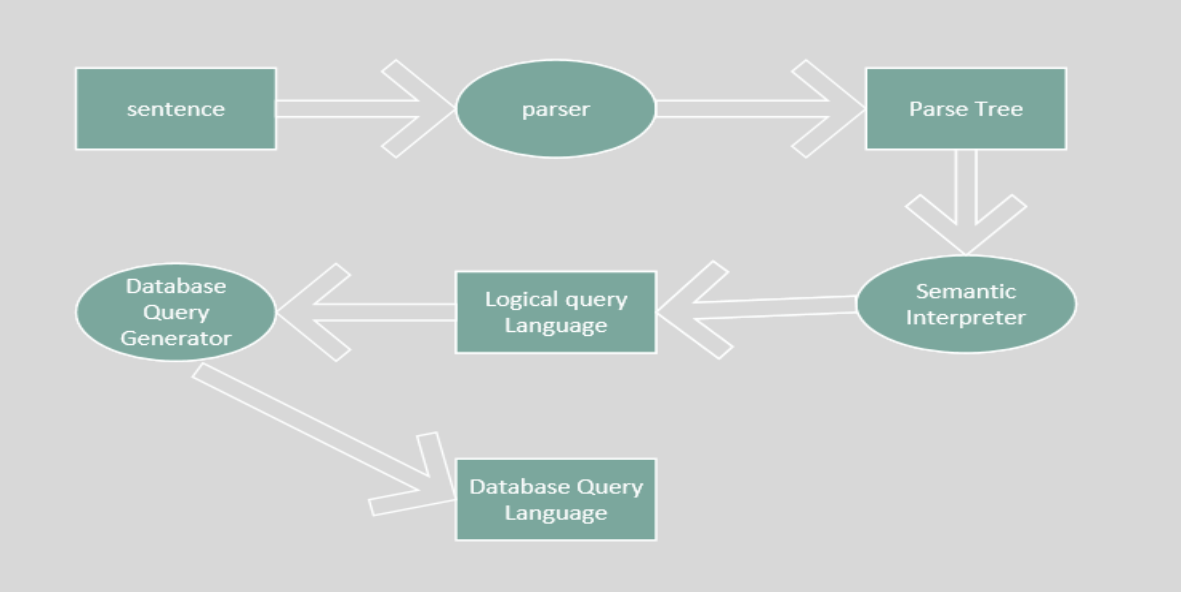


Figure 2.1 Intermediate Representation Language

In the intermediate representation language approach, the system can be divided into two parts. One part starts from a sentence up to the generation of a logical query. The other part starts from a logical query until the generation of a database query. In part one, the use of logic query languages makes it possible to add reasoning capabilities to the system by embedding the reasoning part inside a logic statement [21].

2.2 Decision Tree Classifier

2.2.1 Introduction

One of the approaches that are used in the natural language interface to the database is a machine learning approach. It uses an algorithm that learns from the training dataset and predicts the output variable. As the length of the training data increases; its predicting capability will also increase [39]. The main advantage of the machine learning approach is its flexibility, adaptability, and easy maintainability. Handcrafting rules are not required to maintain and adapt to a new environment when something has changed.

One of the classifier algorithms that are used in the machine learning approach is the Decision tree classifier. A decision tree is a classifier expressed as a recursive partition of the instance space. The decision tree consists of nodes that form a rooted tree called “root” that has no incoming edges. All other nodes have exactly one incoming edge. A node with outgoing edges is called an internal or test node. All other nodes are called leaves also known as terminal or decision nodes. In a decision tree, each internal node splits the instance space into two or more sub-spaces according to a certain discrete function of the input attributes values [40].

Each leaf is assigned to one class representing the most appropriate target value. Alternatively, the leaf may hold a probability vector indicating the probability of the target attribute that has a certain value. Instances are classified by navigating them from the root of the tree down to a leaf, according to the outcome of the tests along the path. Figure 2.2 shows an example of a decision tree that predicts whether a given creature is a Mammal or Non-Mammal.

Decision tree induction is closely related to rule induction. Each path from the root of a decision tree to one of its leaves can be transformed into a rule simply by conjoining the tests along the path to form the antecedent part and taking the leaf’s class prediction as the

class value. For example, one of the paths in Figure 2.2 can be transformed into the rule: "If the body temperature of a creature is warm, and if it gives Birth, then the creature is Mammal".

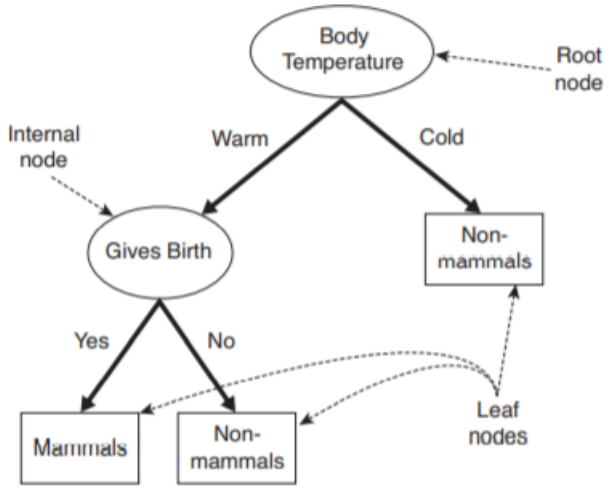


Figure 2.2 Decision Tree for the mammal classification problem

A classification technique (or Classifier) is a systematic approach to building classification models from an input data set. The decision tree classifier employs a learning algorithm to identify a model that best fits the relationship between the attribute set and the class label of the input data. The model generated by a learning algorithm should both fit the input data well and correctly predict the class labels of records it has never seen before. Therefore, a key objective of the learning algorithm is to build models with good generalization capability; i.e. the models that accurately predict the class labels of unknown records [41].

The general approach for solving a classification problem is shown in Figure 2.3. As you can see, the training data set which consisting of records whose class labels are known must be provided in the first place. The training set is used to build the classification model, which is subsequently applied to the test set, which consists of records with unknown class labels.

A classification model can be treated as a black box that automatically assigns a class label when presented with the attribute sets of an unknown record. Classification techniques are

most suited for predicting or describing a data set with binary or nominal categories. They are less effective for ordinal categories.

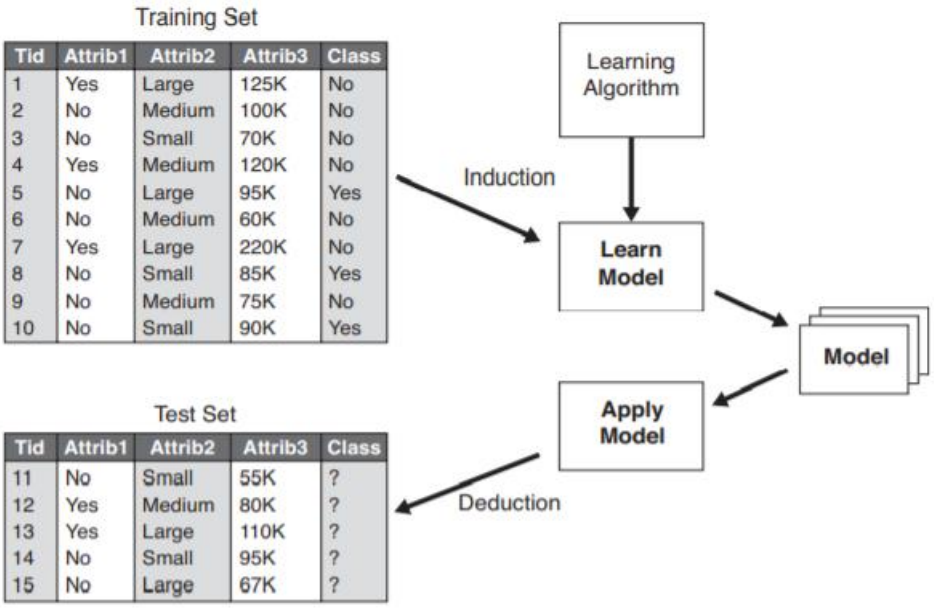


Figure 2.3 General approach for building a classification model

2.2.2 How Decision tree built

Decision tree inducers are algorithms that automatically construct a decision tree from a given dataset. In principle, there are exponentially many decision trees that can be constructed from a given set of attributes. Typically, the goal is to find the optimal decision tree by minimizing the generalization error. Efficient algorithms have been developed to induce a reasonably accurate, suboptimal decision tree in a reasonable amount of time. These algorithms usually employ a greedy strategy that grows a decision tree by making a series of locally optimum decisions about which attribute to use for partitioning the data. One such algorithm is Hunt's algorithm, which is the basis of much existing decision tree classifier algorithm including ID3, C4.5, and CART [41].

In hunt's algorithm, a decision tree is grown recursively by partitioning the training records into successive subsets. Let D_t be the set of training records that are associated with node t and $y = \{y_1, y_2, \ldots, y_c\}$ be the class labels. The following is a recursive definition of Hunt's algorithm.

Step 1: If all records in D_t belongs to the same class y_t, then t is a leaf node labeled as y_t

Step 2: if D_t contains a record that belongs more than one class, an attribute test condition is selected to partition the records into a smaller subset. A child node is created for each outcome of the test conditions and the records in D_t are distributed to the children based on the outcome. The algorithm is recursively applied to each child node.

A learning algorithm for inducing decision trees must address the splitting of the training records and the endpoint that stops the splitting procedure. A decision tree induction algorithm needed to provide a method for expressing attribute test conditions and its corresponding outcome for different attribute types. The common test conditions are Binary attributes, Nominal attributes, Ordinal attributes, and continuous attributes.

Many measures can be used to determine the best way to split the records. These measures are defined in terms of the class distribution of the records before and after splitting. Let p(i|t) denotes the fraction of records belonging to class I at a given node t. The measures developed for selecting the best split are often based on the degree of impurity of the child nodes. The smaller the degree of the impurity, the more skewed the class distribution. For example, a node with a class distribution (0, 1) has zero impurity, whereas a node with a uniform class distribution (0.5, 0.5) has the highest impurity.

Some example of impurity measures includes Entropy, Gini, and classification error as shown in formula 2.1, 2.2, and 2.3. Where c represents the number of classes

$$\text{Entropy (t)} = -\sum_{i=0}^{c-1} \text{p(i|t)log2 p(i|t)} \quad \text{(2.1)}$$

$$\text{Gini (t)} = 1 - \sum_{i=0}^{c-1} [p(i|t)]^2 \quad \text{(2.2)}$$

$$\text{Classification Error (t)} = 1 - \max_i [p(i|t)] \quad \text{(2.3)}$$

One advantage of the Decision tree classifiers is they are self–explanatory and when compacted they are also easy to follow. In other words, if the decision tree has a reasonable number of leaves, it can be grasped by non–professional users. Furthermore, decision trees can be converted to a set of rules. Thus, this representation is considered as comprehensible.

The next advantage is that it can handle both nominal and numeric input attributes. Its representation is rich enough to represent any discrete– value classifier. Decision trees are capable of handling datasets that may have errors, missing values. It is considered to be a nonparametric method as the decision trees have no assumptions about the spatial distribution and the classifier structure [40].

On the other hand, the disadvantage of the decision trees is that since they use the "divide and conquer" method, they tend to perform well if a few highly relevant attributes exist, but less so if many complex interactions are present. One of the reasons for this is that other classifiers can compactly describe a classifier that would be very challenging to represent using a decision tree. Its greedy characteristic makes the decision tree to be over-sensitive to the training set, to irrelevant attributes, and noise.

2.3 Amharic Language Structure

2.3.1 Ethiopic Script

Ethiopia is one of the oldest countries which have their alphabet that has evolved through many forms over centuries. Ethiopia has 83 different languages with up to 200 different dialects spoken [43]. The Ethiopic script is a general term coined for Ethiopian Semitic languages, such as Amharic, Geez, Tigrigna, Guragegna, etc, and the script has been used in the country as a unique writing system since the fifth century. Amharic remained the working language of the federal government of Ethiopia and the language grew as the second most spoken Semitic language in the world next to Arabic [42].

The Amharic writing system is consisting of a standardized set of Ethiopic alphabet called feddel. It is written in tabular form with rows and columns. There are 33 basic rows and 7 basic columns apart from some derived letters. The seven columns represent the vocal sounds of characters in the order of *ä, u, i, a, e, ə,* and o which stands for the first, second, third, fourth, fifth, sixth, and seventh character [42].

Base Sound	Orders						
	1st (ä)	2nd (u)	3rd (i)	4th (a)	5th (e)	6th (ə)	7th (o)
h	ሀ	ሁ	ሂ	ሃ	ሄ	ህ	ሆ
l	ለ	ሉ	ሊ	ላ	ሌ	ል	ሎ
h	ሐ	ሑ	ሒ	ሓ	ሔ	ሕ	ሖ
m	መ	ሙ	ሚ	ማ	ሜ	ም	ሞ
s	ሠ	ሡ	ሢ	ሣ	ሤ	ሥ	ሦ
r	ረ	ሩ	ሪ	ራ	ሬ	ር	ሮ
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
s'	ፀ	ፁ	ፂ	ፃ	ፄ	ፅ	ፆ
p'	ጰ	ጱ	ጲ	ጳ	ጴ	ጵ	ጶ
f	ፈ	ፉ	ፊ	ፋ	ፌ	ፍ	ፎ
p	ፐ	ፑ	ፒ	ፓ	ፔ	ፕ	ፖ
v	ቨ	ቩ	ቪ	ቫ	ቬ	ቭ	ቮ

Figure 2. 4 Some of Ethiopic script

The first column in each row (*ä sound*) represents basic characters and other columns (*u, i, a,e, ə,o sounds*) show modified characters. Thus, the Ethiopic script is a writing system in which vowel sounds (modifiers) are denoted by diacritical marks or other systematic modification of the basic characters.

2.3.2 Amharic Word Class

A word class is a set of words that display the same formal properties, especially their inflections and distribution. The term word class is similar to the part of speech. Sometimes they are also called grammatical category, lexical category, and syntactic category [44]. All words of The Amharic language belong to this word class.

Regarding the number of word class in the Amharic language, two ideas are raised, some researchers categorize Amharic word class into eight and some researchers categorize Amharic word class into 5 word-classes [45, 46, 47, 48]. Since this Master's research doesn't want to go deep into parts of word classes, it uses the second option so that the word classes are Noun, Verb, Adjective, Adverb, and Preposition.

The Amharic Nouns are words used to name or identify a class of things, people, places, ideas, etc. or any subset (member) of these classes. The Amharic noun can be formed in two ways, either Primitive or Derived [45]. If the Amharic noun exists in its original form like (*məsa*, *lunch*), the Amharic nouns are called primitive. On the other hand, if they originated

or derived from other categories specially Nouns, Verbs and they are formed in various ways either by contraction (e.g. lam*ə, a cow from lahəme*) or by change (e.g. k*ədusə, holy from kädäsä*) or augmentation of their letters (e.g. *tänagari, the speaker from tänagärä and lədätə, birth from wälädä*), they are called derived Amharic Nouns.

Considering the termination of Amharic Nouns, they can end in any of the seven sounds except the first [49]. Examples include k*əfu, bad* when they end with second, *gəbi, inside* whey they end with a third sound, *Leba, thief* when they end with the forth sound, *zəmare, song* when they end with a fifth sound, *lamə, cow* why they end with a sixth sound, *Doro, hen* when they end with seventh sound.

The Amharic Verbs are those words that usually come at the end of a complete grammatical declarative sentence and used express the action that the sentence does. In Amharic verb, the third Muscular single of the preterits in the simple forms of the verb (e.g. *käbärä, he buried*) is considered to be the origin of all the other verbal forms. Amharic Verbs can be marked for person, gender, number, case, and tense/aspect.

The Amharic verb in general can be divided into Active and passive verbs. When the verb is active, the subject of the sentence does the verb to the object. Consider the sentence

Käbädä ȧnəbäsawənə gädäläwə, which can be translated as *Käbädä killed the lion* -- (2.1)

The verb *gädäläwə* is an active verb which indicates the subject of the sentence does the action to the object. On the other hand, when using the passive verb, the subject of the sentence is acted upon. Considering the sentence

ȧnəbäsawə tägädälä which can be translated as *the lion was killed* ----------(2.2)

In that sentence, the verb *tägädälä* indicates that an action is acted upon the subject. In the Amharic language usually, the passive verb is formed with Amharic character *tä (ተ).*

The Adjectives are another Amharic Word class that usually come before nouns and serve as modifiers of nouns they precede. In the phrase, *gobäzə temari,* which stands for the clever student, the word *gobäzə/clever* is an adjective word which comes before *tämari/student* and used to provide more explanation about that student.

The propositions are words which don't explain a meaning unless they are attached to other word class. A preposition can be placed around a given word class in different ways. The preposition can appear before a given word to explain a meaning either as a single entity (e.g. *wede betə, to the house*) or attached with a word (e.g. *käbetə, from house*). The preposition can also found after a word as a single entity like w*ä*nəbärə *layə, on the chair* which indicates *layə* as a preposition. The final placement of preposition is before and after a given word as indicated by the word *käwänədəmu garə, with his brother.* In this case, the preposition *kä* appears before the word *wänədəmu* through attachment. The next proposition *garə* appears as a single entity after the word.

The Amharic Adverb words are fewer in number and used to modify the verb used in the sentence [50]. They are mostly located before the verb to explain more about the action that the sentence does. For the statement *tolo mäta,* which can be translated as *he came quickly, tolo* indicates how quickly the person came. So it is one of the Amharic Adverbs.

2.3.3 Amharic Phrases

An Amharic phrase is a group of words that stands together as a single grammatical unit, typically as part of a clause or a sentence. A phrase does not contain a subject and a verb and, consequently, cannot convey a complete thought as a contrast from clause. It is constructed from one or more word categories. Amharic phrases are categorized into noun phrases (NP), verb phrase (VP), Adjectival phrase (AdjP), adverbial phrase (AdvP), and prepositional phrase (PP) [45].

A noun phrase is a phrase that contains head (H) as a noun. An NP can be simple or complex. The simple noun phrase consists of a single noun-like *ȧ*su (he). On the other hand, a complex noun phrase consists of a noun as a head and other words like complements, specifiers, an adverbial and adjectival modifier that modify the head from a different aspect. Consider the phrase *ya yätələnətu tələku yäsarə betə* which can be translated *that yesterday's big thatched house.* In this sentence *Ya(that), yätələnətu(yesterday's), tələku(big), yäsarə(thatched), betə(house)* are the specifier, an adverbial modifier, adjectival modifier, adjectival modifier, and finally a head with noun respectively.

A verb phrase is a phrase that contains head (H) as a verb and other words like specifier, complements, and modifier. Consider the example b*ämäkina wädä betə hedä* which can be translated as *he went to house by car.* In this sentence b*ämäkina(by car), wädä betə (to the house)*are prepositions that modify the verb *hedä (went)* which is the head of the phrase. Thus this sentence is taken as an example of a Verb phrase.

The Adjectival phrase is a phrase that contains head (H) as an adjective and other word like complement, specifier, and modifier. Consider the phrase *yaċi bätamə konəjo* that can be translated as *that very beautiful.* In this sentence *yaċi(that), bätamə(very)* are the specifier, modifier respectively that specifies and modifies the adjective *konəjo(beautiful*) which is the head of the phrase. Thus this sentence is taken as an example of an Adjectival phrase.

The prepositional phrase is a phrase that contains head (H) as a preposition and other words like noun, noun phrase, verb, and verb phrase. Consider the example, *anədə säwə bämədərə which can be translated as a man on the earth.* In this sentence *anədə, bä* are prepositions that are attached to *säwə(man)* and *mədərə(earth)* respectively to form a prepositional phrase.

The Adverbial phrase is a phrase that contains head (H) as an adverb and other words like a verb, verb phrase. Consider the phrase k*əfuġa tägaċu* which can be translated as *crashed severely.* In this sentence k*əfuġa(severely)* is an adverb which considered as the head of the phrase to form an adverbial phrase.

2.3.4 Amharic Sentence Construction

A sentence is an aggregate of words expressing a judgment of the mind. The sentence can be simple or complex. The very common constituent parts of every sentence are the subject, object, and verb [49]. The subject is the principal or the reigning part of every sentence. It also indicates the doer of the action that the sentence performs. In Amharic's sentence, the subject might be expressed explicitly or implicitly. In the sentence, k*äbädä wädä betə hädä, which can be translated as Käbädä goes to the house, Käbädä* is the subject which is explicitly expressed before the object and the verb. In other words, for the sentence, *wädä betə hädä,* the subject is implicitly expressed by the verb *hädä*. In every sentence, the subject precedes the object and the verb. The subject can also be expressed as a compound

subject which consists of several nouns (e.g s*ä*may*ə na mədəre yaləfalu, Heaven and earth shall pass away*), Numerals (e.g. *ė*n*ä*ġ*ə mäto sämanəya säwoċ hedu, those hundred and eighty men have gone*)

The object of a sentence is the person or the thing that receives the action of the verb. It is the who or what that the subject does something to. In Amharic language, it is usually located before the verb. The object in the Amharic sentence is formed by adding the character n*ə(ን)* at the end of the object when the active verb is included in the sentence. In the statement indicated by 2.1, the object is formed by a combination of *ȧnəbäsawə and nə.* When the passive verb is present in the sentence, it is not required to add *nə* when forming an object as it is indicated by sentence 2.2.

The verb of the sentence is the action word in the sentence that indicates what the subject is doing. Along with nouns, verbs are the main part of a sentence or phrase, telling a story about what is taking place. In Amharic's sentence, the verb is usually located at the end of the sentence as indicated by sentence 2.1 and 2.2. In those sentences the word *gädäläwə, tägädälä* are the verbs of the sentence. The Amharic sentence can be classified into three categories simple, compound, and complex [51]. In the simple sentence the items which form the sentence are not qualified by the modifier, but when modifiers such as adjectives are added, the sentence is termed compound. When the modification process reaches the point where a clause is produced, the sentence is said to be complex.

In the simple sentence, the only verbal form is present and the individual items which form the sentence are not burdened with qualifiers. The simple sentence can be constructed using existence assertions like n*ä*w*ə(ነው), honə(ሆነ)*, and predications like *mäta (መጣ), wäsädä(ወሰደ).* The sentence *ṁäṫəhafu ȧdisə näwə* that can be translated as *the book is new*, is an example of a simple Amharic sentence using existence assertion. The sentence tələnətəna ləju mätsəhafunə l*ä*ne s*ä*t*ä* that can be translated as *yesterday, the boy gave the book to me* is an example of a simple sentence using predication [55].

In Compound sentence, each of the items in the simple sentence; subject, object, an indirect object may be expanded by the addition of adjectives or other modifiers. When this expansion process takes place, the sentence is termed a compound sentence. Consider the sentence

ȧnədə ləjə mätsəhafunə wäsädä that can be translated as *one boy take the book* --------(2.3)

As you can see from sentence 2.3, the modifier *ȧnədə/one* modifies the noun boy, thus this can be taken as an example of a compound Amharic sentence.

A complex sentence is one in which the expansion process has developed to the point at which a clause is generated. A clause is a subdivision of a sentence, containing a subject and a predicate. The complex sentence is formed either using a subordinating particle or gerund or infinitive or relative construction [48]. Consider the example

Mätsəhafunə *lämäwəsädə ləju mäta, the boy came to take the book*----------(2.4)

As you can see from the sentence 2.4, two clauses connected by the infinitive *lä(to)* to form a complex Amharic sentence.

2.3.5 Punctuation in Amharic

Punctuation is the use of spacing, conventional signs, and certain typographical devices as aids to the understanding and correct reading of the written text. The Amharic language punctuations are shown in Table 2.2

Table 2. 2 Amharic punctuation mark

No	Symbol	Description
1	።	It is an Ethiopic full stop(*ȧratə nätəbə*) used to indicate the end of the sentence similar to the English full stop(.) mark
2	፡	It is Ethiopic separator (*hulätə nätəbə*) used to separate two words. It is used interchangeably to space similar to the English separator
3	፤	It is an Ethiopic semicolon(dərəbə säräzə) used to join two independent clauses similar to English semicolon (;) mark
4	፣	It is an Ethiopic comma (*nätäla säräzə*) used to separate words and word groups in a simple series of three or more items. It is the same as the English comma(,) mark
5	?	It is an Ethiopic question mark (*təyaqe məlәkətə*) used to indicate question in the sentence.

6	□□□፥	It is an Ethiopic colon used as a mark before giving a list of nouns ideas, phrases. It is the same as English colon(:) mark
7	“”	It is an Ethiopic quotation mark (*təməhərətä ṫəqəsə*) used to indicate that is being reproduced word for word or when someone else’s word is specified in other writing.

2.4 Structured Query Language structure

The name SQL is derived from Structured Query Language. Originally, SQL was called SEQUEL and was designed and implemented at IBM Research as the interface for an experimental relational database system. SQL is now the standard language for commercial relational DBMSs [1]. A joint effort by ANSI and ISO has led to a standard version of SQL (ANSI 1986) called SQL-86 or SQL. Structured Query Language is a query language that is standardized by the American National Standards Institute for most commercial RDBMS. To retrieve or update information users execute 'queries' (SQL Statements) to pull or modify the requested information from the database using criteria that are defined by the user. SQL supports DDL and DML [52]. In this section, basic data manipulation queries i.e. select, update, and delete will be presented.

2.4.1 Data Retrieving Queries

SQL has one basic statement for retrieving information from a database: the SELECT statement. This query is together with other keywords to extract information from the database. The basic form of the SELECT statement, sometimes called a mapping or a select-from-where block, is formed of the three clauses SELECT, FROM, and WHERE and has the following form [1]

SELECT < attribute list >

FROM < table list >

WHERE < condition>

where < attribute list >, < table list >, < condition > indicates a list of attribute names whose values are to be retrieved by a query, list of relation names required to process the query, a

condition (Boolean) expression that identifies the tuples to be retrieved by the query respectively.

In SQL, the basic logical comparison operators for comparing attribute values with one another and with literal constants are =, <=, >, >=, and <>. Consider the following query which retries the birth date and address of an employee named John Smith as an example,

Select BDate, Address from Employee where Fname= 'Jhon' and Lname= 'Smith' -- (2.4)

This query involves only the EMPLOYEE relation listed in the FROM clause. The query selects the EMPLOYEE tuples that satisfy the condition of the WHERE clause, then selects the result on the BDATE and ADDRESS attributes listed in the SELECT clause.

In the select SQL query, the WHERE clause is optional so that it may be missed from the select query. The absence of the WHERE clause indicates as non-conditions are specified on tuple selection; hence, all tuples of the relationships specified in the FROM clause qualify and are selected for the query result. To retrieve the birth date and address of all employees, the previous query 2.4 can be written as query 2.5

Select BDate, Address from Employee ---------- (2.5)

To retrieve all the attribute values of the selected tuples, we do not have to list the attribute names explicitly in SQL; we just specify an asterisk (*), which stands for all the attributes. To retrieve all attributes of the employee relation, the previous query can be written as query 2.6

Select * from Employee where Fname= 'Jhon' and Lname= 'Smith'---------- (2.6)

One feature of SQL allows comparison conditions on only parts of a character string, using the LIKE comparison operator. This can be used for string pattern matching. Partial strings are specified using two reserved characters: % replaces an arbitrary number of zero or more characters, and the underscore (_) replaces a single character. An example is shown in query 2.7 which retrieve all employees whose address is in 'Addis Ababa'

Select Fname, Lname from Employee where Address like '%Addis Ababa %'--------(2.7)

SQL allows the user to order the tuples in the result of a query by the values of one or more attributes, using the ORDER BY clause. The query in 2.8 retrieves the name of the employee in the first name ascending order

Select Fname, Lname from Employee order by Fname ASC ---------- (2.8)

The default order is in ascending order of values. Specifying the keyword DESC displays the result in descending order of values. The keyword ASC can be used to specify ascending order explicitly

Because grouping and aggregation are required in many database applications, SQL has features that incorporate these concepts. Several built-in functions like COUNT, SUM, MAX, MIN, and AVG exist. The COUNT function returns the number of tuples or values as specified in a query. The functions SUM, MAX, MIN, and AVG are applied to a set or multi-set of numeric values and return, respectively, the sum, maximum value, minimum value, and average (mean) of those values. Query 2.9 finds the sum of the salaries of all employees, the maximum salary, the minimum salary, and the average salary of the employees.

Select SUM(salary), MAX(salary), MIN(salary), AVG(salary) from Employee---------(2.9)

In SQL, aggregate functions to subgroups of tuples in a relation, where the subgroups are based on some attribute values can be applied. In these cases, the relation is partitioned into non-overlapping subsets (or groups) of tuples. Each group (partition) will consist of the tuples that have the same value of some attribute(s), called the grouping attribute(s). Then the function is applied to each such group independently. SQL has a GROUP BY clause for this purpose. The GROUP BY clause specifies the grouping attributes, which should also appear in the SELECT clause so that the value resulting from applying each aggregate function to a group of tuples appears along with the value of the grouping attribute(s). Query 2.10 partitions the employee based the department and finds the average salary of the partitioned subgroups.

Select DNO, COUNT(*),AVG(salary) from Employee GROUP BY DNO ----------(2.10)

SQL provides a HAVING clause, which can appear in conjunction with a GROUP BY clause. HAVING provides a condition on the group of tuples associated with each value of the grouping attributes. Only the groups that satisfy the condition are retrieved in the result of the query. Query 2.11 shows only those partitioned departments whose average salary is greater than 10,000

Select DNO,COUNT(*),AVG(salary) from Employee GROUP BY DNO HAVING AVG(salary) > 10,000 ---------- (2.11)

2.4.2 Data Updating Queries

The UPDATE command is used to modify attribute values of one or more selected tuples. A WHERE clause in the UPDATE command selects the tuples to be modified from a single relation. However, updating a primary key value may propagate to the foreign key values of tuples in other relations if such a referential triggered action is specified in the referential integrity constraints of the DDL. An additional SET clause in the UPDATE command specifies the attributes to be modified and their new values. Query 2.12 updates the project name and the department number of project 10.

Update Project set PLocation= ‘Bellaire’ and DNUM=5 WHERE PNumber=10 ----- (2.12)

2.4.3 Data Deleting Queries

The DELETE command removes tuples from a relation. It includes a WHERE clause, similar to that used in an SQL query, to select the tuples to be deleted. Tuples are explicitly deleted from only one table at a time. However, the deletion may propagate to tuples in other relations if referential triggered actions are specified in the referential integrity constraints of the DDL.

Depending on the number of tuples selected by the condition in the WHERE clause, zero, one, or several tuples can be deleted by a single DELETE command. A missing WHERE clause specifies that all tuples in the relation are to be deleted; however, the table remains in the database as an empty table. Query 2.13 deletes the record whose last name is ‘Brown’

Delete from Employee where LName= ‘Brown’ ---------- (2.13)

2.5 Spring Boot

The Spring Framework has been around for over a decade and has found a place as the de facto standard framework for developing Java applications. Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run". Spring Boot makes the life of a developer a lot easier by providing important debug and deployment tools [53].

Spring started as a lightweight alternative to Java Enterprise Edition (JEE, or J2EE as it was known at the time). Rather than develop components as heavyweight Enterprise JavaBeans (EJBs), spring offered a simpler approach to enterprise Java development, utilizing dependency injection and aspect-oriented programming to achieve the capabilities of EJB with plain old Java objects (POJOs) [54].

But while spring was lightweight in terms of component code, it was heavyweight in terms of configuration. Initially, spring was configured with XML (and lots of it). Spring 2.5 introduced annotation-based component-scanning, which eliminated a great deal of explicit XML configuration for an application's components. And Spring 3.0 introduced a Java-based configuration as a type-safe and refactor able option to XML.

Spring Boot performs four core tricks to spring application development. The first trick is its auto-configuration so that the spring boot can automatically provide configuration for application functionality common to many spring applications. The next trick is that it provides for the developer to tell the kind of functionality that developers what to include through starter dependencies. With this dependency, the required libraries will be added to the build. Additionally, the spring boot provides a command-line interface that lets the developer write complete applications with just application code, but no need for a traditional project build. Finally, it provides an insight into what's going on inside of a running Spring Boot application [54].

Spring Boot helps the developer in integrating the application with several frameworks, configuration management, Logging, Transaction Management, Error/Exception handling, Monitoring, and health care, integrating unit testing, and mocking frameworks. It also takes the responsibility of managing versions of dependent libraries.

The reason that the developers want to use spring boot is that it reduces project start-up time. The developers will add frameworks and spring boot automatically configures the framework. It also provides a feature called starter project which helps to quickly add specific features to the application. It makes life easier for the developer by providing debug and deployment tools.

On the other hand, spring is not an application server so that it is impossible to create web applications as self-executable JAR files that can be run at the command line without deploying applications to a conventional Java application server. Spring Boot accomplishes this by embedding a servlet container (Tomcat, Jetty, or Undertow) within the application. Spring Boot doesn't implement any enterprise Java specifications such as JPA or JMS. It does support several enterprise Java specifications, but it does so by automatically configuring beans in spring that support those features. Spring Boot doesn't employ any form of code generation to accomplish its magic. Instead, it leverages conditional configuration features from Spring 4, along with transitive dependency resolution offered by Maven and Gradle, to automatically configure beans in the spring application context [53]

Chapter Three: Related Work

3.1 Introduction

Natural Language Interface to the Database is one of the popular research areas in the world with the advancement of Natural language processing. A natural language interface to a database 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, Hindi, and Amharic). This was introduced because asking questions to databases in natural language is a convenient and easy method of data access, especially for casual users who do not understand complex database query language such as SQL [3]. As compared to other languages, many types of research have been done on English language Interface to the database using various techniques. In other languages like Hindi, Marathi few types of research have been done using rule-based techniques. In this chapter, researches which are related to the Natural language interface to the database will be presented.

3.2 NLIDB for non-Amharic Languages

Garima Singh and Arun Solanki [5] developed an intelligent interface using a semantic matching technique that translates natural language query to SQL using a set of production rules and data dictionary. The research is specifically designed to accept English questions as input. The data dictionary consists of semantics sets for relations and attributes. A series of steps like lower case conversion, tokenization, speech tagging, database element, and SQL element extraction is used to convert Natural Language Query (NLQ) to SQL Query. The transformed query is executed and the results are obtained by the user. The system is developed and implemented in PHP, HTML, CSS, and JavaScript as front-end and the database is implemented in MySQL as the back-end. To test the developed system, a question set consisting of 28 NLQ questions and 50 NLQ questions were fired and an accuracy of 82.1% was achieved.

Prasun Kanti et al [6] used rule-based techniques to convert simple English written questions to questions to SQL in the areas of Natural language Interface to Database. They used tools like Morphological, Lexical, and Syntactic and Semantic analysis to process the English written statements. Then they developed an algorithm that identifies the attributes, the tables

to which the attributes belong, and conditions or values, if any, specified by the user. The research was limited to handle simple SELECT queries.

Prof.Khan Tabrez et al [7] solved the limitation of other research in such a way that it handles both simple and complex Data manipulation queries like nested, join, group, order, queries, etc. The research is specifically designed to accept questions written in the English language. It is based on the rule-based techniques which consist of three modules namely Domain Establishment, Linguistics component, and Data Base components. The Domain Establishment is responsible for the user account and database creation. The second component, the Linguistics component is responsible for translating natural language input to a logical query using morphological, syntactic, and semantic analysis. The Database component is used for SQL generation, execution, and displayed the result back to the user.

Abhilasha Kate, et al [8] added other capabilities to the previous rule-based conversion techniques in such a way that the system handled natural language queries expressed in speech. It is designed to handle English statements as input. Using their system, users can also enter the query using speech. The system will convert speech into the text format. This query will get transformed into the SQL query. The system will execute the query and gives output to the user.

K.Javubar Sathick, et al [9] developed a generic Framework for semantic query conversion in social Web sources. The research is designed for the English Language. Morphological, Lexical, Syntactic, and Semantic analysis and Mapping tables are some of the techniques used by the research. It was implemented using java as the front end and SQL Server as the back end. The authors prepared 3 tables in the SQL Server database for testing the developed system.

Pengcheng Yin et al [10] unlike the previous researches which focus on the translation of English Natural language to structured query language using rule-based techniques, used a Neural Network-based model that learns to understand queries and execute them on a knowledge base table from examples of queries and answers. It finds distributed representations of queries and KB tables, and executes queries through a series of neural network components called “executors”. Executors model query operations and compute

intermediate execution results in the form of table annotations at different levels. NEURAL ENQUIRER can be trained with gradient descent, with which the representations of queries and the KB table are jointly optimized with the query execution logic.

Pooja A.Dhomne et al [11] developed a natural language interface to the database for Hindi and Marathi language. The system uses both semantic and syntactic grammar system architecture. The architecture contains Syntactic Analysis, Token analyzer Spelling checker, Ambiguity reduction. It has two sub-components of NLIDB. The first component, the Linguistic Component is responsible for translating natural language input into a formal query and generating a natural language response based on the results from the database search. The second component, the database component performs traditional Database Management functions.

Mr. Mahesh Chauhan et al [12] developed a Natural language interface to the database using rule-based techniques for the Hindi language. The research explains the advantage of Natural language Interface to Database (NLIDB) in such a way that it handles negation and quantification type of questions and avoids learning artificial languages. Their proposed system involves Lexicon and Tokenizer which retrieve the set of tokens that contain word stem from the word stem and given a token t, retrieve the set of database elements matching t. The lexicon and Tokenizer further involve several steps like Tokenizer, Morph Analyser, Postagger, Chunker, Head Computation so that it divides the tokens according to their syntactic category.

Rashid et al [13] developed an algorithm that efficiently maps a natural language query interface to the database for the Urdu language. The research first tokenizes the statements expressed in the Urdu language, extracts patterns, and converts statements to SQL using attribute value mapping algorithms. They designed a domain-specific dictionary to keep the synonyms of the columns and tables names. They tested their algorithm two query sets with a total of 400 queries that were collected from users of the relevant departments. The first query set was to query the School Management System and the second was to query the Employee Information System. They achieved an average accuracy of 86% and 84% respectively.

Rodolfo et al [14] developed a natural language interface to relational databases for the Spanish language. The architecture of the querying system is consisting of two modules: The query module and Domain Customization module. The Query module is responsible for converting the user input to SQL expressing after it was processed by a lexical parser, syntax checker, and semantics analyzer. The Domain Customization module is used to customize and maintain the data and the dictionaries used by the NLP modules of the system.

3.3 NLIDB for Amharic Language

Anwar Indris [55] constructed a query constructor framework for a web-based search interface to relational databases. It focuses on improving keyword and synonyms based on searching on relational databases using a query constructor framework over a set of databases by constructing a combined database schema. The users are expected to input a keywords and the system returns all information related to that keyword from the combined database schema. It was designed in three-tier architecture and consisted of query text pre-processing, query constructor, SQL query manager, and generating display steps. For its evaluation, five individuals of the mixed level of expertise have been randomly selected from the commercial bank of the Ethiopia service desk department. From many in-house developed database applications of commercial banks of Ethiopia, four of them were selected for the construction of the combined database schema. Its evaluation was expressed in terms of user-level satisfaction and it was about 67.5% that the system responds the relevant result to the provided search keywords.

To the best of the Author's knowledge, there is a single research that has been done for Amharic natural language interface to the database which is directly related to this research. This is a research done by the Tihitina Petros.

Tihitina Petros [15] developed the Amharic Natural language Interface to the Database using rule-based techniques. The research was designed for the bi-lingual (English - Amharic) database since more and more organizations are automating their systems and several databases have bilingual contents. The system allows users to access database contents using Amharic language statements that are not syntax-based. It identified and used the necessary tools and resources such as tokenizer, normalizer, stop word remover, Amharic stemmer,

and bi-lingual (Amharic-English) databases, list of the thesaurus, and list of stop words. The system selects 10 critical tables of Ethio-Telecom's Customer Information Database for experimental testing. The overall accuracy of the system was evaluated to be 76.5%.

3.4 Summary

From the above researches, it is clear to observe that extensive work has been performed in the area of Natural language interface to the database. The old researches used rule-based techniques to convert simple English relevant statements which mainly involve select statements into structured query language. Then the researchers increased the type of the English statement in such a way that the system handles more complex SQL queries. With the advancement of natural language processing, machine learning techniques have been applied to increase the conversion efficiency. By referring to the English language interface to the database, researchers have been developed a Natural language interface to the database for other languages using different techniques. Since Natural language Interface to a database done by the previous researchers on other languages is language-dependent, accessing databases using Amharic natural language is required. So far the natural language interface to the database for the Amharic language is developed only for SELECT queries and it restricts the users only to input pre-described words which are defined by the developer. Thus this research is designed to enable users to access information from the database using free Amharic plain text without restriction by increasing its efficiency. The system is also designed to handle the other data manipulation queries such as update, delete, and some additional select queries which are not covered by [15]. The system can be applied for the databases either its content is written in one language or a mix of two or more languages.

Chapter Four: Proposed Solution

In this chapter, the design and model of the Amharic Database Querying System are explained. It starts with the architecture of the ADQS where each component of the system that makes up ADQS is described one by one. Some of the components of the system have sub-components in which each subcomponent is designed to perform related tasks. Then the description of the data preparation and the machine learning algorithm used for classification comes next. Finally, description and explanation of all algorithms used for the development of this system.

4.1. Overview of ADQS

The name ADQS which stands for Amharic Database Querying System is given by the authors to represent the prototype system that was developed based on this research. It is a web-based application through which the users extract information from the database by describing the questions in the Amharic language. The system is designed in such a way that the users can type the question using Amharic free plain text without restriction. The system then analyzes the Amharic question using the machine learning algorithm and the database information through semantic free grammars and generates SQL queries. This structured query language is executed by query executor and finally, the required information is displayed to the users through a graphical user interface.

4.2 Architecture of ADQS

The ADQS system accepts users' questions which are written in Amharic languages as input and after analysis; it then converts into equivalent SQL and finally returns the extracted information to the user through the user interface. To perform this all processes, the ADQS uses the following modules.

- Web-based Graphical User Interface
- Pre-processor
- Trainer
- Tags Modifier
- Query Element Identifier
- SQL Query Constructor
- SQL Query Executer

Each module is designed to perform specific tasks and produce the output which is used as an input for the next module. The system is divided into modules based on the similarity of a task that they are performing. All modules which are used in this system are indicated by the architecture of ADQS as shown in Figure 4.1

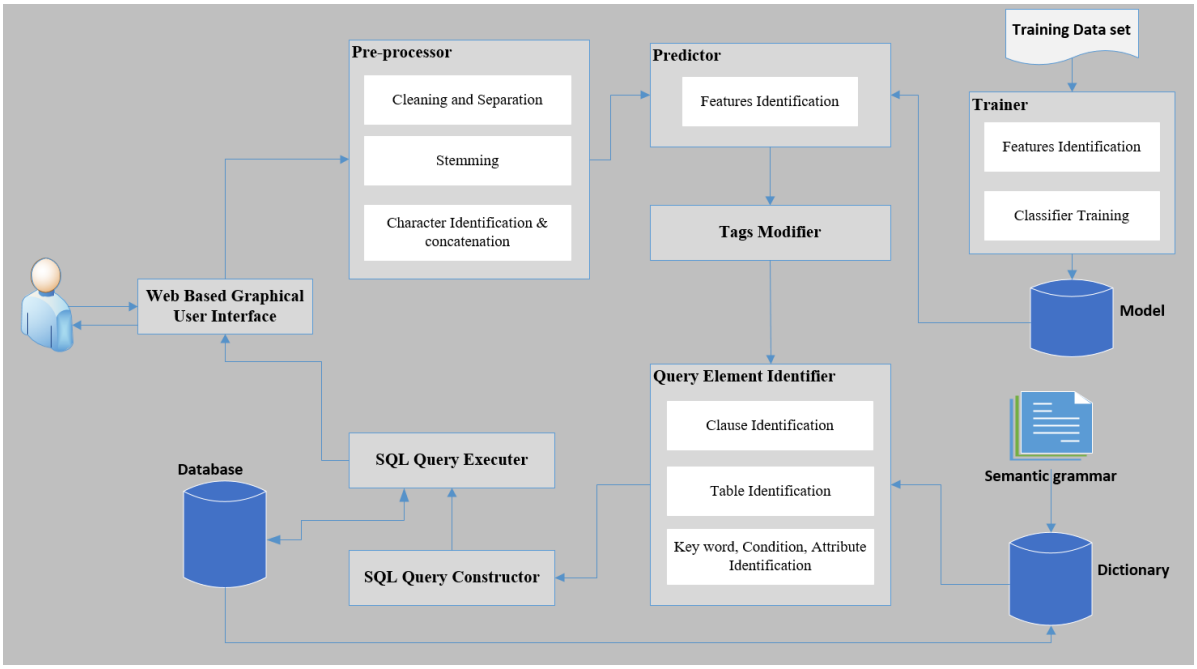


Figure 4.1 Architecture of ADQS

4.3 Web-based Graphical User Interface

This is the first component of the system through which the users must submit the questions to the system. The main function of this module is to accept the users' questions to the system and display back the result to the users.

Based on the languages used for writing questions, the graphical user interface accepts three types of questions. The first types of questions are purely Amharic as the characters are written in the Amharic language. The concepts or meanings are also expressed using Amharic semantics. The second type of question is mixed with English characters. Some of the attributes or table names are commonly known in English languages and they may or may not have a direct translation to the Amharic language. The users may describe these words as it is using characters of the English language. The last type of user's question is

Amharic in syntax and English in semantics. Such types of questions are written purely using Amharic characters but the meanings are expressed in the English language.

In this research, to facilitate the interaction that the users have made with the ADQS system, a graphical user interface has been developed using Java programming language and spring boot framework. The Java programming language is preferred due to its interacting capability with python and familiarity of the authors.

The graphical user interface that has been developed in this research has two major functions in addition to its advantages as compared to the command-line interface. First, it accepts the user's question which is written in Amharic language or a combination of Amharic words with English words. Since the back end is developed with python programming, this user interface must provide the inputs to the back end module in the way that must be understood by the python. The Process Builder option is used to send this Amharic question inputs to the python module. The reason that this option is preferred that the input questions contain characters other than English or a combination of Amharic and English characters.

The second major function of the user interface is to accept the output which is produced by the back end module which is specially made up of python programming. Then it displays the output back to the user in a friendly environment. If the user question is asked using the select clause, the selected result is displayed on the grid view where the user can export the data in excel format. If the user question is asked using the update or the delete clause, a confirmation or error message is displayed to the users. Figure 4.2 shows the process flow of the graphical user interface.

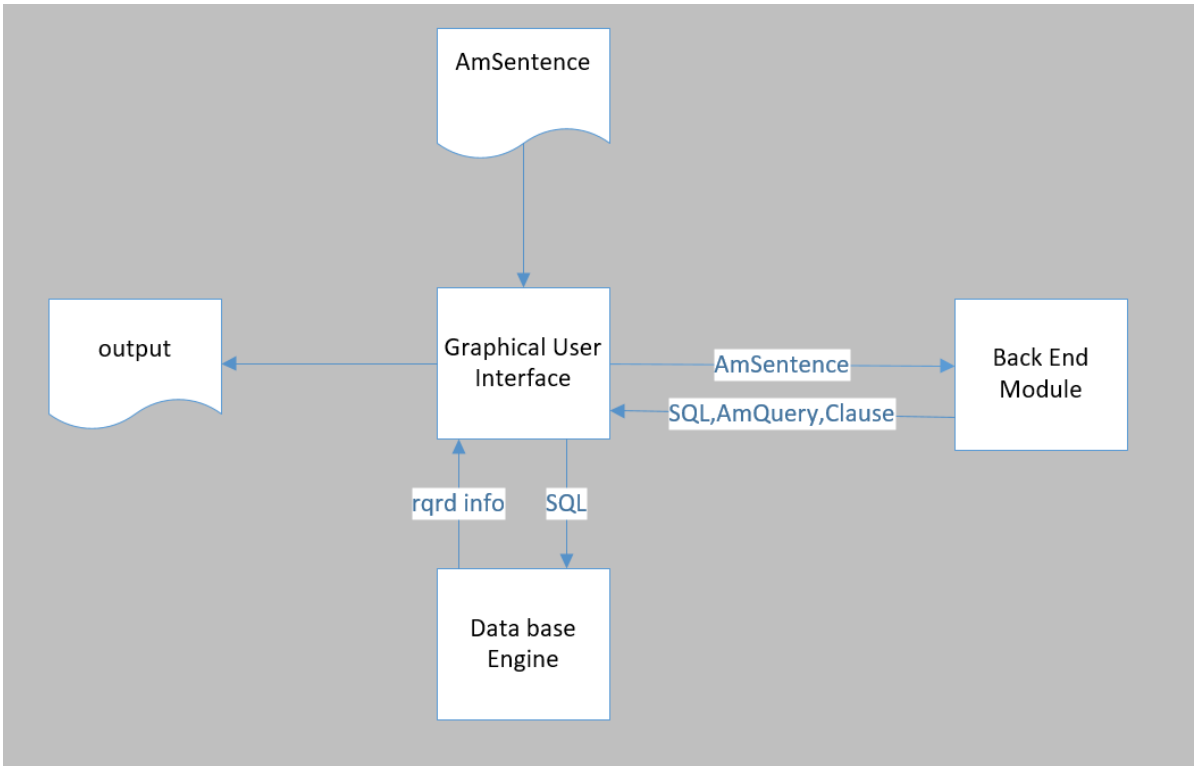


Figure 4. 2 Process of Graphical user Interface

As you can see from Figure 4.2, the graphical user interface accepts Amharic statements which are written in Amharic language or combinations of Amharic and English words. These are given to the Back End module which uses a python programming language to process the Amharic statements. After analysis and generation, the back end module sends back the English SQL query, Amharic query, and clause to the graphical user interface. The graphical user interface sends the English SQL query to the database engine for execution. The AmQuery refers an equivalent SQL which are expressed using Amharic characters. The clause is used to identify the results which are displayed back to the user in the select, update and delete cases. The Database Engine returns the result to the graphical user interface. Finally, based on the clause given by the back end module, the graphical user interface displays the output result or the confirmation message to the user.

The algorithm which is developed for the graphical user interface is shown in Annex A. As you see, there are two main algorithms. One of them is used to find SQL Query with the help of the back end module as indicated by the FindQuery () method and the second is used

to execute the generated SQL with the help of the database engine as indicated by ExecuteQuery ().

The Find Query in Annex C first accepts whatever is given by the back end module and makes some analysis based on the sequence of the sentences returned by the back end module. After that, it will decide whether the returned sentences are a correct SQL query or an error message using the ErrorOrSuccess () method.

On the other hand, the execution query begins execution after a correct SQL query is generated by the back end module. The users may decide whether execution to be continued or not based on their interest. The executed query first makes a connection with the database based on the database name which is selected by the user. Then the clause returned by the back end module, determines whether the required information to be retrieved from the database or a confirmation of update or delete query.

4.4 Pre-processor

Apart from the graphical user interface, the remaining modules are called back end module. These modules are the basic modules which perform the logic behind the ADQS system. They are developed using Python programming language. They accept user's questions from the graphical user interface. They start with the pre-processing module and ends with query generation. The constructed query is also returned to the graphical user interface by the query constructor module.

The first component of the back end module is the pre-processor. The plain Amharic text which comes from the user goes to the pre-processor component for further analysis. This component has three major modules which are described below.

- Cleaning and Separation
- Stemming
- Numbers Identification, Conversion, and Concatenation

4.4.1 Cleaning and Separation

In this module mainly the following four important activities namely cleaning, synonym character conversion, and quotation identification, concatenated word separation and finally short word expansion are performed.

The first task that is performed by the pre-processor module is cleaning. The ADQS allows the user to write the question using Amharic free plain text so that the question may contain many characters that are not necessary for the system. Thus this module is responsible to remove these unwanted characters like stop words, punctuation marks. Additionally, the user may attach the punctuation marks which might be useful or not useful for the system. These unwanted punctuation marks must be unattached and removed from the sentence. Finally, the whole sentence is split into each constituent word based on the type of the separation character inputted by the user.

The second task that is performed by the pre-processor module is to convert synonym characters to some fixed representation. One feature of the Amharic characters which makes different from other language characters is that it has various character representations for a single word. For example, a single word '*ሀ*' can be represented as '*ሐ*', '*ኀ*'. Thus these synonym characters must be converted to a single common word that can be understood by the system.

Some of the where or update value that consists of multiple words might be expressed under quotation as a single word like 'ADAM CONSTRUCTION PLC'. Since in the ADQS system, each separated word is treated as a single word, the system must identify such cases and convert them into a single word as 'ADAM_CONSTRUCTION_PLC'. This is what the quotation identifier component performs during pre-processing.

When concatenated word separation comes, in the user's question some important character or word might be attached to some other word as like ለ07/06/2020. The first letter is a stop word which is unnecessary for the system but the second word is date value which is useful for the system. These must be separated and treated individually as ለ and 07/06/2020. The process flow is indicated in Figure 4.3.

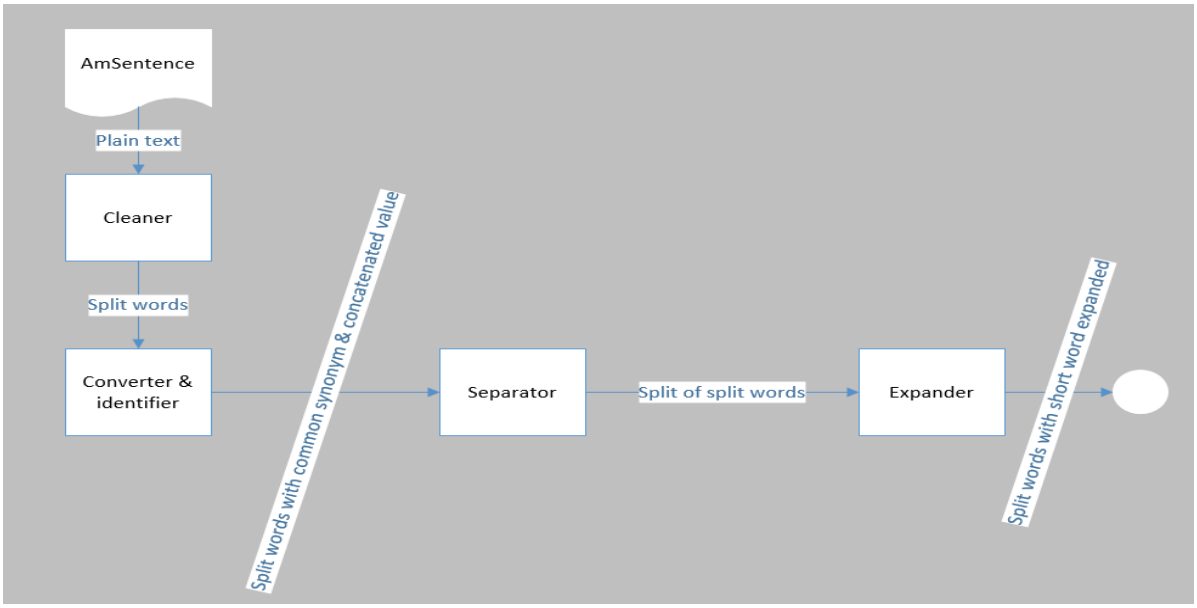


Figure 4. 3 Cleaning and separation

As you can see from Figure 4.3, the cleaner accepts free Amharic text or a combination of Amharic words and English words. The cleaner checks the presence of stop punctuation marks which are defined in the array and removes if they present. On the contrary, if the useful punctuation marks are attached to some other word, the cleaner treats each required word as a separate character and finally split the word based on the white space. The algorithm is shown in Algorithm 4.1.

```
Input: AmSentence
Output: split words
For each character in Stop_punctuations
             If character is found in AmSentence
                          Remove character
             If character is ":"
                          Replace character with empty space
Create and initialize variable index to 0
While index < length of the AmSentence
           If character at index is found in Wanted_punctuations
                  Take characters from 0 up to index and concatenate
                  with empty space, character at
                  Index, Empty space, characters from idx+1 up to
                  length of the AmSentence
                  Increment index by 2
            Else
                  Increment index by 1
Remove the empty space found before and after the AmSentence
Split the AmSentence based on empty space and assign to AmSentence
Return AmSentence
```

Algorithm 4. 1 Cleaner

4.4.2 Stemming

This is the second part of the pre-processing module where the Amharic questions entered by the users are converted into its equivalent stems. This is a very important part of the ADQS system as every word is represented as a single or a few words. In this research only stem of the common words such as words found in the dictionary, aggregate words, conditions words are stored. If the system finds a word which is not found in these stem collection, its stem word is represented as an empty string. The system is designed in this way to save preparation time, resources, and execution time. Secondly, it is also important for the system to decide whether the word is required for query generation. The process done by the stemmer is shown in Figure 4.4.

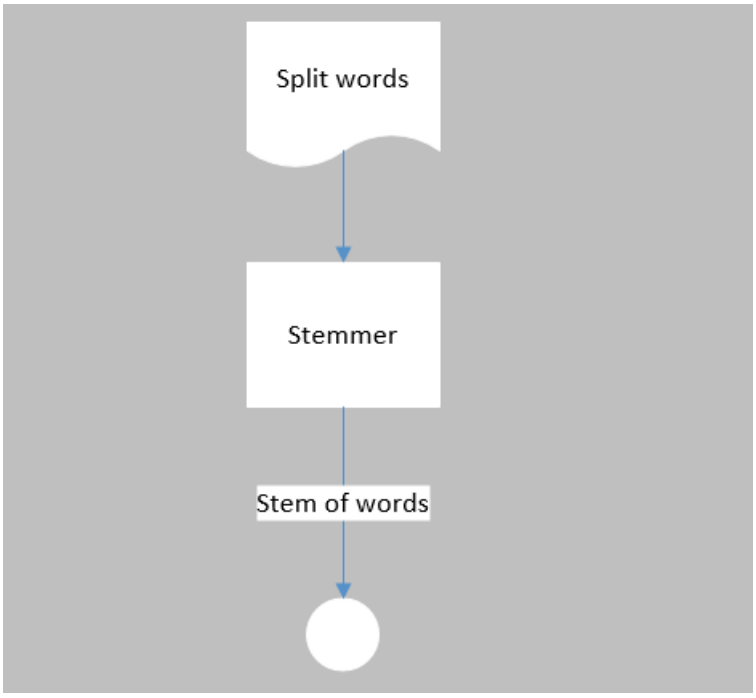


Figure 4. 4 Stemmer

Since the ADQS system is accepting some English words, finding the stem of an Amharic word must be separated from finding the stem of an English Word. The algorithm for finding the stem of the Amharic word is shown in Annex B. Some Amharic Words are exceptional to the Stemmer due to their unique features. These words are taken as an exception and the algorithms are designed to escape these words. The first thing that the Stemmer does is it finds the Prefix and suffix Indexes. Once the prefix and suffix indexes are determined, the stemmer removes these prefixes and suffixes using the algorithm as

shown Annex B. The next task of the stemmer is converting the last character into sixth sound as the stem of the words are stored in such fashion. The character conversion relies on the Unicode representation of the Amharic characters as you can see in the Convert Character Method.

4.4.3 Numbers Identification, Conversion, and Concatenation

In this module, the following three important activities namely identification of numbers written in letters, concatenation of words with the hyphen and finally enclosing values with quotations are performed.

Since in SQL, the numbers are understood as they are written as a digit, there must be a mechanism that converts numbers written in Amharic letters into its correspondence digits. For example, the numbers written in Amharic letters as 'ሁለት መቶ ሀምሳ ዘጠኝ' must be converted into 259.

Some of the attributes and names used in the tables of a given database are described with the combination of two or more words. This is because some common words can be used in many ways to represent different events or different things. For example, the word name can be combined with other words such as Father, Mother, and Wife to identify whether it represents Father's name, Mother's name, or Wife's Name. These words are not quite enough to express a given entity. Similarly, for the single word container, it doesn't say whether it refers to container number or container size, or container type. The same applies to the Amharic language also. The Amharic words 'የአባት ስም', 'ኮንቴይነር ቁጥር', 'ገቢ መሉ ኮንቴይነር' must be concatenated and represented as a single word with the help of hyphen as 'የአባት_ስም', 'ኮንቴይነር_ቁጥር', 'ገቢ_መሉ_ኮንቴይነር'. In this case, both the Amharic questions entered by the users and their corresponding stems are concatenated for further analysis.

To avoid ambiguity of identifying a given word as value or attribute, or to give emphasis to a given word, it has been a good practice to enclose the value under quotations. In this way, the user can easily identify what the statement is talking about. For example, consider the statement; the student named 'Beniyam' has done his research. This statement talks about a single student 'Beniyam' since it was emphasized as indicated by a single quotation. The word student explains more information about the 'Beniyam'. Similarly, it will be a good

practice to enclose the value under quotation so that the ADQS can easily differentiate it from the attributes. Since the user may forget enclosing the value with a quotation, this pre-processing module puts a quotation before and after the end of the word. Thus the Amharic statement ‘በቀን 07/20/2020 ወደብ የደረሱ ሙሉ ኮንቴይነሮችን አምጣልኝ’ must be converted into ‘በቀን ‘07/20/2020’ ወደብ የደረሱ ሙሉ ኮንቴይነሮችን አምጣልኝ’ which can be translated as ‘give me the name of full containers which arrived by the date ‘07/20/2020’. Figure 4.5 shows the process flow of the pre-processor module which is responsible for performing the above-mentioned points.

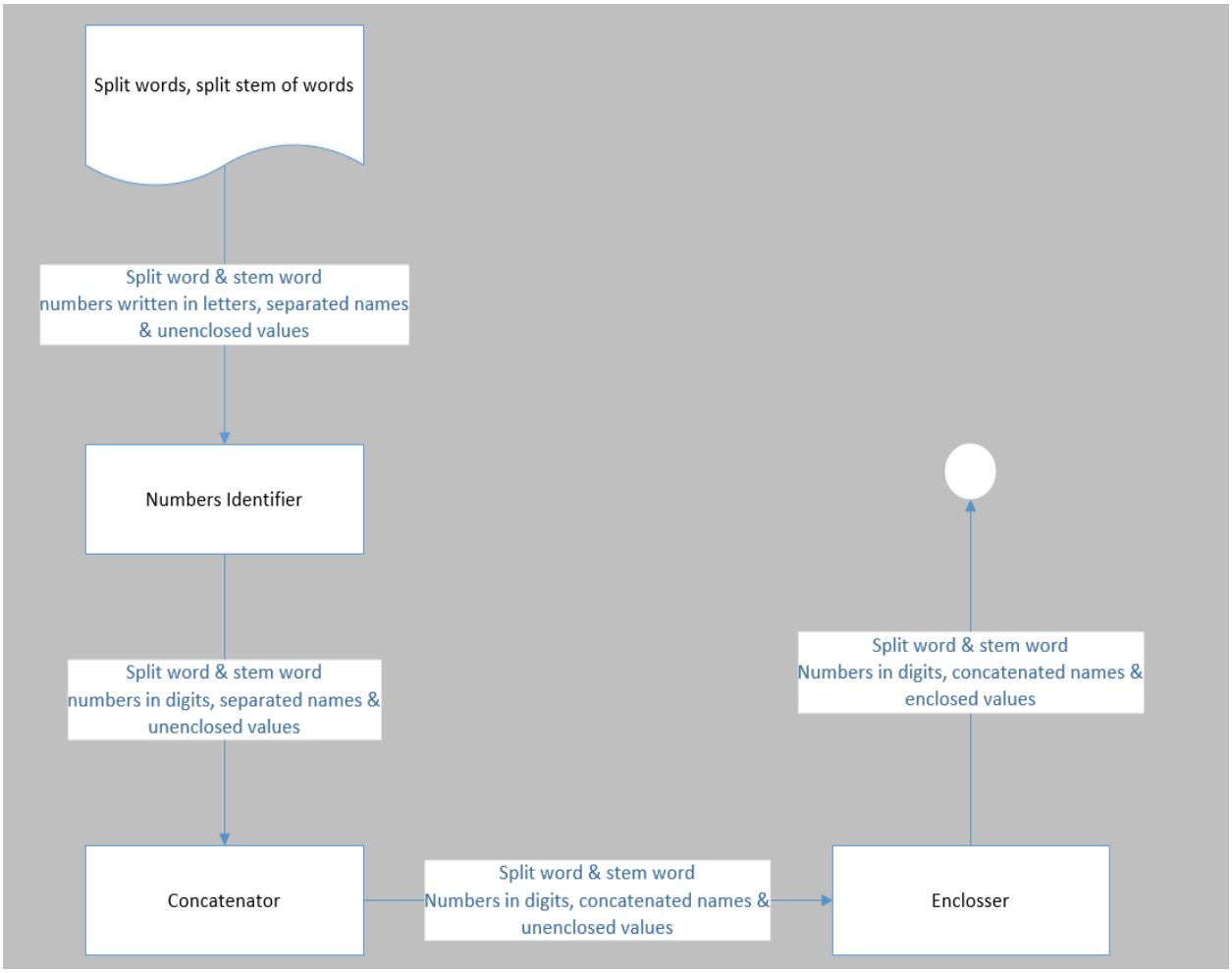


Figure 4. 5 Numbers identification, concatenation, and Enclosing

As you can see from Figure 4.5, the user’s question may contain a number, separated, and unenclosed values. These all inputs are converted into the appropriate format required by the ADQS system. The algorithm for number identification first searches each word in the

numbers dictionary and converts the letters into numbers using an algorithm. The algorithm for the numeric identifier is shown in Annex C. The dictionary that is used for the numbers is also shown in Annex D.

If the users' questions contain two or more names of the attributes, tables, or some common words defined in the ADQS system next to each other, the pre-process module is also responsible for concatenating these words. The system treats these concatenated words as a single word for further analysis. The algorithm developed in the ADQS system for concatenating these words is shown in Annex E.

It would be better for the system, if where and update values are enclosed with a quotation by the user. This is taken as one feature of identifying database elements from stop words. This is also a usual practice in linguistics to emphasize a given word. If the users forget enclosing where and update values, the pre-processor module identifies and makes a quotation based on some general facts. The algorithm used for quotation insertion is given in Algorithm 4.2.

```
Input: sentence array, stem word array, stem order keys array
Output: sentence array, stem word array, stem order keys array
Create and initialize j variable to 0
While j is less than length of sentence array
       If (word at index j is (digit or date) and if it not
       enclosed by quotation) or
       If (word has a single character and if it is not 'h' and
       'ɑ' and 'ɐ' and any of punctuations)  or
       If (word has a single character and if it is one of
       'h','ɑ','ɐ' and if it is proceed by any of
       'h', 'ɑ', 'ɐ') or
       If (word found in Empty Value array and if it proceed or
       succeed by stem not equal to empty)
          Insert quotation at j and j+2 of the sentence array
          Insert quotation at j and j+2 of the stem word array
          Insert quotation at j and j+2 of stem order keys array
          Increment j by 3
       Else

          Increment j by 1

Return sentence array, stem word array, stem order keys array
```

Algorithm 4. 2 Algorithm for quotation insertion

4.5 Trainer

This module deals with the machine learning algorithm that is used in the ADQS system. In this module, data preparation, features identification and classifier training are covered. Since this research uses a machine-learning algorithm, it is required to prepare data sets that are used to train the classifier. For this research, The Decision Tree classifier algorithm uses about 352 Amharic questions that are prepared manually. Each question is made up of a minimum of 4 or a maximum of 20 words. Thus the algorithm trained on average of 3520 words.

There are no publically available Tagged Amharic questions that are used to extract information from the database. Thus the authors prepared the dataset which composed of a word and tag combinations. In this research, a total of 11 tags or output variables are defined as shown in Table 4.1. The Amharic questions are collected by converting English questions that are intended to extract information from the databases. Specifically, English questions found in [1] and [57] are used to prepare the Amharic questions. These questions are prepared in such a way that their main purpose is to extract or update or delete records of the tables which are defined in the ADQS. The system uses two major databases of the Ethiopian Shipping and Logistics Service Enterprise for implementation. Thus these all Amharic questions are prepared to extract or update or delete information that is found in these two databases.

After Amharic question preparation, each word of the sentence was tagged with one of the output variables. Thus a single statement is split into words and each word is tagged with an appropriate tag and stored as a single Amharic question. The system is capable of adding a new Amharic statement into the training dataset by tagging each word with the output variable. Algorithm 4.3 shows the algorithm used to tag each word and serializes it in the training dataset as a pickled element. The system is also capable of removing the wrongly added Amharic question from the dataset.

The training and test dataset are serialized as a pickled element using pickle built-in modules of python. The pickle is used because the machine learning algorithm needs to be saved to make new predictions at a later time without having to rewrite everything again.

Additionally, once the data sets are pickled, it can be saved on disk so that we can move it to some other applications easily.

```
Input: Amharic sentences, Tags, IsforTraining
Output: tagged sentence
Split each Amharic sentences by '|'
Split each Tags by '|'
Open the training dataset
Open the test dataset
For each split single sentence in Amharic sentences
      Split each word by empty space
      Split each tag by empty space
      Create empty sentence_tag array
      For each word in single sentence
           Tag each word with each tag
           Add into sentence_tag array
      If it is for training dataset
          Add sentence_tag to training dataset
      Else
          Add senetence_tag to test dataset
      End if
If it is for training dataset
   Dump training dataset as pickle element
Else
   Dump test dataset as pickle element
End if
```

Algorithm 4. 3 Algorithm for preparing the training and test dataset

Once the users' questions are submitted to the system, the system identifies important features of each word that are used in classification. Some of the important features which are taken into consideration for the classification are the word itself, its previous word, its next word, part of speech tag of the word, part of speech tag of its previous word, part of speech Tag of its next word, the position of the word relative to the verb, position of the word relative to the quotations, position of the word relative to the beginning and the end of the statement, whether the word contains punctuations, whether the word is an object or a subject, whether conjunctions are present between the nouns.

Once the features of each word are identified, these features are given to the system to train the classifier. The system uses the Decision Tree Classifier machine learning algorithm to train and classify the required output. For the training purpose, the authors prepared about 352 Amharic questions each of them can have an average of 10 words. All of these words are tagged with a single letter which can be considered as an output variable. 11 output variables are used in the system as shown in Table 4.1.

Table 4. 1 Output variable

No.	Output Variable	Description
1	T	Indicates the name of the table
2	A	It indicates the filter attribute of the table which is attached with a value
3	R	It indicates the required attribute of the table needed by the user query
4	W	It indicates where value which mostly expressed under quotations
5	V1	It indicates the Select clause
6	V2	It indicates the Update clause
7	V3	It indicates the Delete operation
8	U	It indicates update value which mostly expressed under quotations
9	S	It indicates stop words which are not required for the system
10	C	It indicates conductions
11	K	It indicates a keyword

All words which are used in the training phase are tagged with one of the output variable as described in the table. For example, the Amharic question ‘ሰራተኛ “Biniyam Legesse Tsehay” የሚናገራቸውን ቋንቋዎች እና የትውልድ ቦታውን ከግል ማህደሩ አሳየኝ’ is tagged as

[('ሰራተኛ', 'A'), ('“', 'S'), ('Biniyam_Legesse_Tsehay', 'W'), ('”', 'S'), ('የሚናገራቸውን', 'S'), ('ቋንቋዎች', 'R'), ('እና', 'C'), ('የትውልድ_ቦታውን', 'R'), ('ከግል_ማህደሩ', 'T'), ('አሳየኝ', 'V1')]. This can be

translated as show me the languages that employee ‘Beniyam legesse Tsehay’ is speaking and his birth place from his personal information. The process is indicated in Figure 4.6.

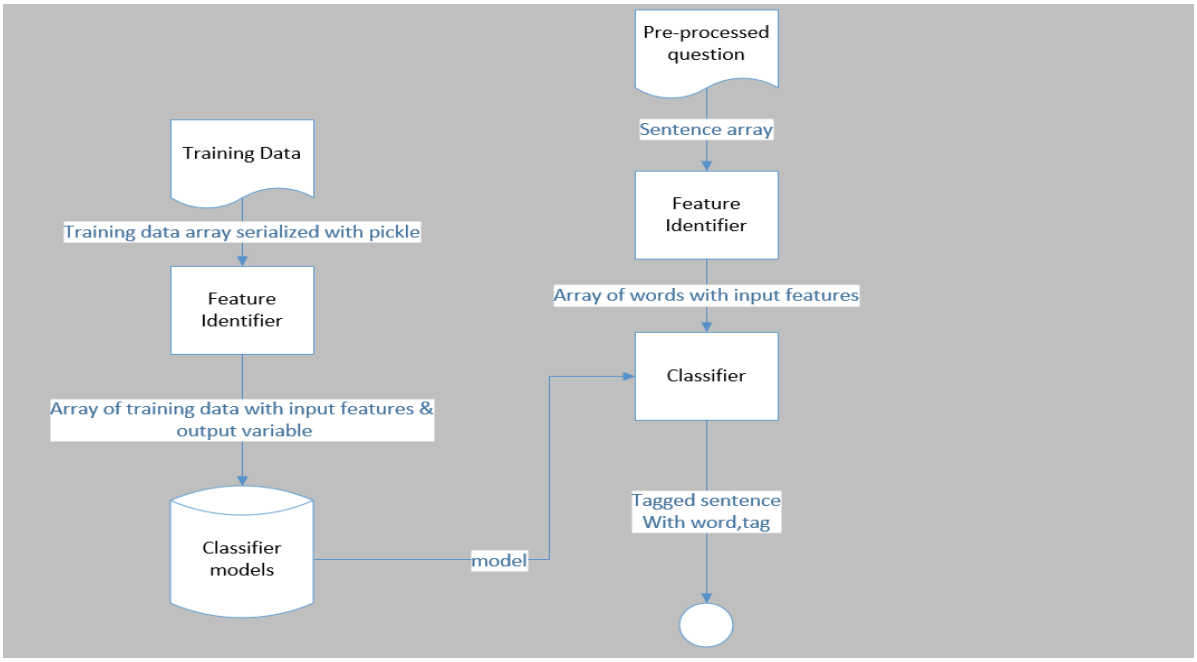


Figure 4. 6 Classifier

The classifier first builds a model that is used for classification using the training data. By finding the features of each word of the training sentence, it builds the classification model. When the pre-processed sentence is given by the pre-processor module, the classifier finds the feature using a feature identifier as shown in Figure 4.6. Finally, it predicts the tag of each word from its feature and classifier model. The output of this classifier is a tagged sentence consists of a word and its correspondence output variable.

The algorithms which are developed for the classifier have two parts. The first focuses on finding features of a given word and the second focuses on training and prediction. The algorithm for feature identification is shown in Algorithm 4.4. For each word of the sentence, about eighteen features are determined by the ADQS system.

```
Input: sentence array, index, sentencePosition, verbIndex
Output: features array
Return {
Set word to sentence array at index
Set prev_word to empty if it is the first word else set to
sentence array at index-1
Set next_word to empty if it is the last word else set to
sentence array at index+1
Set Pos_word to FindPosTag(sentence array, index)
Set Pos_Next_word to empty if it is the last word else set to
FindPosTag(sentence array,index+1)
Set Pos_prev_word to empty if it is the first word else set to
FindPosTag(sentence array,index-1)
Set Position_relative_verb to FindPositionOfWord(word at index,
sentencePosition, verbIndex)
Set IsUnderquatationFlage to DetermineQuoatationFlag(sentence
array, index)
Set Position_relative_quoatation to
DeterminePositionToquatation(sentence array, index)
Set Is_firstFlag to true if it is the first word else set to
false
Set Is_lastFlag to true if it is the last word else set to false
Set Is_Object_Present to true if the last character of the word
at index is 'ʔ' else set to false
Set Is_a_subject to true if the first character of the word at
index is 'ɑ', 'h' or 'ʕ' else set to false
Set has_hypen to true if '-' or '_' present in the word at index
else set to false
Set Is_Numeric to true if the word is a number else set to false
Set Is_Punctuation to true if the word is punctuation else set
to false
Set ConjuctionBnNouns to CheckConjuctionsBetweenNouns (sentence,
index)
Set Is_English_Word to true if the word is English else set to
false
}
```

Algorithm 4. 4 Feature identifier

One of the methods which are described in the feature identifier is the POS tag determiner which is used to find the part of the speech tag of the word. The ADQS system uses the algorithm indicated by Annex F to determine the POS tag of each word. The POS tag Finder algorithm first finds the prefix and suffix indexes and then removes the prefix and suffix

characters based on these indexes. Annex F also shows the removing process and extracting of the root word from the original word.

The main reason that this Master's research uses the indicated algorithm is for the execution time and resource efficiency. Additionally, the system doesn't need all POS tag variables used by other researchers. The most important point here is to identify a word that acts as a verb from other words. This is because most of the time, the select or update or delete clauses are described using this verb. In this research using this POS tag determiner, only verbs are identified and words other than verbs are tagged as nouns. These nouns are specifically tagged as 'K', 'C', and 'S' if they are keywords, conjunctions, and stop words respectively.

Additionally, the verbs are classified further into 'V1', 'V2', 'V3', and 'V' to indicate the select, update, delete, and common clauses respectively. The verbs indicated by 'V' are common words that they can be used in all clauses with the help of some other additional verbs.

4.6 Tags Modifier

Tags in this Master's research are the identifier of each word of the sentence which can be considered as an output variable of the machine learning algorithm. The output tag of the ADQS system is first determined by the classifier as discussed in the previous section. However, when the classifier makes predictions, the output tag may not be accurate. Some of the tags may represent invalid value. This is since some words are used differently in a different sentence. A given word can be used as a required attribute in one sentence and as a filter attribute in another sentence. The remaining features help to decide whether to use this word as a required or filter attribute. However, when the user writes a question, they can freely type as long as it transfers a complete meaning. In such a situation, the required and filter attribute might have almost similar features. Additionally, the features of where value and update value show some similar features.

The Tags modifier module is responsible to amend the tag determined by the classifier based on some general facts. For example, the update values, required attributes must never present in the select, delete clause respectively. Additionally, the information which is

available in the dictionary might force the amendment of the output tags which are determined by the classifier. Some of the general facts which cause a tag modification are listed below.

- ✓ If update value('U') presents in the select clause, it is converted into where value('W') based on the similarity of its features with the where value
- ✓ If Required attribute('R') presents in the delete clause, it is converted into the attribute of a table based on the similarity of its feature with the attribute
- ✓ If Key word('K') presents in any clause as other tags, it is converted into a keyword based on the dictionary of all keywords registered in the system
- ✓ If update('U) or where value('W') in any clause and fails to be validated correctly, it is converted into Stop word based on its corresponding features
- ✓ The table name, required and filter attributes are validated with the help of the database information and converted into stop word if they fail
- ✓ The required and filter attributes are validated with the help of where values specified in the input and converted into the appropriate tag

The algorithm shown in Algorithm 4.5 indicates one of the algorithms that the ADQS system uses to modify the tags.

```
Input: TaggedSentence, InputSentence, requiredTag
Output: TaggedSentence
Unzip the taggedSentence into words and Tags
Convert Tags into list and assign to Tags
If InputSentence is not empty
        For j in length of InputSentence
              If word of InputSentence at j is not empty
                   Set Tags at j to requiredTag
        Convert Tags into tuple and assign to Tags
        Zip words and Tags and assign to TaggedSentence
Else if InputSentence is empty and 'V1', 'U' are present in Tags
         For j in length of Tags
              If Tags at j is 'U' and if j is greater than 0
              and previous word is in Wanted_punctuation
                    Set Tags at j to requiredTag
         Convert Tags into tuple and assign to Tags
         Zip words and Tags and assign to TaggedSentence
Else if InputSentence is empty and 'V3', 'R' are present in Tags
         For j in length of Tags
                    If Tags at j is 'R'
                              Set Tags at j to requiredTag
         Convert Tags into tuple and assign to Tags
         Zip words and Tags and assign to TaggedSentence
Return TaggedSentence
```

Algorithm 4. 5 Tags Modifier example

4.7 Query Element Identifier

This module is responsible to analyze the input sentence and identify the elements which are required for query construction. Query elements are the database elements that are needed for the construction of structured query language. Every SQL queries are constructed with the use of these query elements. These are clauses, tables, keywords, conditions, values, and attributes. To construct SQL queries, the ADQS system must identify at least three of these query elements.

4.7.1 Clause identification

This module is used to analyze and identify the clause from the specified input Amharic sentence. In this research, three clauses, select, update, and delete clause are considered. Thus the main function of this module is to choose the appropriate clause from these clauses.

In this research, the clause is mainly identified by the machine learning algorithm based on the input features given to the classifier. From the Amharic input questions collected in the training phase, it is easily observed that the main words or phrases which indicate whether the clause is select or update or delete, are verb clauses. In other words, if the sentence is analyzed by finding part of the speech tag of each word, there is some hint which helps the identification of the clause. Thus, this hint is taken as one major feature of the Decision Tree classifier to identify the verb. Additionally, since the Amharic verbs are usually positioned at the end of the statement, the position of the word relative to the sentence is also considered as the main feature of the machine learning algorithm.

The algorithm tags these clauses either as ‘V1’ or ‘V2’ or ‘V3’. ‘V1’, ‘V2’, ‘V3’ indicates the select, update, and delete clause respectively as described in Table 4.1. These three clauses almost share the same features as they are expressed similarly in the Amharic questions. However, the verbs are separately used in a given sentence at a time. Words that are used to write select clauses are not used in other clauses. Therefore, in these research words which are used to describe select clauses are grouped and stored separately from the update and delete clause.

There are some common verbs like ‘እጠይቃለው’, translated as ‘I am asking’ which are used in any clause. These words are not enough to describe the mentioned clause since they require the help of other words. This question can be changed into ‘እንድታመጣልኝ እጠይቃለው’ which means ‘I am asking you to give me’. Thus the addition of these verbs will also help the identification of the appropriate clause.

4.7.2 Table Identification

After clause identification, the next thing which must be identified for the query construction is the table element. It is one of the database element which stores specific information about that relation. All clauses fetch the required information from the table element. It is like a container where the required data are stored. They might vary in each query based on the user’s questions

When the users type the Amharic questions, the table element may or may not be expressed explicitly. Thus it is the responsibility of the ADQS system to identify the table element

from the user questions. When the table elements are written explicitly, they are usually written with the help of conjunctions and their part of speech tag is a noun. Sometimes they are written as a description of the doer of the sentence. Most of the time, they are positioned before the verb of the sentence or at the beginning of the sentence. These points are taken into consideration and considered to be the dominant features of the classifier.

Sometimes the table element may not be expressed explicitly in the input Amharic sentence. In such cases, they are expressed in terms of the required or filter attribute or some other stop word. The ADQS stores the most dominant or frequently used words to represent each table of the databases. Thus if the table element is not identified explicitly, the system takes the stem of each word and search for those dominant words of each table.

The ADQS system first accepts the output tags which are identified by the Decision Tree Classifier. The classifier, in turn, takes the features, performs mathematical calculations, and tags each word with the output variable. In a single Amharic statement, there might be more than one word that behaves like a table element. Since this research is limited not to handle joins, only one word or combination of words must be identified as a table element. Thus the ADQS takes those words which are tagged as 'T' from the classifier and perform further analysis.

The correctness of the classifier is checked with the help of the dictionary which stores valid databases, tables, and attributes. The stem of those words which are tagged as 'T' by the classifier is searched from all tables of the database selected by the user. If the match is not found in both explicit and implicit cases, the ADQS system stops execution and informs the user to modify the Amharic questions.

4.7.3 Key word, Condition, Attribute Identification

Once the database and the table elements are identified, the next thing is the identification of keywords, conditions, and attributes. In this research, Keywords are common words that are defined at once in the ADQS system. They are important for query construction as they determine the way that the query is going to be constructed. Keywords are defined query element that helps for the construction of SQL query. They are well known by the database engine so that missing or misspelling of these keywords generates a syntax error. A given

SQL query needs at least one keyword during its construction. For example, when the users want to see the result with some either ascending or descending order, they are expected to use some of the order indication keywords. When they want to see all the required attributes instead of selecting specific attributes, they are expected to use some of all indication keywords. All keywords that are used in the ADQS system are described in the Annex G. These keywords are defined in the system at once and the users are expected to use any one of these words to express their ideas through Amharic sentence. In the ADQS system, they are tagged as 'K' to indicate that they are keywords.

On the other hand, conditions are defined as keywords that are used to filter the records of the table. They are important for query construction as they determine how the records are going to be filtered. Conditions are other query elements which determine the amount of data that is filtered from the table. They determine whether all or specific records of the table are selected or updated or deleted. They are used in the Where clause when the users want to see specific records of the table. For example, the users might want to see the records which are the same as to the value described in the input Amharic statements. They might want to see the records which are between two values described in the input Amharic statements. All conditions that are used in the ADQS system are described in Annex H. These conditions are defined in the system at once and the users are expected to use any one of these words to express their ideas through Amharic sentence. In the ADQS system, they are tagged as 'K' to indicate that they are condition.

Values are the main required information which is stored in a table as a record. These values might be supplied by the user to see or update or delete all records which are related to that specified value in the user's question. The values in the ADQS system are tagged as 'U', 'W' for update and where value respectively.

The next query element, attributes are part of the table elements which are used either infiltration or visualization. Most of the query construction takes one or more of these attributes based on the user requirement. The attributes that are found in the ADQS system, are classified into filtering attributes and required attributes.

The filtering attributes are used to filter the record of the table based on some values that are extracted from the user's questions. They are associated with somewhere values and conditions to filter records of the table. In the ADQS system, these attributes are tagged as 'A' to indicate that they are filtering attributes of the table.

The required attributes are the table's elements that are required to be displayed for the user based on the user's questions. When the users want to see all attributes, they either describe the questions using any of the key_word_all keywords as shown in Annex G or they can explicitly specify the name of the attributes. The ADQS system identifies and takes the appropriate attribute based on the features given to the classifier. In the ADQS system, these attributes are tagged as 'R' to indicate that they are required attributes of the table.

The process flow of these query elements is shown in Figure 4.7. As you can see, the first query element that must be identified is the clause. The output from the clause identifier goes to the table element identifier. The table identifier uses the clause, database name, and dictionary of the database information then identifies the appropriate table name. The dictionary of the database information is prepared using semantic free grammar.

Since the database element names are defined by the developer and the users are not aware of that database element names defined by the developer, there must be a mechanism that fills this gap. In this research, these semantic free grammar rules are used to fetch the root node from users' questions. Using this semantic grammar, non-terminal categories are formed on conceptual rather than syntactic bases [38]. In other words, even though, the users don't know the syntactic structure of the database element, this semantic grammar helps to get the names based on the conceptual representation. This is used to solve the linguistic issues [56]. The reason that this semantic grammar is used in this research is due to a limited domain area of the user question.

In the ADQS system, the semantic free grammar rules or concepts that match the semantic category with the root node to the node in semantic slots are developed. Table 4.2 shows a sample that is used in this system. As you can see, the developers used 'MCRR400' as a table name in the human resource management system which will be discussed in section 6. No one knows this name except the developers. It is described syntactically. The developers

use this name to store personal information. The concept of personal information is known by all users. Thus this concept can be expressed as an employee or as a person or as '*ግለሰብ yägələ mahədärə*' or '*ሰራተኛ/säratäǧa*' as indicated in Table 4.2

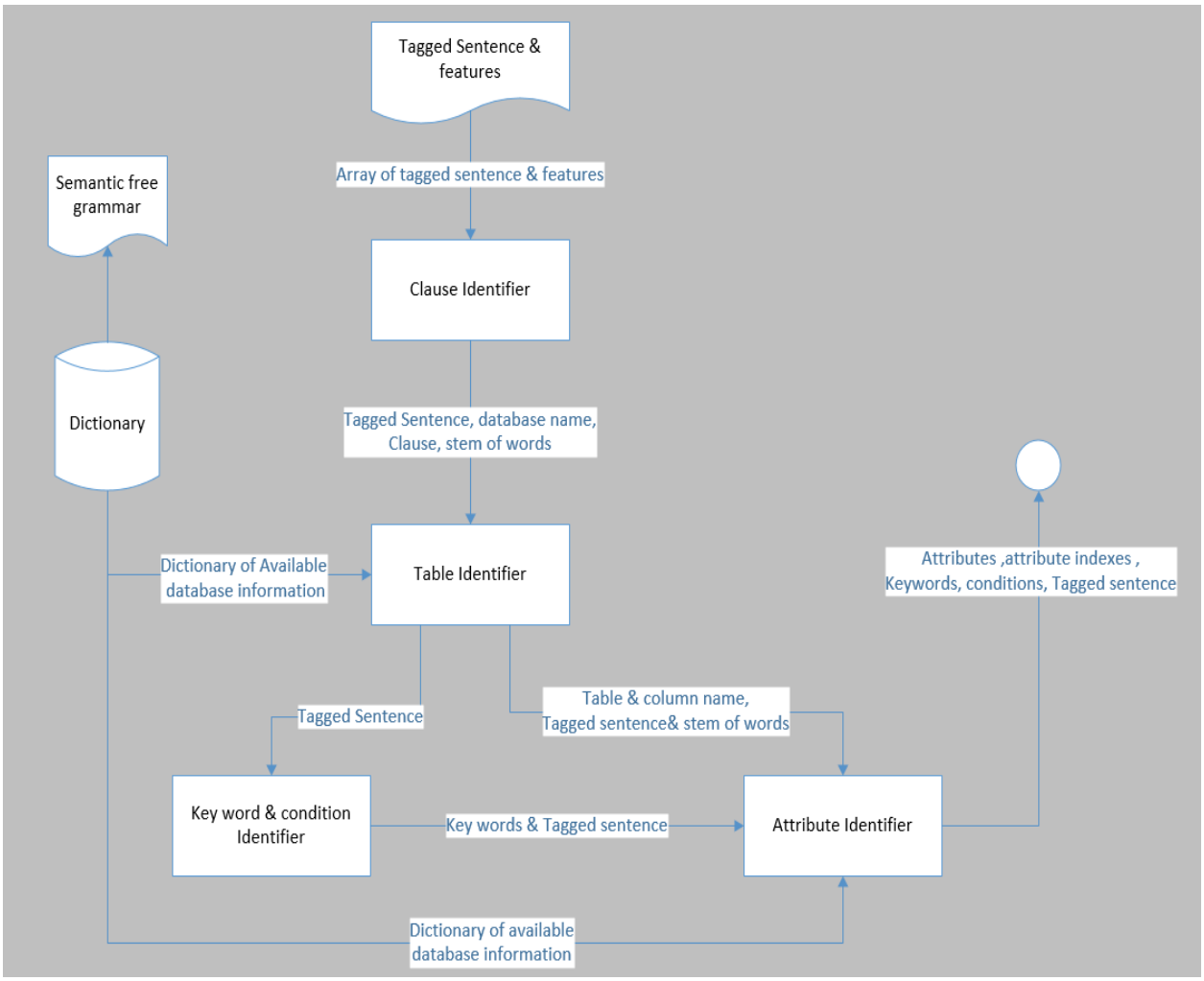


Figure 4. 7 Query Element process flow

Once the table name is identified by the system, the key words, conditions, and attributes are determined using the table name, the stem of the word, the tagged sentence, and the available database information as shown in Figure 4.7. The algorithm for identifying the clause element is shown in Annex I.

Table 4. 2 Semantic root node with semantic slots

No	Semantic category	Semantic slots
1	vwEmployee	employee| personal information | ሰልጣኝ(*sälәṫġə*) | የግል ማህደር(*yägələ mahədärə*) |
2	vwCertificates	certificate |education |education certificate| ሰርተፍኬት(*särətfəketə*) |የትምህርት ማህደር(*yätəməhərətə mahədärə*) |
3	MCRR400	salary |salary earn| የደሞዝ ማህደር(*yädämozə mahədärə*) | የገቢ ደሞዝ ማህደር(*yägäb dämozə mahədärə*) | የወጪ ደሞዝ ማህደር (*yäwäċi dämozə mahədärə*) |
4	MCRW030	experience| የልምድ ማህደር(*yäləmədə mahədär*) |
5	MCRW002	address| አድራሻ (*adəraṡa*)| የአድራሻ ማህደር (*yäadəraṡa mahədär*) |

The clause which is identified by clause determiner is given as an input for the table identifier to identify the appropriate table name. The algorithm for the table identifier is given in Annex J. In the ADQS system, the table name is identified in two different ways. If the table name is expressed explicitly in the user questions, its actual name is determined by the algorithm shown in Annex F. If the table name is not expressed explicitly, the ADQS makes further analysis and determines the appropriate table name using the algorithm described in Annex J. The ADQS system makes further analysis by checking each word of the user's question with the help of the dictionary. Additionally, it uses the main words which describe a given table as a reference and makes an educated guess as indicated.

The algorithm which is used for attribute determination begins analysis from the clause that is identified by clause identifier. The algorithm developed for the select clause is shown in Annex K. It determines the required and filter attributes using the tagged sentence and available database information in the dictionary. Other database elements like keywords and conditions are also determined with the help of keyword and condition identifier. The concepts for identifying the attributes of update and delete clauses are similar to the select clause.

The last Query elements which are used in query construction are values. In this research the values are classified into where and update value depending on the situation they are used in the query. The update value is only found in the update clause whereas where value is found in all clauses. One similarity of these two values is that they are usually enclosed with a quotation mark. The second similarity is that they are records or required information that is

stored in the table. The difference is how they are used on the records of the table. Where value is used together with where clause, attributes, and conditions. Its main function is to filter the records of the table. On the other hand, the update value is used to replace the records which are already stored in the table. The process flow of these database elements is shown in Figure 4.8.

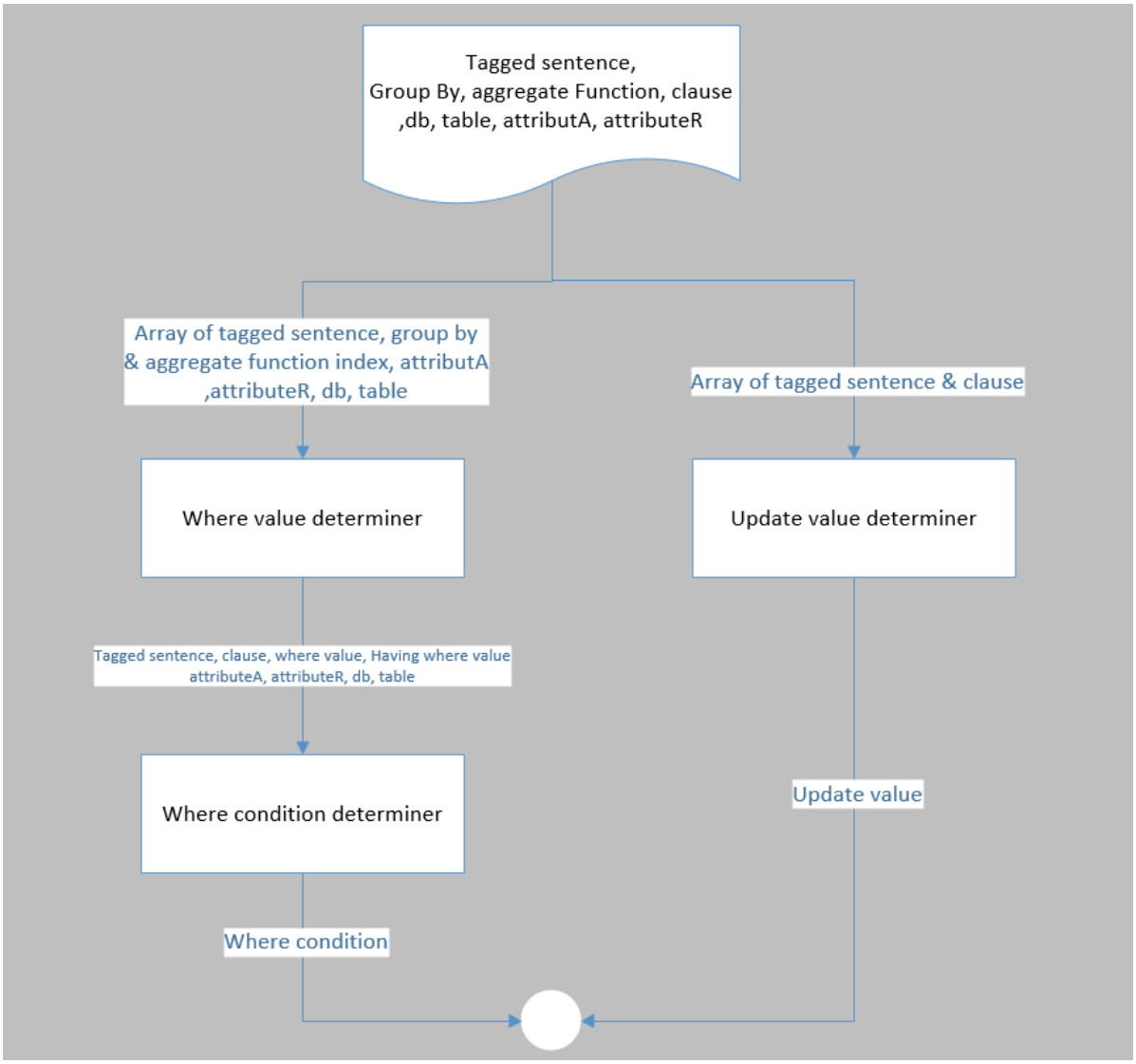


Figure 4. 8 Value determiner

By combining two or more query elements, a new query element might be produced. One of them is where condition. The conditions used in where clause is formed with a combination of filter attributes, conditions, and where values. The number of where values that are

specified in the user's question must be the same as the number of filter attributes. The pseudo-code in Annex L shows the algorithms that find the appropriate where conditions.

4.8 SQL Query Constructor

The next component of the ADQS system is the SQL query constructor which constructs the structured query language based on the query elements that are identified in the previous modules. The query constructor takes the appropriate clause, table, filtering, and required attributes, values, conditions, and keywords which are identified by the previous modules and generates the required SQL query.

The query constructor identifies the type of the SQL queries to be generated i.e. whether it is expected to generate either select query or update query or delete query based on the appropriate clause. The required input needed for the query construction varies based on this clause. For example, when the required query is select, the query constructor takes the table name, filtering, and required attributes, values, conditions, and some other keywords as input for construction. However, when the required query is a delete query, the query constructor takes only table name, filtering attributes, values, and conditions.

Once the construction is done, the query constructor generates the normal SQL queries and an equivalent Amharic based SQL queries. The normal SQL queries are given to the next modules of the ADQS system and the Amharic based SQL queries used for better understanding and future research work.

The query constructor constructs the whole SQL step by step starting from the simple query element. It first starts constructing the order by clause if the user's question contains an order key word indication. The algorithm is shown in Annex M. As you can see, the whole construction is divided based on the user's question whether it contains the where or having value. In the Annex M only construction of query when the user's question contains the where value is described. The concept of query construction is the same for other cases.

The query construction for each case is further classified based on the conditions which are provided in the user's question. If the condition is explicitly expressed in the user's query, the ADQS system identifies and constructs where condition value using this condition. If the conditions are not expressed explicitly, the system takes equality as a default condition.

These in turn are further classified based on the select, update, and delete clauses. The pseudo-code shown in Annex M indicates the query construction algorithm when the user's question contains somewhere value and the conditions are not expressed explicitly.

The query construction in the select clause is further classified based on various conditions. These include whether the users want to see all columns or specific column, whether the user wants to see the result in some order, whether the user wants to see all records or some of the top or last records, whether the user wants to group the result based on some attributes and so on. You can see the detail in the flow chart shown in Figure 4.9. Note that the Figure 4.9 shows only one case of query construction where the user question doesn't contain the where and having value. The same principles are followed for the query construction of other cases.

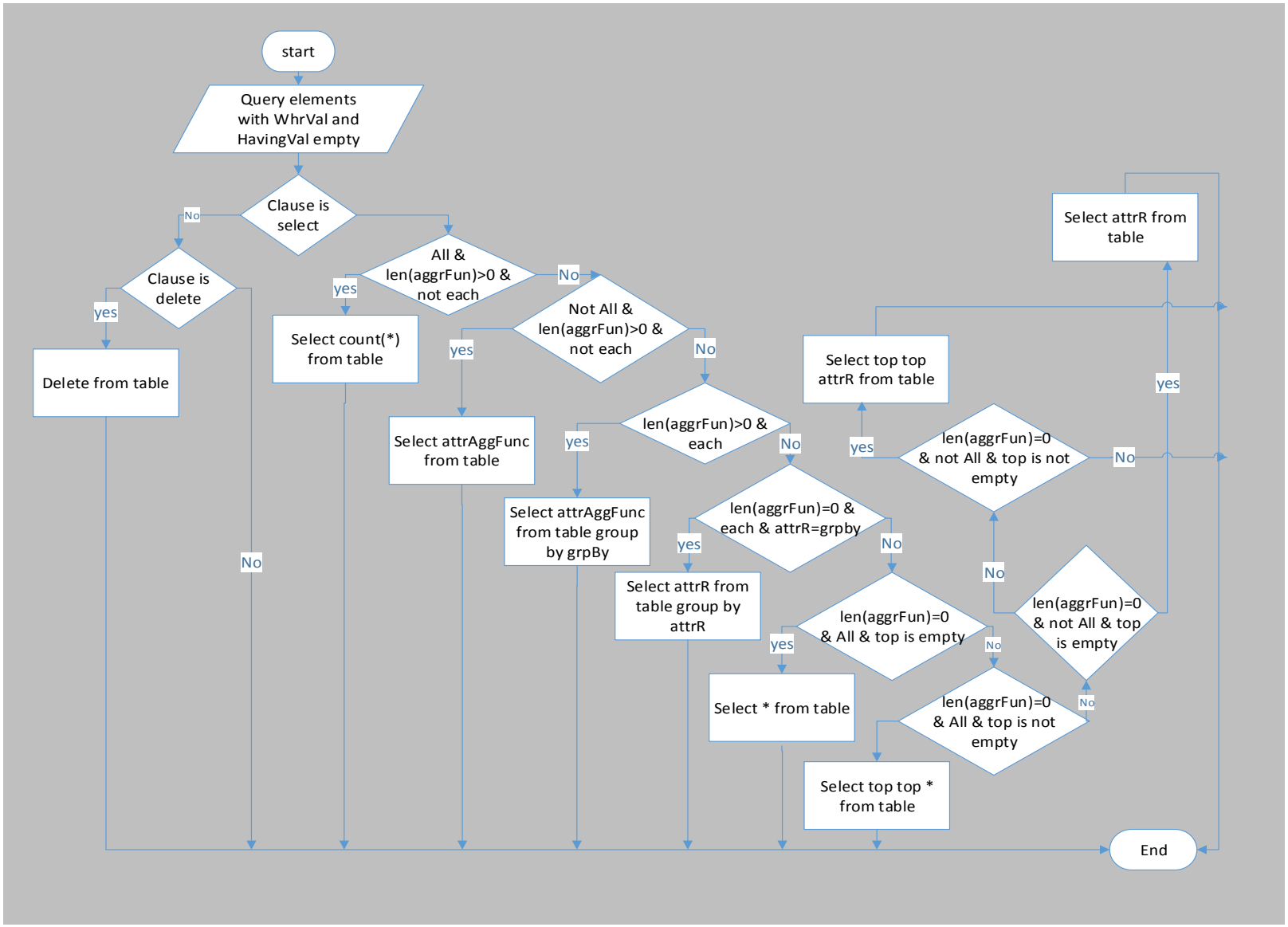


Figure 4. 9 Flow chart for query construction when where and having values are empty

4.9 SQL Query Executer

This is the final module of the ADQS system where the generated SQL queries are executed. The SQL Query Executer takes the normal SQL queries generated by the previous module as an input and executes the query. The SQL Query Executor establishes the connection to the database based on the endpoints specified by the system. The endpoints are in turn is determined by the user through database name selection. The built-in query engine components of the database are responsible for query execution using this connection. The result of the execution is finally displayed back to the user through a web-based graphical user interface.

Chapter Five: Experiment

An Experiment has been conducted to test the effectiveness of the Amharic Database Querying system. For that purpose, the prototype is developed which enables the users to easily access the content of the database using the Amharic language. In this chapter, an explanation about the experimental environment, graphical user interface, database design, types of queries that are handled by the system will be presented first. Then evaluation and discussion will come next.

5.1 Experimental Environment

The prototype is developed by Java and python programming language and spring boot framework. The databases are constructed using Microsoft SQL Server and Oracle. The system is designed to be a web-based application so that the user can easily access it using the web browser. For the testing purpose, the application was deployed on a single laptop computer in the Ethiopian Shipping and Logistics Service Enterprise network using Tomcat. To deploy the ADQS system on the mentioned laptop which is used as a server, apache tomcat with version 9.0.37 is used. User has easily accessed this deployed application as long as they are in the ESLSE's network using web Brower. The system is deployed on the laptop computer with the following specification.

- ✓ Operating system - Windows 10 pro @ 2018 Microsoft Cooperation
- ✓ RAM size - 8GB
- ✓ Hard Disk -500GB
- ✓ System Type – 64-bit Operating System
- ✓ Processor - Intel(R) Core(TM) i7-6600U CPU @ 2.60 GHZ 2.81 GHZ
- ✓ Brand- Dell Latitude E5570

5.2 Graphical User Interface

The Front End of the ADQS system is developed using the Java programming language and spring boot framework. For its development, Eclipse Integrated development environment (IDE), version 2020-03, as a development tool is used. The following dependencies are used in spring boot for the front end development of the ADQS system.

- ✓ spring-boot-starter-security
- ✓ spring-boot-starter-thyme leaf
- ✓ spring-boot-starter-web
- ✓ web jars bootstrap
- ✓ web jars jquery
- ✓ web jars jquery-UI
- ✓ Microsoft SQL Server MySQL JDBC
- ✓ Oracle database jdbc8 version 12.2.0.1
- ✓ Thyme leaf-extras-springSecurity5

In addition to the merits provided by the spring boot framework, custom cascading style sheet (CSS) and Javascript queries (JQuery) are also used. Additionally, some templates taken from bootstrap made [66] also used.

5.3 Back End

The back end modules of the ADQS system which are responsible for handling the logic are developed using the python programming language. For the development, PyCharm, version 2019.2.3 community edition is used as a development tool. Once the program is tested in Pycharm, it is added to Eclipse Project through PyDev which is a python IDE for Eclipse. The following modules are used for the back end development of the ADQS system.

- ✓ Scikit-learn Decision Tree Classifier
- ✓ Scikit-learn DictVectorizer
- ✓ Scikit-learn pipeline
- ✓ String
- ✓ Pickle
- ✓ time

The Back End module, in addition to the training and testing data, it consists of six python file namely Data Collection.py, Pre_Processing.py, ADQS.py, Query.py, QueryConstruction.py, and Dictionary.py. The description is found in Table 5.1

Table 5. 1 Back End Modules

No	Module name	description	Lines of code
1	Data collection	The data collection file is mainly used to train the model, coordinate the other modules	190
2	Pre-processing	The Pre-processing module is used to pre-process the sentence entered by the user.	620
3	ADQS	The ADQS module is used to handle NLP related activities	1600
4	Query	The query module is used to identify the query elements	1800
5	Query Construction	The query construction is used for the construction of the query	1680
6	Dictionary	It is used to provide information for the database element by applying semantic free grammar	360

5.4 Database selection

For the implementation of the ADQS system, two databases of the Ethiopian Shipping and Logistics service enterprise are selected namely DPOIS, HRMS. DPOIS stands for dry port operation and information system and it is an information management system that allows dry port operations to capture data about containers, RORO, and break-bulk. The DPOIS system organizes these data in such a way that it is useful for managerial purposes like planning and controlling dry port operational activities.

The DPOIS system database is constructed using Microsoft SQL server 2012 and it is developed in-house by employees of The Ethiopian shipping and logistics service enterprise. The database consists of around 62 tables, 300 views, 170 stored procedures, and 7 functions. All tables, views procedures, functions, and database names are written in English Character. For the demonstration purpose, 9 main tables and 5 main views with some modifications are prepared. The reason why the DPOIS is selected for the ADQS implementation is that it is developed locally and the Authors are part of the development team. The DPOIS system includes a process to be followed one after the other so that this is used to test the performance of the ADQS system. Additionally, the system is domain-specific that only specific peoples who are familiar with the dry port operation can easily understand the communication. In this way, the possibility of applying this ADQS system for a specific domain can be tested.

The second database which is selected for the implementation of the ADQS system is HRMS which stands for the human resource management system. The Ethiopian Shipping and logistics service enterprise uses a system called SES for managing the shipping business and human resources. The Authors selected the tables and views from SES which are related the human resource management with some modification and called it HRMS. This system is used to manage the human resource of the Ethiopian Shipping and logistics service enterprise. It is constructed using Oracle databases 11g and all nomenclatures are done with English characters. The system is off-the-shelf in the way that the company bought from the software company located outside Ethiopia. The ADQS selected five tables and five main views that are related to the human resource management system. Some modifications to the selected views are done to facilitate communication with the ADQS system. The reason why HRMS is selected for the implementation of the ADQS system is that the system is developed by foreign developers with a language other than Amharic.

5.5 System Usage

The ADQS system is designed to make the interaction with the user friendly so that the user can access the databases easily using the web browser. The System begins with Authentication as only authorized users are allowed to access the system. Figure 5.1 shows the login page.

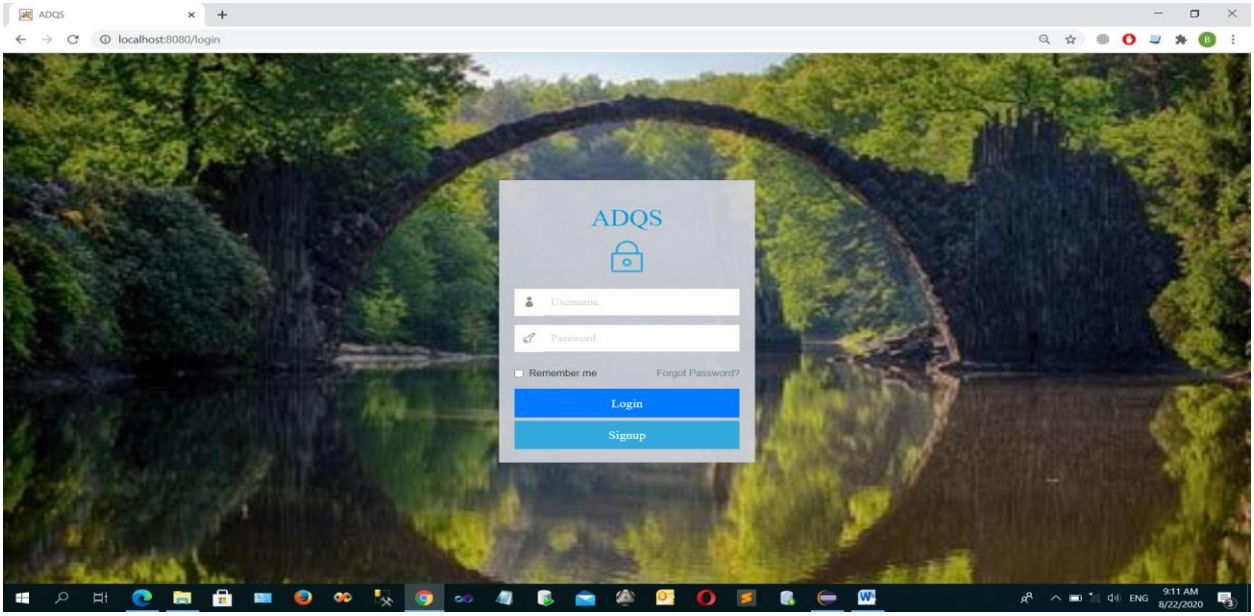


Figure 5. 1 ADQS Login

For the testing purpose, the credentials of some users are defined using the in-memory authentication method of the spring boot security. The users are expected to write the IP address of the server, port number, and the name of the system which is ADQS like http://192.168.75.83:8081/ADQS.

Once the system authorized the user to enter into the system, the index page welcomes the user by providing some introduction about the system. Figure 5.2 shows the index page of the ADQS system. The index page is prepared to give an introduction about the ADQS system by defining some important words and concepts that are used to describe the ADQS system. The system is designed with a mix of English and Amharic language to avoid the language barrier to the system. Annex N shows the Amharic content of the index page of the system. To display the Amharic content on the page, the Eclipse development tool allows changing the preference encoding to UTF-8.

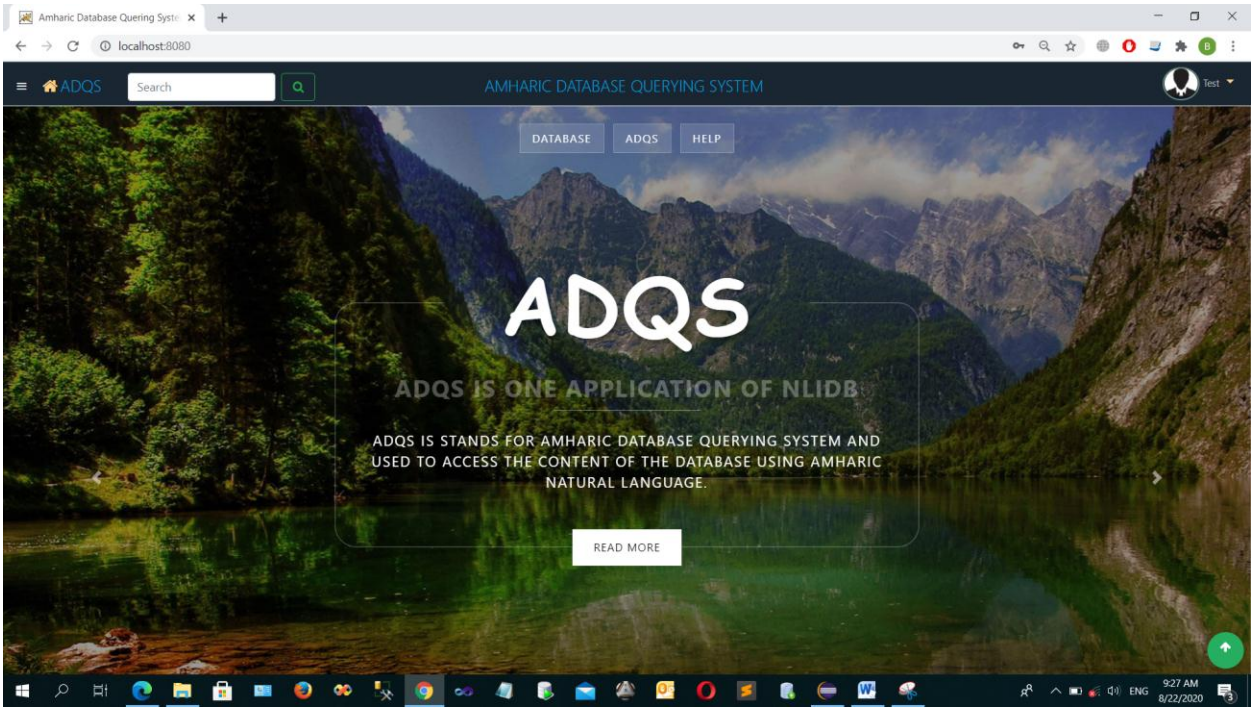


Figure 5. 2 Index page

Next, the users will go to the ADQS page to ask the system using the Amharic natural language. Figure 5.3 shows the ADQS page

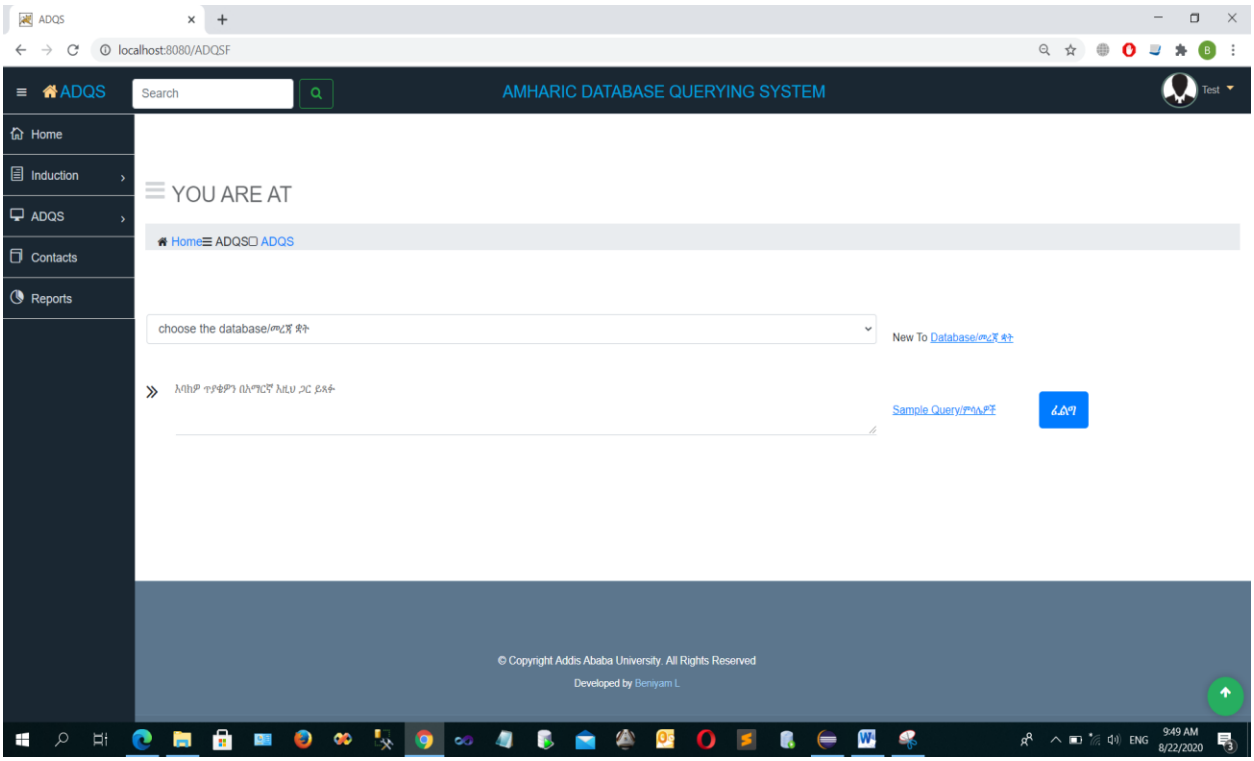


Figure 5. 3 ADQS page

As you can see from Figure 5.3, the users are expected to first choose the database as the ADQS system implemented on two different databases. If they don't have any information about the databases, they can get the general knowledge by clicking the *New to Database* link. Annex N shows the database page. This page displays information about the databases with a mix of English and Amharic language. It first defines the database meaning and shows the two databases that the ADQS is implemented on. It explains what each database does for the user and the available information that the users can get from the database. Annex N also shows how the database information is displayed to the user. Then the users describe the questions in the space provided by the system. If the user confused how to write the question, they can see sample queries by clicking *Sample Query* link. Annex N shows a sample query page. The query page begins by defining the query and its usage. It also gives information for the user that three clauses select, update, delete queries are included in the ADQS system. It then shows examples of how to write Amharic questions with important tips taken from linguistics to remember the user when they write a complete Amharic sentence. Annex N shows a detailed example of Amharic questions with the generated SQL. Finally, the user will click the search button to submit the question to the ADQS system. The system takes this Amharic question and the

selected database as an input and performs an analysis. After analysis or execution, three kinds of responses will be returned to the user as an output. The first response is when the select query is successfully executed. In this case, SQL queries will be returned to the user as shown in Figure 5.4. As you can see from Figure 5.4, a successful select query will show the questions entered by the user, its equivalent Amharic SQL query, and finally the usual English based SQL query.

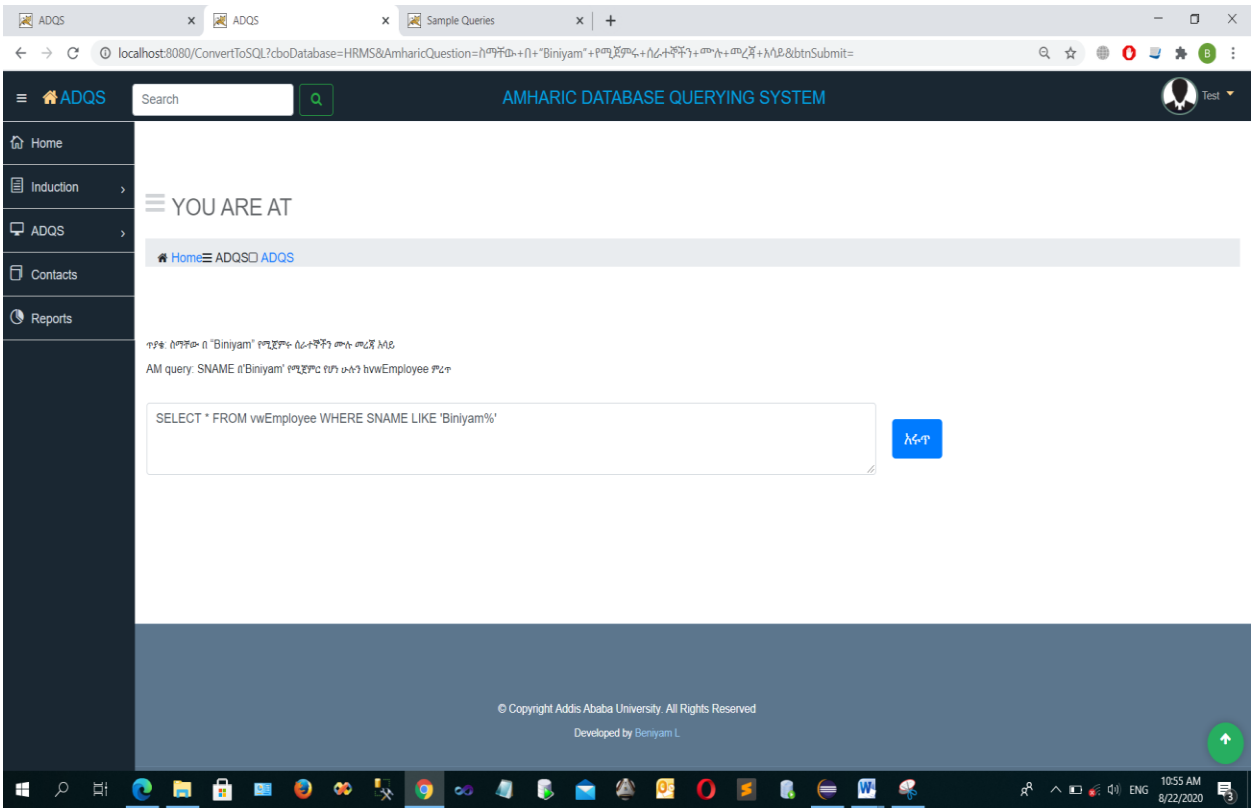


Figure 5. 4 generated SQL query

Then the user will click *the run* button to execute the SQL query. Finally, the ADQS system will display the required information based on the user query in the form of a table as shown in Figure 5.5. As you see in Figure 5.5, the result and the input question are displayed on the same page to provide another option for the user if they want to modify the question. They can rearrange the column of the result which is displayed in the grid view by dragging and dropping the column name in the order that the user wants. They can also select either specific or all rows of the grid view and export the data to an excel sheet for further analysis.

The second response that the system displays for the user are when update or delete queries is successfully updated/deleted as shown in Figure 5.6.

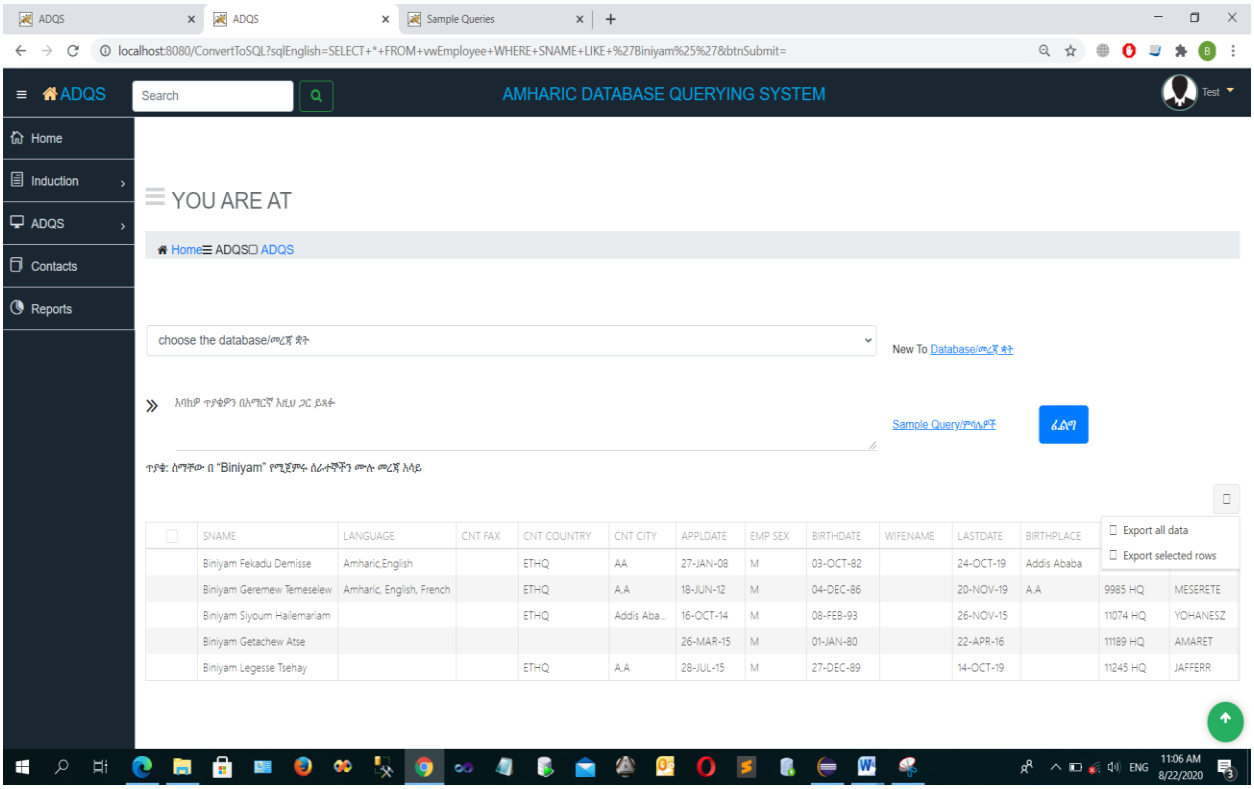


Figure 5. 5 Output

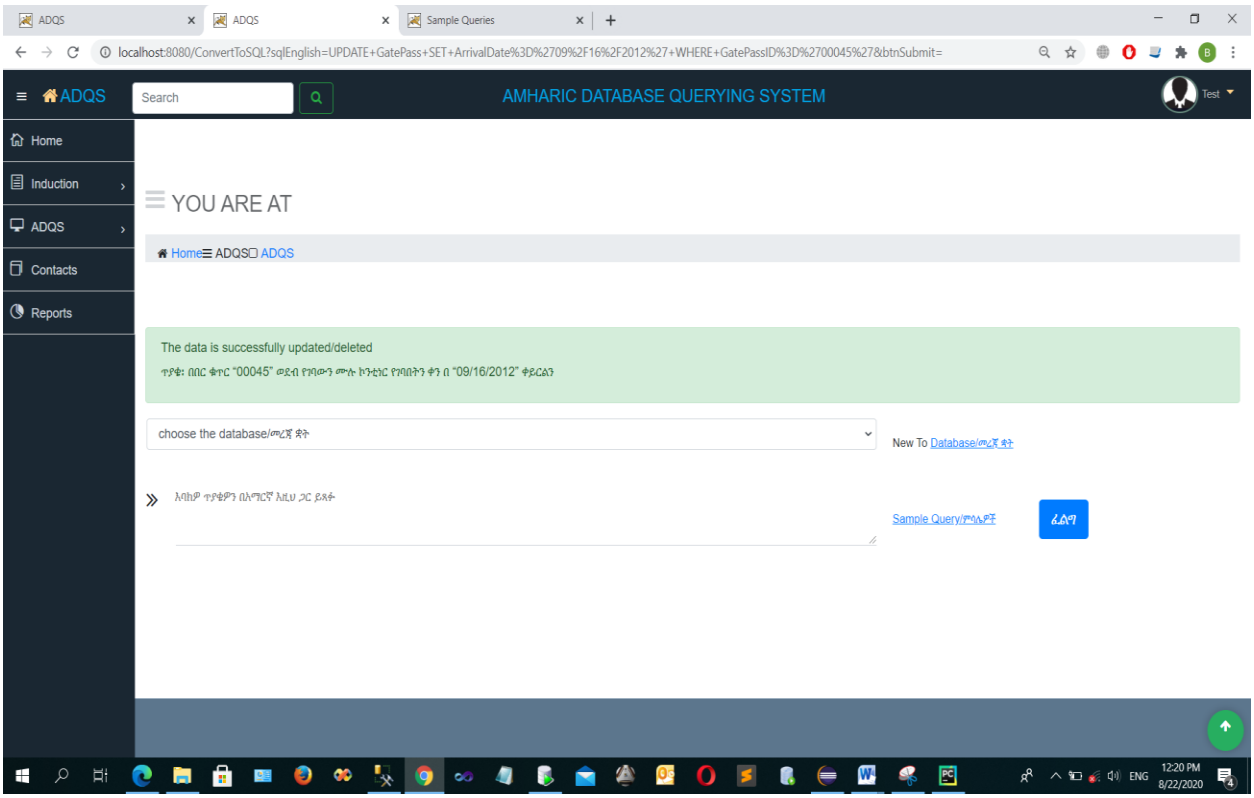


Figure 5. 6 Successful update/delete query response

The last response of the system comes when an error or incomplete sentence to the system is provided. One such case occurs when the user question is not complete so that the ADQS system unable to extract the required information. Figure 5.7 shows the response when such kinds of questions are entered into the system. Another kind of error might occur during query execution by the database engine. In this case, the query is constructed successfully by the system; however, during execution, some exceptions might occur. Figure 5.8 shows the response of the system when an error occurs during query execution

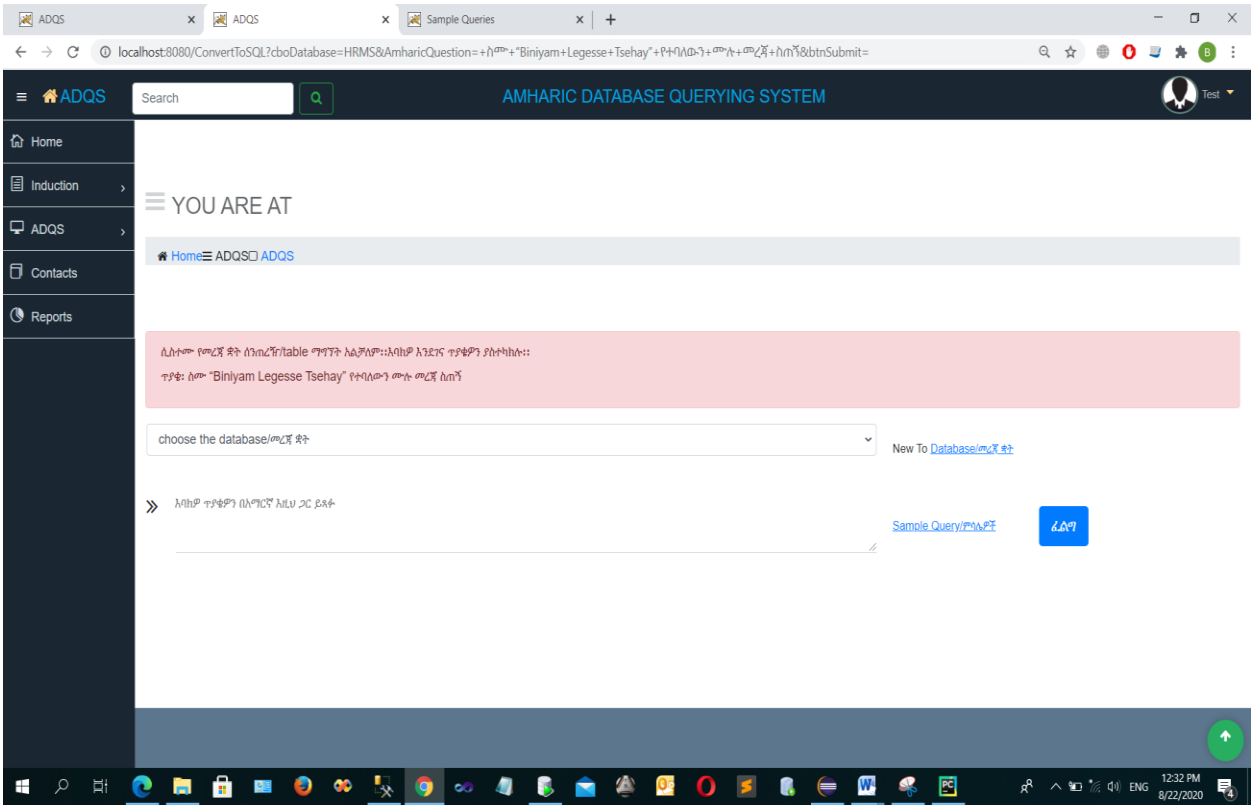


Figure 5. 7 response of the system for incomplete sentence

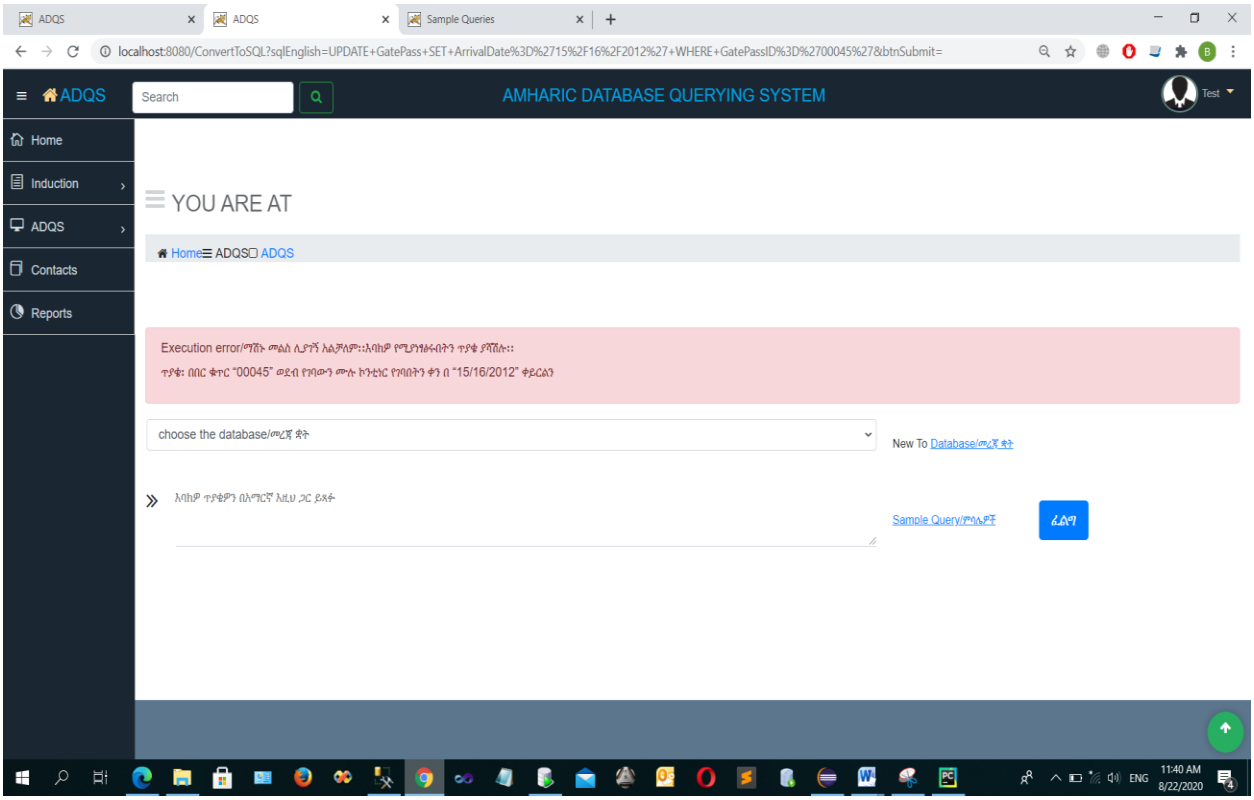


Figure 5. 8 response of the system when an error occurs during execution

5.6 Types of queries

The ADQS system handles three major types of queries namely select, update, and deletes queries based on the clause. Under each type of query, the user might request different information. All kind queries that the ADQS returns to the users are described below.

5.6.1 Select Queries

This kind of query is constructed by the ADQS system when the users request the system to extract information from the database. The users freely express the question without restriction as they communicate with other human beings. The only thing that they expected is to write a complete sentence that has a meaning. The select queries types that are handled by the system are described below in the order of complexity.

Type 1

This type of query is constructed when the user query doesn't contain the where and having the condition. The variant of this type 1 question with an example is described in Table 5.2. In the table Order, Each, Aggregate, Top, all indicates when the keyword indicating the order by, each, aggregate function, top, and all respectively found in the user query

Table 5. 2 Type 1 queries

Type	condition	Sub condition	Example(Amharic)	Example(English character)	SQL query
1.1	Order=false	Each=false, Aggregate=false, Top=false,All=true	ወጪ ባዶ ኮንቴይነር ውስጥ ያሉ መረጃዎችን በሙሉ አምጪ	*Wäċi bado konəteyənärə wəsəṫə yalu märä jawochənə bämulu aməċi*	SELECT * FROM vwSearchSentToDjibouti2
1.2		Each=false, Aggregate=false, Top=false,All=false	**ወደብ ውስጥ የገቡ ሙሉ ኮንቲነሮችን ኮንቴይነር ቁጥራቸውን እንድታመጣ**	*wädäb wəsəṫə yägäbu mulu konətinärochənə konəteyənär kuṫrachäwənə aənədətamäṫa*	SELECT [ContainerNo] FROM vwSearchGatePass2
1.3		Each=false, Aggregate=true, All=true	ገቢ ሙሉ ኮንቴይነር ውስጥ ያሉትን መረጃዎች ብዛት አሳየኝ	*gäbi mulu konəteyənärə wəsəṫə yalutənə märä jawochə bəzatə asayäṅ*	SELECT COUNT(*) as total FROM FullContainerProfile2
1.4		Each=false, Aggregate=true, All=false	ኢባትሎአድ ውስጥ የሚሰሩ ሰራተኞች ብዛት ስንት ነው	*aibatəloaädə wəsəṫə yämisäru säratäṅochə bəzatə sənətə näwə*	SELECT COUNT(SNAME) AS SNAME FROM vwEmployee
1.5		Each=true, Aggregate=true	በእያንዳንዱ የስራ መደብ ውስጥ ያለው የሰራተኞች ብዛት ስንት ነው	*bäaəyanədanədu yäsəra mädäbə wəsəṫə yaläwə yäsäratäṅochə bəzatə sənətə näwə*	SELECT COUNT(SNAME) AS SNAME,RANK_CODE FROM vwEmployee GROUP BY RANK_CODE
1.6		Each=false, Aggregate=false, Top=true,All=true	**የመጀመሪያዎቹን አስራ አምስት ሰራተኞች ሙሉ መረጃ አምጣልን**	*yämäjämäriyawochunə asəra aməsətə säratäṅochə mulu märäja amətalənə*	SELECT * FROM vwEmployee WHERE ROWNUM<=15
1.7		Each=false, Aggregate=false, Top=true,All=false	**የመጀመሪያዎቹን አስራ አምስት ሰራተኞች ስም ዝርዝር አምጣልን**	*yämäjämäriyawochunə asəra aməsətə säratäṅochə səmə zərəzərə amətalənə*	SELECT SNAME FROM vwEmployee WHERE ROWNUM<=15
1.8	Order=true	Each=false, Aggregate=false, Top=false All=true	**ወጪ ባዶ ኮንቴይነር ውስጥ ያሉትን ሁሉንም መረጃዎች በኮንቴይነር ቁጥር ቅደም ተከተል መሰረት አስቀምጥ**	*Wäċi bado konəteyənärə wəsəṫə yalutənə hulunəmə märäjawochə bäkonəteyənärə kutərə kədämə täkätälə mäsärätə*	SELECT * FROM vwSearchSentToDjibouti2 ORDER BY [Container No]

				asəkämәṫə	
1.9		Each=false, Aggregate=false, Top=false,All=false	ወደብ ውስጥ የገቡ ሙሉ ኮንቲነሮችን ኮንቴይነር ቁጥራቸውን በኮንቴይነር ቁጥር ቅደም ተከተል እንድታመጣ	*Wädäbə wəsәṫə yägäbu mulu konәtinärochənə konәteyənär kuṫrachäwənə bäkonәteyənärə kutərə kədämə täkätälə aənədәtamäṫa*	SELECT [ContainerNo] FROM vwSearchGatePass2 ORDER BY [ContainerNo]
1.10		Each=false, Aggregate=false, Top=true All=true	መጀመሪያ ላይ የሚገኙትን ሀያ ሰባት ሰራተኞች ሙሉ መረጃ በቅደም ተከተል ከላይ ወደ ታች አምጡልን	*mäjämäriya layə yämigäṅutənə haya säbatə säratäṅochə mulu märäja bäkədämə täkätälə kälayə wädä tachə aämәtulənə*	SELECT * FROM (SELECT * FROM vwEmployee ORDER BY EMPCODE desc) WHERE ROWNUM<=27
1.11		Each=false, Aggregate=false, Top=true All=false	መጀመሪያ ላይ የሚገኙትን ሀያ ሰባት ሰራተኞች ስም ዝርዝር በስማቸው ቅደም ተከተል መሰረት ከላይ ወደ ታች አምጡልን	*mäjämäriya layə yämigäṅutənə haya säbatə säratäṅochə səmə zərəzərə bäsəmachäwə kədämə täkätälə kälayə wädä tachə aämәtulənə*	SELECT SNAME FROM (SELECT SNAME FROM vwEmployee ORDER BY SNAME desc) WHERE ROWNUM<=27

Type 2

This type of query is constructed When the user query contains having a condition and doesn't contain where condition. The variant of this type 2 question with an example is described in Table 5.3. In the table Order, Each, Aggregate, Top, all indicates when the keyword indicating the order by, each, aggregate function, top, and all respectively found in the user query.

Table 5. 3 Type 2 queries

Type	condition	Sub condition	Example(Amharic)	Example(English Character)	SQL query
2.1	Order=false	Each=true, Aggregate=true	እያንዳንዱ department ካለው ሰራተኞች ውስጥ የሰራተኞቹ ብዛት ከአስር በላይ የሆነውን department አሳይ	*aəyanədanədu department kaläwə säratäñochə wəsəṫə yäsäratäñochu bəzatə käasərə bälayə yähonäwənə department asayə*	SELECT COUNT(SNAME) AS SNAME,RANK_CODE FROM vwEmployee GROUP BY RANK_CODE HAVING COUNT(SNAME)>'10'
2.2	Order=true	Each=true, Aggregate=true,	እያንዳንዱ department ካለው ሰራተኞች ውስጥ የሰራተኞቹ ብዛት ከአስር በላይ የሆነውን department በሰራተኞቹ ብዛት ቅደም ተከተል መሰረት አሳይ	*aəyanədanədu department kaläwə säratäñochə wəsəṫə yäsäratäñochu bəzatə käasərə bälayə yähonäwənə department bäsäratäñochə bəzatə kədämə täkätälə mäsärätə asayə*	SELECT COUNT(SNAME) AS SNAME,RANK_CODE FROM vwEmployee GROUP BY RANK_CODE HAVING COUNT(SNAME)>'10' ORDER BY SNAME

Type 3

This type of query is constructed when the user query contains where condition and doesn't contain having condition. In other words, such type of questions occurs when the user requests to fetch the data from the database which fulfills certain conditions. The variant of this type 3 question with an example is described in Table 5.4. In the table Order, Each, Aggregate, Top, all indicates when the keyword indicating the order by, each, aggregate function, top, and all respectively found in the user query.

Table 5. 4 Type 3 queries

Type	condition	Sub condition	Example(Amharic)	Example(English Character)	SQL query
3.1		Each=false, Aggregate=false, Top=false All=true	ሰራተኛ "Biniyam Legesse Tsehay" በቀን "31-MAY-20 " የሚያገኘውን ደሞዝ ሙሉ መረጃ ስጥ	*säratäñə "Biniyam legesse Tsehay" bäkänə "31-MAY-20" yämiyagäñäwənə dämozə mulu märäja səṫə*	SELECT * FROM vwSalaryEarn WHERE SNAME='Biniyam Legesse Tsehay' AND DATE_TO='31-MAY-20'

3.2	Order=false	Each=false, Aggregate=false, Top=false All=false	የሰራተኛ “Biniyam Legesse Tsehay” ተንቀሳቃሽ እና የቤት ስልክ ቁጥር ከግል ማህደሩ አምጣ	*yäsäratäñə “Biniyam legesse Tsehay” tänəkäsakashə aəna yäbetə səläkə kutərə kegələ mahədäru amət̠a*	SELECT CNT_PHONE1,CNT_PHONE2 FROM vwEmployee WHERE SNAME='Biniyam Legesse Tsehay'
3.3		Each=false, Aggregate=true, All=true	የልደት ቀናቸው በ “01-DEC-1989” እና “27-DEC-1989” መሀከል የሆኑ የኢባትሎአድ ሰራተኞችን ሙሉ መረጃ ብዛት አምጣ	*yäləd̠ätə känachäwə bə “01-DEC-1989”aəna “27-DEC-1989” mähakälə yähonu yäaibatəloaädə säratäñochənə mulu märäja bəzatə amət̠a*	SELECT COUNT(*) as total FROM vwEmployee WHERE BIRTHDATE BETWEEN '01-DEC-1989' AND '27-DEC-1989'
3.4		Each=false, Aggregate=true, All=false	የቅጥር ጊዜያቸው ከ “28-Jul-15” በፊት የሆኑ የኢባትሎአድ ሰራተኞችን ብዛት እንድታመጣልኝ እፈልጋለው	*yäkətərə gizeyachäwə k “28-Jul-15” bäfitə yähonu yäaibatəloaädə säratäñochənə bəzatə anədətamätaləna aəfäləgaläwe*	SELECT COUNT(SNAME) AS SNAME FROM vwEmployee WHERE APPLDATE<'28-Jul-15'
3.5		Each=true, Aggregate=true	ሀገራቸው “ETHQ” የሆኑትን ሰራተኞች በስራ ክፍላቸው አደራጅተህ የሰራተኞቹን ብዛት አምጣላት	*hägärachäwə “ETHQ” yehonutənə säratäñochə bäsəra kəfəlachäwə adärajətähə yäsäratäñochunə bəzatə amət̠alatə*	SELECT COUNT(SNAME) AS SNAME,RANK_CODE FROM vwEmployee WHERE EMP_NATION='ETHQ' GROUP BY RANK_CODE
3.6		Each=false, Aggregate=false, Top=true All=true	በቀን “09/16/2012” ወደብ ውስጥ ከገቡት ሙሉ ኮንቴይነሮች ውስጥ የመጀመሪያዎቹን አስር ኮንቴይነሮች ሙሉ መረጃ አሳየን	bäkänə *“09/16/2012” wädäbə wəsət̠ə kägäbutə mulu konəteyənärə wəsət̠ə yämäjämäriyawochunə asərə konəteyənärochə mulu märäja asaynə*	SELECT TOP 10 * FROM vwSearchGatePass2 WHERE [Arrival Date]='09/16/2012'

3.7		Each=false, Aggregate=false, Top=true All=false	በቀን “09/16/2012” ወደብ ውስጥ ከገቡት ሙሉ ኮንቴይነሮች ውስጥ የመጀመሪያዎቹን አስር ኮንቴይነሮች አሳየን	bäkänə *“09/16/2012” wädäbə wəsəṫə kägäbutə mulu konəteyənärə wəsəṫə yämäjämäriyawochunə asərə konəteyənärochə asaynə*	SELECT TOP 10 [ContainerNo] FROM vwSearchGatePass2 WHERE [Arrival Date] ='09/16/2012'
3.8	Order=true	Each=false, Aggregate=false, Top=false All=true	በ“Jimma University” የተመረቁ የኢባትሎአድ ሰራተኞችን ሙሉ መረጃ በቅደም ተከተል አምጣ	*bä “Jimma University” yätämäräku yäaibatəloaädə säratäṅochənə mulu märäja bäkədämə täkätälə aäməṫa*	SELECT * FROM vwcertificates WHERE ISSUED_AT='Jimma University' ORDER BY EMPCODE
3.9		Each=false, Aggregate=false, Top=false All=false	ሰራተኛ “Biniyam Legesse Tsehay” በቀን “31-MAY-20” የሚገባለትን ደሞዝ በቅደም ተከተል ብታመጣልን ምን ይመስልሀል	*säratäṅa “Biniyam legesse Tsehay” bäkänə “31-MAY-20” yämigäbalätəne dämozə bäkədämə täkätälə bətamätaləne mənə yəmäsələhälə*	SELECT B_AMOUNT FROM vwSalaryEarn WHERE SNAME='Biniyam Legesse Tsehay' AND DATE_TO='31-MAY-20' ORDER BY B_AMOUNT
3.10		Each=false, Aggregate=false, Top=true All=true	ስማቸው በ “B” ከሚጀምሩ ሰራተኞች መሀከል የመጨረሻዎቹን አርባ ሶስት ሰራተኞች ሙሉ መረጃ ስጠኝ	*Səmachäwə bä “B” kmijäməru säratäṅochə mähakälə yämäčäräṡawochunə arəba sositə säratäṅochənə mulu märäja səṫänə*	SELECT * FROM (SELECT * FROM vwEmployee WHERE SNAME LIKE 'B%' ORDER BY EMPCODE DESC) WHERE ROWNUM<=43
3.11		Each=false, Aggregate=false, Top=true All=false	ስማቸው በ “B” ከሚጀምሩ ሰራተኞች መሀከል የመጨረሻዎቹን አርባ ሶስት ሰራተኞች ስም ዝርዝር ስጠኝ	*Səmachäwə bä “B” kmijäməru säratäṅochə mähakälə yämäčäräṡawochunə arəba sositə säratäṅochənə səmə zərəzərə səṫänə*	SELECT SNAME FROM (SELECT SNAME FROM vwEmployee WHERE SNAME LIKE 'B%' ORDER BY EMPCODE DESC) WHERE ROWNUM<=43

Type 4

This type of query is constructed when the user query contains the where and having condition. In other words, such type of questions occurs when the user requests to fetch the data from the database which fulfill certain conditions for a single or group of records. The variant of this type 4 question with an example is described in Table 5.5. In the table Order, Each, Aggregate, Top, all indicates when the keyword indicating the order by, each, aggregate function, top, and all respectively found in the user query.

Table 5. 5 Type 4 queries

Type	condition	Sub condition	Example(Amharic)	Example(English Character)	SQL query
4.1	Order=false	Each=true, Aggregate=true	ሀገራቸው "ETHQ" የሆኑትን ሰራተኞች በየስራ ክፍሉ ከፋፍለህ ከፍተኛ ደሞዛቸው ከ "10000" በላይ የሆኑትን ሰራተኞች ስጠን	*hägärachäwə "ETHQ" yehonutənə säratäñochə bäyäsəra kəfəlu kefafəlähə käfətäna dämozachäwə kä "1000" bälayə yähonutənə säratäñochə səṫnə*	SELECT MAX(B_AMOUNT) AS B_AMOUNT,RANK_CODE FROM vwSalaryEarn WHERE EMP_NATION='ETHQ' GROUP BY RANK_CODE HAVING MAX(B_AMOUNT)>'10000'
4.2	Order=true	Each=true, Aggregate=true,	ሀገራቸው "ETHQ" የሆኑትን ሰራተኞች በየስራ ክፍሉ ከፋፍለህ ከፍተኛ ደሞዛቸው ከ "10000" በላይ የሆነውን በደሞዛቸው ቅደም ተከተል መሰረት አስቀምጥ	*hägärachäwə "ETHQ" yehonutənə säratäñochə bäyäsəra kəfəlu kefafəlähə käfətäna dämozachäwə kä "1000" bälayə yähonwənə bädämozachäwə kədämə täkätälə asəkämətə*	SELECT MAX(B_AMOUNT) AS B_AMOUNT,RANK_CODE FROM vwSalaryEarn WHERE EMP_NATION='ETHQ' GROUP BY RANK_CODE HAVING MAX(B_AMOUNT)>'10000' ORDER BY B_AMOUNT

5.6.2 Update Queries

This kind of query is constructed by the ADQS system when the users want to update specific records that are found in the database. In this query, the users are expected to write a complete statement freely using Amharic plain text. This type of query occurs only when the user question contains a where condition. This there is only one type of update query as shown in Table 5.6.

Table 5. 6 Update query

Type	condition	Example (Amharic)	Example (English character)	SQL query
5.1	Where condition=True	በመለያ ቁጥር “11245” የተጠቀሰውን ሰራተኛ መኖሪያ ቦታ አድራሻ “JiMMA Ethiopia” አድርገው	*Bämäläya kutərə “11245” yätätäkäsäwənə säratäṅa mänoriya bota adəraśa “JiMMA Ethiopia” adərəgäwə*	UPDATE CRW002 SET CNT_CITY='JiMMA Ethiopia' WHERE EMPCODE='11245'

5.6.3 Delete Queries

This query is constructed when the users want to delete a specific or all records from the database. The users in this case are expected to write a question that has a complete meaning. Table 5.7 shows delete query types that are generated by the ADQS system.

Table 5. 7 Delete queries

Type	condition	example	Example (English character)	SQL query
6.1	Where condition =true	ስማቸው በ “Emrias” የሚጀምሩ፤በቀን “03/05/1978” የተወለዱ እና የትውልድ ቦታቸው “A.A”የሆኑ ሰራተኞችን መረጃ አስወግድ	*Səmachäwə bä “Emrias” yämijäməru, bäkänə “03/05/1978” yätäwäladu äna yätəwələdə botachäwə “A.A” yähonu säratäṅochənə märäja asəwägədə*	DELETE FROM CRW010 WHERE SNAME LIKE 'Emrias%' AND BIRTHDATE='03/05/1978' AND BIRTHPLACE='A.A'
6.2	Where condition=false	ወጪ ባዶ ኮንቴይነር ውስጥ ያለውን ሁሉንም መረጃ እንድታጠፋው አዝሀለው	*Wäċi bado konəteyənärə wəsəṫə yalutənə hulunəmə märäja aänədətatäfawə azəhaläwə*	DELETE FROM GalaffeHeader2

5.7 User selection

For testing the Amharic database querying system, 18 different users from a different department with different job positions were selected. The testers are employees of Ethiopian Shipping and Logistics Service Enterprise. The mother tongue languages of the testers also include languages other than Amharic. Most of the testers are not familiar with the databases that were selected for the implementation of the ADQS system. Few of them are familiar with the databases as they are currently working either as an end-user or as a support team. The testers accessed the ADQS system using web browsers that are installed on their personal computer. The endpoints and the credentials are given to the user to test the system. A total of 170 questions were collected from the mentioned users and the result was described in the next section.

5.8 Evaluation

The prototype is evaluated based on the result obtained from the tester based on the above user selection. The confusion matrix is used to describe the performance of the ADQS system. The confusion matrix is a performance table with 4 different combinations of the predicted and actual values. The only thing that the AQDS system requires to generate SQL queries is to have a complete sentence and valid statements. Thus generally the Amharic statements that are gathered from the tester are classified into valid and invalid statements. The valid statements are statements without error that convey a complete meaning. These statements are also valid to the system so that the ADQS system is expected to give an output. On the other hand, invalid statements are statements which are identified based on the scope of the research that is explained in section 1.6. These are statements that have an either syntactic or semantic error. A question whose output generation depends on the analysis of the value stored in the table is also considered as invalid according to the scope. In complete statements are also under this category. Table 5.8 shows the confusion matrix of the testing result.

Table 5. 8 Confusion matrix

N=170	Correctly answered	Incorrectly answered	
Valid questions	TP=137	FN=12	TT=149
Invalid questions	FP=0	TN=21	FT=21
	PT=137	NT=33	

From the above confusion matrix, the measurement shown in Table 5.9 is drawn. Based on the test result, it is indicated that the system accuracy is 92%.

Table 5. 9 Performance measurements

No	Measurement	Value (%)	Description
1	Accuracy (TP+TN)/N	92%	It describes how often the system is correct when valid and invalid questions are given
2	Error (FP+FN)/N	8%	It describes how often the system is wrong when valid and invalid questions are given
3	Precision (TP/PT)	100%	It describes how often are the questions valid when the system answers correctly
4	Recall (TP/TT)	91%	It describes how often the system answers correctly when a valid question is given.

5.9 Discussion

From the above result, it is clear that some factors affect the performance of the system. In the ADQS system, the issues that affect the performance of the system can happen during query generation or query execution.

The test result shows that these errors occurred due to three reasons. One of them might occur due to syntax errors. As it was explained in the scope of the system, spelling checking is out of the scope of the system. As a result, the users might make a mistake while typing the questions. The question might not be a complete sentence so that it can't have a meaning. This is not corrected by the ADQS system. The system may alarm the users if the required information is not provided. The second types of error are semantic errors. In this case, the question doesn't have any syntactic errors, but it doesn't transfer a complete meaning. The users might request a question about a given relation name using attributes of other relations.

The third type of issue that affects the performance of the system is due to system limitations. Even if the user question is correct syntactically or semantically, the system might not respond to the user as expected. From the test result that was observed during the testing phase, this has happened when the classifier tags the required(R) and filter attribute (A). It misplaces the tag of the above attributes. This is because these two attributes share many features in common. Some features distinguish these attributes. Thus if questions are amended to support those unique features, the issue will be solved. The second error that comes from the system limitation is that it responds incorrectly to the questions that are out of the scope of the project. In this case, the

system must alarm the user that this is out of its project. The next error that has occurred during the system test comes due to the description missing. For those where or update values, the users are expected to describe the values explicitly. If that is not satisfied, the system guesses the appropriate attribute based on some general fact. During this time some errors might occur. Again this can be solved by amending the question to include a description of each of the values provided by the user.

For errors that have occurred during query execution, appropriate tables selection plays a greater role. If the table name is not expressed explicitly in the user's question, it is the responsibility of the ADQS system to determine the appropriate relation name. However, some tables might overlap with each other. Even if the SQL query generated by the ADQS system is correct, the required information might not exist in the table that overlaps to other tables. Additionally, during the update and delete query execution, some errors have occurred even after successful SQL query generation. This is because the referenced record may not exist in that table, or it may be referenced in another table so that referential integrity error might occur. This all can be solved at the database designing phase.

The pre-processor module plays a greater role for the accuracy of the ADQS system. In the absence of pre-processor, there is a high probability that important where and update values can be missed. The Tag modifier is also another important module of the ADQS which affects the performance of the ADQS system. Figure 5.9 shows the accuracy of the system with presence and absence of the pre-processor and Tags modifier.

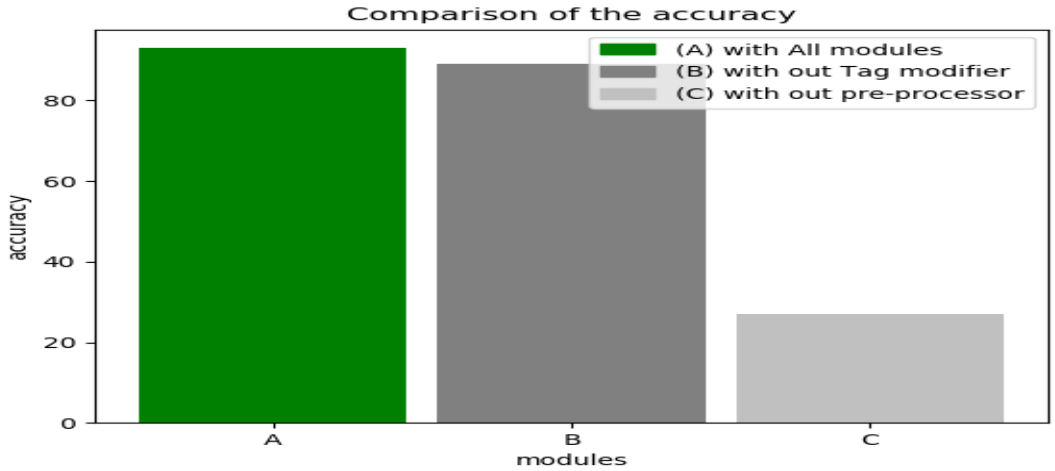


Figure 5. 9 comparison of accuracy of the system with different modules

Chapter Six: Conclusion and Recommendations

In this chapter, the research works that have been done for the development of the ADQS system will be summarized. Additionally, tasks that can be updated or added to this research will be discussed in the Recommendation.

6.1 Conclusion

Nowadays, user activities depend on the information that is distributed in different parts of the world. One of the major sources of information is the database. To access the contents of the database, the users are required to know structured query language (SQL). If users want to access information from the database, they are required to know the English language, structured query language/SQL queries, and they are required to know the physical structure of the database. To minimize these steps, the natural language interface to the database comes into place. This work is proposed as one application of NLIDB to allow users to access the content of the database using Amharic natural language.

Before the design and implementation of the prototype of this master's research, the structure of Amharic language, the natural language interface to the database, and the structure Query language have been studied. Other research works that are related to the Natural language interface to the database using various languages give background knowledge to the techniques and the approaches used in the development of NLIDB. A dataset comprising each word of a given Amharic statement is collected and tagged with an appropriate output variable. From the structure of the Amharic statement, features which are used for the identification of the output variables are identified. The morphological analyzer and stemmers are used to find the root or stem word of the Amharic sentence before the features are given as input to the classifier. The semantic free grammar is used to map the identified tag to the query element. The query constructor is responsible to construct SQL query using the query element which is identified by the query element identifier. Based on the design, a prototype called Amharic database querying system (ADQS) was developed using Java and Python programming language. The ADQS system was deployed in the Ethiopian Shipping and Logistics service enterprise network using the apache tomcat server for testing purposes. Two databases, HRMS and DPOIS are selected to implement the ADQS system. These databases are constructed using Oracle and Microsoft SQL

Server. The users from different departments with different background knowledge are selected to test the system. The experimental result shows 92% of accuracy.

The natural language interface to the database using Amharic language avoids the barrier to the SQL and English language for Amharic language speakers. The implementation of this Amharic database querying system enables the user to access the content of the database using the Amharic language. This makes human's communication with the computer to be more flexible and user friendly.

6.2 Contribution of the Research

Accessing the content of the databases using Amharic language is one application of NLIDB for Amharic speakers. The users directly access the database using the Amharic natural language instead of formulating the formal SQL query. In general, the contribution of this Master's research can be summarized as follows

- The research can serve as a foundation for accessing content of database using Ethiopian language without detail knowing the internal structure of the database
- The research constructs an equivalent SQL query for the user question that are written without any restriction
- The use of the concept of semantic grammars handles linguistic issues
- The pre-processing techniques can be used for other applications

6.3 Recommendations

The ADQS system is developed to enable users to interact with the database using natural languages. This creates an environment where the users communicate with computers similarly as they communicate with other humans. The system includes a way to generate many types of SQL queries based on the user's input. It analyzes the user questions and generates an appropriate SQL query. However, its usability can be extended to analyze the values stored in the database. Additionally, the remaining queries such as sub queries that are not covered in this research can also be added to extend its coverage area.

8. References

[1] Ramez Elmasri, Shamkant B. N avathe “Fundamentals of Database System”, Book, Pearson Education, Inc(2003), fourth edition

[2] Androut sopoulosy, G.D. Ritchiey, P. Thanischz “Natural Language Interfaces to Databases - An Introduction”, Journal of Natural Language Engineering, Department of Artificial Intelligence and Computer science, University of Edinburgh, Scotland, U.K.1994

[3] Sujatha, Viswanadha, and Humera, “A Survey of Natural Language Interface to Database Management System”, International Journal of Science and Advanced Technology (ISSN 2221-8386), Volume 2 No 6 June 2012

[4] World Population Review retrieved from “http://worldpopulationreview.com ” United State, 2019, last accessed on October 24, 2019

[5] Garima S., Arun S “An Algorithm to Transform Natural Language into SQL Queries for Relational Database”, Thesis, Department of Computer Science and Engineering, International Academy of Ecology and Environmental Sciences, Gautama Buddha University, Greater Noida, India,2016

[6] Prasun Kanti Ghosh, Saparja Dey, Subhabrata Sengupta, “Automatic SQL Query Formation from Natural Language Query”, International Journal of Computer Applications (0975 – 8887), International Conference on Microelectronics, Circuits, and Systems (MICRO-2014)

[7] Prof.Khan Tabrez, Shaikh Shagufta, Shaikh Sharmeen, Momin Ummiya, Shaikh Rameeza, “NLP TO Create SQL Query”, International Journal for Scientific Research & Development, Vol. 3, Issue 09, 2015

[8] Abhilasha K., Satish K., Aishwarya B., Mrunal J. “Conversion of Natural Language Query to SQL Query”, Proceedings of the 2nd International Conference on Electronics, Communication, and Aerospace Technology (ICEC 2018), IEEE conference Record no.42487, 2018

[9] K.Javubar Sathick, Dr. A.Jaya, Mohamed Nayeemudeen “Generic Framework for Semantic Query Conversion in Social Web sources”, Department of Computer Applications, B.S.Abdur Rahman University, 2014

[10] Pengcheng Yin, Zhengdong Lu, Hang Li, Ben Kao “Neural Enquirer: Learning to Query Tables in Natural Language”, Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, New York, 2016

[11] Pooja A.Dhomne, Sheetal R.Gajbhiye, Tejaswini S.Warambhe, Vaishali B.Bhagat "Accessing Database Using NLP", International Journal of Research in Engineering and Technology, 2013

[12] Mr. Mahesh Chauhan, Ms. Nikita Bhati "A Noval Hindi Language Interface for Databases", International Journal of Computer Science and Mobile Computing (2014)

[13] Rashid Ahmad, Mohammad Abid Khan, and Rahman Ali, "Efficient Transformation of a Natural Language Query to SQL for Urdu", Master's Thesis, Department Of Computer Science, University of Peshawar Pakistan, Proceedings of the Conference on Language & Technology, 2009

[14] Rodolfo A. Pazos Rangel, Alexander Gelbukh, J. Javier González Barbosa, Erika Alarcón Ruiz, Alejandro Mendoza Mejía, and A. Patricia Domínguez Sánchez "Spanish Natural Language Interface for a Relational Database Querying System", P. Sojka, I. Kopeˇcek, and K. Pala (Eds.): TSD 2002, LNAI 2448, pp. 123–130, 2002., Springer-Verlag Berlin Heidelberg, 2002

[15] Tihitina Petros Degsew "Modeling and Designing Amharic Query System to Bilingual (English-Amharic) Databases", MSc thesis, Addis Ababa University, College of Natural Science, Department of Computer Science, 2014

[16] Desmond Bala "Design Science Research Methodology in Computer Science and Information Systems", International Journal of Information Technology, 2014

[17] Scriber Team, "How to write a literature review", on https://www.scribbr.com, Amsterdam, The Netherlands, last access on August 12/2020

[18] Ashish Kumar and Kunwar Singh Vaisla "Natural Language Interface to Databases: Development Techniques", Department of Computer Science and Engineering, BTKIT, Dwarahat, Almora, Uttarakhand, India, Elixir Comp. Sci. & Engg. 58 (2013) 14724-14727, 2013,

[19] Aditya Jain, Gandhar Kulkarni, Vraj Shah "Natural Language Processing" JCSE international journal of Computer Sciences and Engineering, Vol-6, Issue-1, India, 2018

[20] Rajendra Akerkar and Manish Joshi, "Natural Language Interface Using Shallow Parsing"; International Journal of Computer Science and Applications, Vol. 5, No. 3, pp 70 – 90, 2005

[21] Neelu Nihalani, Dr. Sanjay Silakari, Dr. Mahesh Motwani "Natural language interface for Database; A Brief Review", IJCSI International Journal of Computer Science Issues, Vol.

8, Issue 2, March 2011

[22] Siasar djahantighi F, Norouzifard M, Davarpanah S H, Shenassa M H. “Using natural language processing in order to create SQL queries.” In: IEEE International Conference on Computer and Communication Engineering (ICCCE); 13-15 May 2008; Kuala Lumpur, Malaysia: IEEE. pp. 600 - 604.

[23] Green B. F., Wolf A. K., Chomsky C., Laughery K., “BASEBALL: An Automatic Question Answerer”, Proceedings of the Western Joint Computer Conference, Volume 19, pp 219–224. Reprinted in (Grosz et al., 1986), pp. 545–549

[24] Woods, W., Kaplan R., N-Nebber B., “The Lunar Sciences Natural Language Information System”, BBN Report 2378, Bolt Beranek and Newman, Cambridge, Massachusetts, 1972

[25] Waltz, D., “Natural Language Access to a Large DataBase: An Engineering Approach”, Proc. 4th International Joint Conference on Artificial Intelligence, Tbillsl, USSR, pp. 868-872, 1975

[26] Hendrix G., Sacerdoti E., Sagalowicz D., Slocum, J., “Developing a Natural Language Interface to Complex Data”, ACM Transactions on Database Systems, 3(2), pp. 105-147, 1978

[27] Warren, D., “Efficient Processing of Interactive Relational Database Queries Expressed in Logic”, Proc. Seventh International Conference on Very Large Data Bases”, Cannes, France, pp. 272-283, 1981

[28] Moore, R., “Problems in Logical Form”, Proceedings of the 19th Annual Meetings of the Association for Computational Linguistic, California June 1981, pp. 117-124, 1981

[29] Robinson J., “DIAGRAM: A Grammar for Dialogues", Communications of ACM, Vol. 25, No. 1, pp. 27-47, 1982

[30] Ballard B., Stumberger D., “Semantic acquisition in TELI”, Proceeding of the 24th Annual meeting of ACL, New York, pp. 20-29, 1986.

[31] Bobrow R., Resnik P., Weischedel R., “Multiple underlying systems: translating user request into programs to produce answers”, Proceeding of the 28th Annual meeting of ACL, Pittsburgh, pp. 227-234, 1990

[32] Yunyao Li, Huahai Yang, and H. V. Jagadish “NaLIX: A generic natural language search environment for XML data”, University of Michigan, 2006

[33] P.R. Cohen. “The Role of Natural Language in a Multimodal Interface”, Technical Note

514, Computer Dialogue Laboratory, SRI International, 1991

[34] Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates "Modern Natural Language Interfaces to Databases: Composing Statistical Parsing with Semantic Tractability", University of Washington, 2004

[35] Yuk Wah Wong, Raymond J. Mooney "Learning for semantic parsing with statistical machine translation" the University of Texas at Austin, 2006

[36] Miikkulainen R., "Natural language processing with sub-symbolic neural networks", Neural Network Perspectives on Cognition and Adaptive Robotics, 1997

[37] Eric D. Brill "A Corpus-Based Approach to Language Learning", IRCS Technical reports Series 93-94, Institute for Research in Cognitive Science, University of Pennsylvania, 1993

[38] Richard Burton, "Semantic Grammar: An Engineering Technique for Constructing Natural Language Understanding Systems", Cambridge, Mass, 1976

[39] A. Purwarianti, M. Tsuchiya, and S. Nakagawa, "A machine learning approach for Factoid Question Answering," Proc. IASTED AIA, pp. 131–136, 2007

[40] Lior Rokach, Oded Maimon "Decision Trees", Department of Industrial Engineering, Tel-Aviv University,2005

[41] Lior Rokach, Oded Maimon "Data Mining with Decision Trees theory and applications", series in Machine Perception and Artificial intelligence, vol.69, World scientific publishing Co.Pte.Ltd, Singapore, 2008

[42] Yaregal Assabie, Josef Bigun "Multifont size-resilient recognition system for Ethiopic script", School of Information Science, Computer and Electrical Engineering, Halmstad University, Sweden, 2007

[43] Ethiopian Treasures, http://www.ethiopiantreasures.co.uk/pages/language.htm, last accessed on August 15, 2020

[44] ThoughCo "Word Class in English Grammar", https://www.thoughtco.com/word-class-grammar-1692608 last accessed on August 17, 2020

[45] Daniel Gochel Agonafer "An Integrated Approach to Automatic Complex Sentence Parsing for Amharic Text", Master's Thesis, School of Graduate Studies of Addis Ababa, 2003.

[46] Baye Yimam "Yamariñña Sáwasáw (Amharic Grammar)" Addis Ababa, EMPDA, 1987 (E.c)

[47] Mersehazen Woldeqirqos “yäamarð säwäsä”, Birhanena Selam Printing Press, Addis Ababa, 1934 E.c

[48] Mesfin Getachew. 2001. Automatic Part of Speech Tagging for Amharic Language: An Experiment Using Stochastic Hidden Markov (HMM) Approach, Master Thesis at School Of Information Studies for Africa, Addis Ababa.

[49] Charles William “Amharic Language”, Microfilm-Xerography, University of Microfilms London, 1965

[50] Getahun Amare “Zamanawi yaamarEna Sawasaw baqalal aqaararab” Commercial Printing Press, Addis Ababa, 1998

[51] F.P. Cotterell “Control in Amharic”, Journal of Ethiopian Studies, Institute of Ethiopian Studies, vol.11, No.1, pp. 75-106, 1973

[52] Prabhudev Konana, “Structured Query Language (SQL): A Primer on Data Definition Language (DDL)”, Ph.D. Thesis, the Graduate School of Business, the University of Texas at Austin, 2000

[53] Spring “Spring Boot”, https://spring.io/projects/spring-boot last accessed on August 18, 2020

[54] Craig Walls “Spring Boot in Action”, Manning Publications Co, ISBN 9781617292545, The United State of America, 2016

[55] Anwar Indris Jemal “Query Constructor Framework for Web-Based Search interface to Relational databases”, MSc thesis, Addis Ababa University, College of Natural Science, Department of Computer Science, 2015

[56] Ashenafi Tekelemariam “Amharic Dialog system for e-Banking Customer Service Support Based on Natural Language Processing Techniques”, MSc thesis, Addis Ababa University, College of Natural Science, Department of Computer Science, 2018

[57] TechBeamers retrieved from “ https://www.techbeamers.com/sql-query-questions-answers- for-practice/”, India, last Accessed on August 07, 2020, 9:00 AM

[58] Bootstrap made retrieved from “https://bootstrapmade.com/”; last accessed April 20, 2020

Annexes

Annex A- Algorithm for graphical user interface

```
Input: AmSentence, Db, Model type mdl, sqlEnglish
Output: Required information
Initialize IsExecute variable to a false, Export variable to
false, the verb to empty string, ErrorFlag to false,
ExecutedResult to false, SuccessFlag to false
If not IsExecute
          FindQuery()
Else
          ExecuteQuery()


Method FindQuery()
Try
    Initialize rslt to false, amQry to false, error variable to
    false, x to 0,line to empty string
    Assign AmSentence to Sentence, DatabaseName to Db
    Initialize pythonPath to DataCollection.py path
    Create array name cmd having 4 indexes
    Assign cmd[0] to python,cmd[1] to pythonPath,cmd[2] to
    Sentence,cmd[3] to Db
    Create RunTime variable rt,create process variable pr
    Execute the array with rt and assign the result to pr
    Create BufferedReader object bfr and read the Input stream of pr
    While bfr.readLine() is not null
             If x is 2
                    Assign bfr.readLine() value to rslt
              Else if x is 1
                    Assign bfr.readLine () value to amQry
              Else if x is 0
                     Assign bfr.readLine () value to error variable
              Else if x is 4
                     Assign bfr.readLine () value to verb
             Increment x by one
     End while
     ErrorOrSuccess ()

Catch
   Assign mdl attribute of the error to "Execution Error"
   Assign mdl attribute of InputQuestion to Sentence
   Set IsExecute to false, ExecuetedResult to false, ErrorFlag to true
```

```
   Assign mdl attribute of InputQuestion to Sentence
   Assign mdl attribute of btnExecute to IsExecute
   Assign mdl attribute of ExecutedResult to IsExecute
   Assign mdl attribute of ErrorFlag to ErrorFlag

Method ErrorOrSuccess()
If x is 1
    Assign mdl attribute of the error to error variable
    Set IsExecute to false, ExecuetedResult to false,ErrorFlag to True
    Assign mdl attribute of InputQuestion to AmSentence
    Assign mdl attribute of btnExecute to IsExecute
    Assign mdl attribute of ExecutedResult to IsExecute
    Assign mdl attribute of ErrorFlag to ErrorFlag
Else if x is greater than 1
     Concatenate "AM query:" with amQry and assign to amQry
     Concatenate "ጥያቄ:" with amQuestion and assign to amQuestion
     Assign mdl attribute of AmQuery to amQry
     Assign mdl attribute of EngQuery to rslt
     Assign mdl attribute of AmQuestin to amQuestion
     Set IsExecute to true,ExecuetedResult to false,ErrorFlag to false
     Assign mdl attribute of btnExecute to IsExecute
     Assign mdl attribute of ExecutedResult to IsExecute
     Assign mdl attribute of ErrorFlag to ErrorFlag

Method ExecuteQuery()
Initialize session variable to null,transation variable to null,qry
to sqlEnglish
Try
      If Db equals "DPOIS"
             Call getSessionFactoryMSSql() and assign the result to
             session variable
       Else
             Call getSessionfactoryOracle() and assign the result to
             session variable
       SelectOrUpdateOrDelete()
Catch
     Assign mdl attribute of error to "Execution Error"
     Assign mdl attribute of InputQuestion to Sentence
     Set IsExecute to false,ExecuetedResult to false,ErrorFlag to true
     Assign mdl attribute of InputQuestion to Sentence
     Assign mdl attribute of btnExecute to IsExecute
     Assign mdl attribute of ExecutedResult to IsExecute
     Assign mdl attribute of ErrorFlag to ErrorFlag
```

```
Method SelectOrUpdateOrDelete()
If the verb is equaled to "ምረጥ"
     Begin transaction using session.beginTransaction()
     Execute native query using createNativeQuery(qry) assign to
     outputQuery
     Map the outputQuery column name to table name using
ALIAS_TO_ENTITY_MAP
     Convert the outputQuery to list and assign to the output
     Commit the transaction
     Set the export variable to true
     Concatenate "ጥያቄ:" with Sentence and assign to amQuestion
     Set IsExecute to false, ExecuetedResult to true
     Assign mdl attribute of Output to output
     Assign mdl attribute of Export to export
     Assign mdl attribute of AmQuestion to amQuestion
     Assign mdl attribute of btnExecute to IsExecute
     Assign mdl attribute of ExecutedResult to IsExecute
     Assign mdl attribute of EngQuery to qry
Else
      Begin transaction using session.beginTransaction()
      Execute query using createNativeQuery(qry).executeupdate assign
      to NoOfQuery
      Commit the transaction
      If NoOfQuery >0
             Set IsExecute to false, ExecuetedResult to
             false, SuccessFlag to true
             Assign mdl attribute of InputQuestion to Sentence
             Assign mdl attribute of btnType to IsExecute
             Assign mdl attribute of ExecutedResult to ExecutedResult
             Assign mdl attribute of SuccessFlag to SuccessFlag
```

Annex B- Algorithm for stemmer

```
Create and initialize originalword to word
Create and initialize Prefix to false
Create and initialize Suffix to false
Create and initialize PrefixIndex to -1
Create and initialize SuffixIndex to -1
Create and initialize stem to the empty string
If the first character of the word is 'ያ'
          Convert 'ያ' into 'የአ', concatenate with the remaining
character and assign to the word
If a word is not in punctuation and word is not in digits and word is
not in ascii_letters and word is not in stop_punctuations and word is
not in Wanted_punctuations
              FindPrifixAndSuffixIndex ()

Method FindPrefixAndSuffixIndex ()
If the length of a word is greater than or equal to 3
          If the length of the word is 3 and equal to 'ከተም'
                    Add 'ከ' before the word and assign to a word
                    Create and initialize newword to word,subchar to 0
          If the length of neword is greater than 5
                               Set subchar=4
          Else if the length of newword is equal to 5
                               Set subchar=3
          Else if the length of newword is equal to 4
                                Set subchar=2
          Else if the length of newword is equal to 3
                                 Set subchar=1
          Else if the length of newword is equal to 2
                                 Set subchar=0
For j starting from subchar up to -1
      If the subword starting from 0 up to j+1 index is in Noun_Prefix
          Set Prefix to true
          Set PrifixIndex to j
          Break out of the for loop
      If the length of the word starting from PrifixIndex+1 is greater
      than or equal to 3
          Assign subword from PrifixIndex+1 to newword
          If the length of neword is greater than 5
                     Set subchar=4
          Else if the length of newword is equal to 5
                      Set subchar=3
          Else if the length of newword is equal to 4
                      Set subchar=2
          Else if the length of newword is equal to 3
```

```
                    Set subchar=1
          Else if the length of newword is equal to 2
                    Set subchar=0
For k starting from length of a word -subchar up to a length of a word
          If the subword starting from k up to a length of word +1
          is in Noun_Prefix
                   Set Suffix to true
                   Set SuffixIndex to k
                   Break out of the for loop
                   CutThePrefixAndSuffix ()

Method CutThePrefixAndSuffix()
If Prefix and Suffix is true
    Take the word from PrifixIndex+1 to SuffixIndex and assign to word
    Take the word from 0 to length of word -1
    ConvertCharacter(last character,5) and assign to lastchr
    Concatenate word and lastchr
If Prefix is true and Suffix is false
    Take the word from PrifixIndex+1 to the length of the word and
    assign to word
    Take the word from 0 to length of word -1
    ConvertCharacter(last character,5) and assign to lastchr
    Concatenate word and lastchr
If Prefix is false and Suffix is true
    Take the word from 0  to SuffixIndex and assign to a word
    Take the word from 0 to length of word -1
    ConvertCharacter(last character,5) and assign to lastchr
    Concatenate word and lastchr
If Prefix is false and Suffix is false
    Take the word from 0 to length of word -1
    ConvertCharacter(last character,5) and assign to lastchr
    Concatenate word and lastchr
    If it is for guess
       If a word is found in Stem_word_for_guess
            Set stem to word
    If it is for AllKey
        If a word is found in Key_word_All
            Set stem to word
    If it is for Each
        If a word is found in Key_word_Each
            Set stem to word
    If it is for Order
        If a word is found in Key_word_Order
            Set stem to word
    If it is not for guess and AllKey and Each and Order
```

```
        If a word is found in Stem_Noun
             Set stem to word

Method ConverAmharicCharacter()
Input: givenCharacter, IndexNeeded
Output:NewCharacter
If givencharacter is not found in punctuation, ASCII _letters, and
digits
          Find the Unicode of a givenCharacter and assign it to
          uniGivenCharacter
          Find the remainder when uniGivenCharacter divide by 4608 and
          assign it to Reminder
          If Reminder is greater than or equal to 8
              Find the remainder when Reminder divide by 8 and
              assign it to Reminder
          If Reminder is greater than or equal to IndexNeeded
              Subtract IndexNeeded from Reminder and assign it to subt
              Subtract subt from uniGivenCharacter and assign it to
              uniNewCharacter
              Find the character representation of uniNewCharacter and
              assign it to NewChar
          Else
              Subtract IndexNeeded from Reminder and assign it to subt
              Add subt and uniGivenCharacter and assign it to
              uniNewCharacter
              Find the character representation of uniNewCharacter and
              assign it to NewChar
Return NewChar
```

Annex C- Algorithm for number identification and conversion

```
Inputs: split words & split stem words
Output: words as digits & split stem words
Create and initialize NumberCorr1 to 0, NumberCorr to 0,Create
and initialize firstIndex to -1, RemovedCharacter to 0, j to
0,Create and initialize val to None, val2 to None,val3 to None,
While j is less than length of split words
    Search stem of word in OneDigits dictionary and assign
    value to val
    If val is None then Search stem of word in TwoDigits
    dictionary and assign value to val2
    If val2 is none then Search stem of word in ThreeDigits
    dictionary and assign value to val3
    If val is not None and j is less than length of the spit
     words minus 1
          If firstIndex is -1 then set firstIndex to j
          Call ConvertToDigit(val,stem word at j) and
          assign to NumberCorr
          If NumberCorr is not equal to 0 then
          Increment j by two, increment RemovedCharacter by two
     Else if val2 is not none and j is less than length of the
     spit words minus 1
           If firstIndex is -1 then set firstIndex to j
           Call ConvertToDigit(val2,stem word at j) and assign
           to NumberCorr
           If NumberCorr is not equal to 0
                 Increment j by two, increment
                 RemovedCharacter by two
     Else if val3 is not none and j is less than length of the
     spit words minus 1
           If firstIndex is -1 then set firstIndex to j
           Call ConvertToDigit(val3,stem word at j) and assign
           to NumberCorr
           If NumberCorr is not equal to 0
                       Increment j by two, increment
                       RemovedCharacter by two
           If NumberCorr is not equal to 0
                 Sum NumberCorr1 and NumberCorr and assign to
                 NumberCorr
                 Assign NumberCorr to NumberCorr1
           Else
```

```
                Create and initialize v to 0
                If val is not none then assign val to v
                Else if val2 is not none then assign val2 to v
                Else if val3 is not none then assign val3 to v
                If v is not equal to 0
                Sum NumberCorr1 and v and assign to NumberCorr
                Assign NumberCorr to NumberCorr1,
                increment RemovedCharacter by one
                If v is equal to 0 and firstIndex is not equal
                to -1 then break out of the loop
                Increment j by one,set val ,val2 ,val3 to
                none,NumberCorr to 0
                Remove converted words from the sentence
                and from the stem sentence

method ConvertToDigit()
Input: firstDigit,SecondLetter,sum initialized to false
Output: number in digit
Create and initialize NumberOfZeros to empty string
Create and initialize NumberCorr to 0
If sum is true
          Search Secondletter in OneDigit dictionary and assign
          the value to val
          If val is not none
               Sum firstDigit and val , assign the sum to
               NumberCorr
Else
          If  SecondLetter is found in ThreeDigits Dictionary
          then Set NumberOfZeros='00'
          Else if  SecondLetter is found in FourDigits
          Dictionary then Set NumberOfZeros='000'
          Else if  SecondLetter is found in SevenDigits
          Dictionary then Set NumberOfZeros='000000'
          Convert firstDigit in to string and assign it to
          NumberCorr
          Concatenate NumberCorr and NumberOfZeros and assign to
          NumberCorr
Return NumberCorr
```

Annex D- Dictionary of Numbers

```
Onedigits={0:'ዜሮ',1:'አንድ',2:'ሁለት',3:'ሶስት',4:'አራት',5:'አምስት',
           6:'ስድስት',7:'ሰባት',8:'ስምንት',9:'ዘጠኝ'}
Twodigits={10:'አስር',20:'ሀይ',30:'ሰላስ',40:'አርብ',50:'ሀምስ',60:'ስልስ'
           ,70:'ሱብ' ,80:'ሰማንይ',90:'ዘጠን'}
Threedigits={100:'መት'}
Fourdigits={1000:'ሺህ'}
Sevendigits={1000000:('ሚሊየን', 'ሚሊዮን')}
```

Annex E- Algorithm for concatenation

```
Input: split word, split stem word
Output: a split word with concatenation, split stem word with
concatenation, stem order keys
While m is less than the length of a split word
      Create and set word to split stem word at m
      If m is greater than 0 and the word is found in
      WordsForConcatenation:
            Find a stem of the previous word and assign to stem
            If the stem is found in WordsToConcatenation
            dictionary
                  Concatenate word at m and m-1 by hyphen and
                  assign to compoundword
                  Concatenate stem word at m and m-1 and assign
                  to compoundstem
                  Set word at m to compoundword
                  Set stem word at m to compoundstem
                  Remove word and stem word at m
                  Decrement m by one
                  If a stem word at m is not empty and found in
                  WordsToConcatenation dictionary
                        Concatenate word at m and m-1 by hyphen
                        and assign to compoundword
                        Concatenate stem word at m and m-1 and
                        assign to compoundstem
                        Set word at m to compoundword
                        Set stem word at m to compoundstem
                        Remove word and stem word at m
                        Decrement m by one
                                    .
                                    .
                                    .
        Increment m by one
Find a stem of order key if they present in the split words and
assign to orderkeys
Find also order sequence type if it present in the split words
and assign to orderSequence
Set orderFlag to true if the order indication keyword found in
split words and assign to Hasorder
Concatenate order keys by a hyphen and remove the old order keys
Return split word, split stem word, split stem order keys
```

Annex F- Algorithm for POS Tag determiner

```
Input: sentence array, index
Output: pos tag
Create and initialize Pos variable to 'N',prefix to false,suffix
to false,prifixIndex to -1,
suffixIndex to -1, word to sentence array at index
if (stem of a word is found in Key_word_All or in Key_word_count
or in Key_word_Each or in Key_word_Order or in
Key_word_Condition or in Key_word_Aggregate_function)
         set pos to 'K'
Else if the stem of a word is found in Key_word_conjuctions then
set pos to 'C'
Else if the stem of a word is found in Stop_word then set pos to
'S'Else
           If word if found in verb_select_Underived
                     Set pos to 'V1', quite out of the function
           If the length of a word is greater than or equal to 3
           Create and initialize newword to word,subchar to 0
           If the length of newword is greater than or equal to
           5 then Set subchar=4
           Else if the length of newword is equal to 5 then Set
           subchar=3
           Else if the length of newword is equal to 4 then Set
           subchar=2
           Else if the length of newword is equal to 3 then Set
           subchar=1
           Else if the length of newword is equal to 2 then Set
           subchar=0
           For j starting from subchar up to -1
                    If the subword starting from 0 up to j+1
                    index is found in Prefixes
                       Set Prefix to true, Set PrifixIndex to j,
                       Break out of the for loop
            If the length of the word starting from
            PrifixIndex+1 is greater than or equal to 3
                  Assign subword from PrifixIndex+1 to
                  newword, set subchar to 0
                  If the length of neword is greater than or
                  equal to 7 then Set subchar=5
                  Else if length of newword is equal to 6 then
                  Set subchar=4
```

```
                    Else if length of newword is equal to 5 then
                    Set subchar=3
                    Else if length of newword is equal to 4 then
                    Set subchar=2
                    Else if length of newword is equal to 3 then
                    Set subchar=1
                    Else if length of newword is equal to 2 then
                    Set subchar=0
                    For k starting from length of a (word –
                    subchar) up to the length of a word
                          If the subword starting from k up to
                          length of word +1 is in Suffix
                                  Set Suffix to true, Set
                                  SuffixIndex to k
                                  Break out of the for loop
                    CutThePrefixAndSuffixVerb ()


Method CutThePrifixAndSuffixVerb()

If Prefix and Suffix is true
       If the next two characters are in prefixes then take a
       a word from preifixIndex+3 to SuffixIndex
       Else If the next one char is in prefixes then take a word
       from prefixIndex+2 to SuffixIndex
       Else if the word starts with 'አሳ' then take a word
      'አ'+from prefixIndex+2 to SuffixIndex
       Else Take the word from PrifixIndex+1 to SuffixIndex and
       assign to word
       Convert each character using ConvertCharacter(each
       character,5) and assign to the word
If Prefix is true and Suffix is false
       If the next two characters are in prefixes then take a
       A word from preifixIndex+3 to the end
       Else If the next one char is in prefixes then take a word
       from prefixIndex+2 to the end
       Else if the word starts with 'አሳ' then take a word
      'አ'+from prefixIndex+2 to the end
       Else Take the word from PrifixIndex+1 to SuffixIndex and
       assign to word
       Convert each character using ConvertCharacter(each
```

```
       character,5) and assign to a word
If Prefix is false and Suffix is true
       Take the word from 0  to SuffixIndex and assign to a word
       Convert each character using ConvertCharacter(each
       character,5) and assign to a word
If Prefix is false and Suffix is false
        set the word to sentence array at index
        Convert each character using ConvertCharacter(each
        character,5) and assign to the word
If a word is found in Verb_select then Set pos to 'V1'
Else if the word is found in Verb_update then set pos to 'V2'
Else if the word is found in Verb_delete then set pos to 'V3'
Else if the word is found in Verb_All then set pos to 'V'
Else st pos to N
Return pos
```

Annex G- Keywords

No	Keyword	Description
1	Key_Word_All	This keyword is used to identify words that are used for selecting all attributes or all information. E.g ሁሉንም
2	Key_Word_Top	This keyword is used to identify words that are used for selecting the nth first record of the table. E.g የመጀመሪያውን
3	Key_Word_Last	This keyword is used to identify words that are used for selecting the nth last record of the table. E.g የመጨረሻውን
4	Key_Word_Each	This keyword is used to identify words that are used for grouping records of the table. E.g በእያንዳንዱ
5	Key_Word_Aggregate_Count	This keyword is used to identify words that are used for counting records of the table. E.g ስንት
6	Key_Word_Aggregate_Sum	This keyword is used to identify words that are used for adding records of the table. E.g ድምሩን
7	Key_Word_Aggregate_Maximum	This keyword is used to identify words that are used for finding the maximum record of the table. E.g ትልቁን
8	Key_Word_Aggregate_Minimum	This keyword is used to identify words that are used for finding the minimum record of the table. E.g ትንሹን
9	Key_Word_Aggregate_Average	This keyword is used to identify words that are used for finding the average of the records. E.g አማካዩን
10	Key_Word_Order	This keyword is used to identify words that are used for displaying the record in some order either ascending or descending. E.g በቅደም ተከተል

Annex H- Conditions

No	conditions	Description
1	Key_word_condition_equal	This condition is used to identify words that are used for filtering the record based on equality.
2	Key_word_condition_not_equal	This condition is used to identify words that are used for filtering the record based on inequality.
3	Key_word_condition_greater	This condition is used to identify words that are used for filtering the record which is greater than some value.
4	Key_word_condition_less	This condition is used to identify words that are used for filtering the record which is less than some value.
5	Key_word_conditon_between	This condition is used to identify words that are used for filtering the record which are between some values.
6	Key_word_condition_start	This condition is used to identify words that are used for filtering the record which starts with some value.
7	Key_word_condition_end	This condition is used to identify words that are used for filtering the record which ends with some value.
8	Key_word_condition_InBetween	This condition is used to identify words which are used for filtering the record which contains some value

Annex I- Algorithm for clause determiner

```
Input: Tagged Sentence, features
Output: clause
Create and initialize clause to empty string, ActualV to false,
ActualV1 to false,
ActualV2 to false, ActualV3 to false
Unzip tagged sentence to Words and Tags
For each word in Words
     If the POS tag is V, then set ActualV to true
     Else if the POS tag is V1, then set ActualV1 to true
     Else if the POS tag is V2, then set ActualV2 to true
     Else if the POS tag is V3, then set ActualV3 to true
Create and set NoOfV to the number of 'V' in Tags
Create and set NoOfV1to the number of 'V1'
Create and set NoOfV2 to the number of 'V2'
Create and set NoOfV3 to the number of 'V3'
If NoOfV1 is greater than NoOfV2 and NoOfV3 then set clause to
'ምረጥ'
If NoOfV2 is greater than NoOfV2 and NoOfV3 then set clause to
'ለውጥ'
If NoOfV3 is greater than NoOfV2 and NoOfV1 then set clause to
'አጥፋ'
If NoOfV is greater than NoOfV1 and NoOfV2 and NoOfV3 then set
clause to 'ምረጥ'
If the clause is still empty and ActualV1 is true then set the
clause to 'ምረጥ'
Else if the clause is still empty and ActualV1 is true then set
the clause to 'ለውጥ'
Else if the clause is still empty and ActualV3 is true then set
the clause to 'አጥፋ'
If the clause is empty then
      For each word in Words
            If a word is in verb_select_question_word or in
(verb_select_undrived and it is at the end)
                           Set clause to 'ምረጥ'
                           Break out of the loop
If the clause is still empty then
          Display an alarm message to indicate that the system
is not able to get the table name
          Exit the system
Return clause
```

Annex J- Algorithm for table identification

```
Input: tagged sentence,database name,stem of words,clause
Output:table name,column , tagged sentence
Unzip tagged sentence into words and Tags
Create and initialize tablename to none, column to empty,
NoOfTable to the count of 'T' tag
If NoOfTable is 1
     take the stem of the word at index 'T' tag and assign to
     tablename
     find the root node of the tablename in the dictionary
     If tablename is not none and 'V1' is found in Tags or
     If (tablename is not none and 'V1' is not in Tags and the
     tablename does not begin with 'vw')
          concatenate database name, tablename,'columns' and
          set to column
    Else
           Convert the tag to list
           If the stem of the word at 'T' index is empty then
           modify this tag as 'S'
           Else if 'V1' is found in Tags
                If the next tag is 'V1' or the distance with \
                'V1' tag is less than 2, Modify this tag as 'R'
                Else if the nearest distance with a quotation is
                less than 2, Modify this tag as 'A'
                Else then modify this tag as 'S'
          Convert the tag to a tuple, zip it with words and
          assign to tagged sentence
          If tablename is none, call FurtherAnalysis() and
          assign to tablename,column,tagged sentence
Else if NoOfTable is greater than 1
        Take the words which are tagged as 'T' and assign to
        candidate
        take the stem of each candidate and assign to tablename
        find the root node of the tablename in the dictionary
        If tablename is not none and 'V1' is found in Tags or
        If (tablename is not none and 'V1' is not in Tags and
        the tablename not begins with 'vw')
            concatenate database name, tablename,'columns' and
            set to column
        Else
           Convert the tag to list
```

```
            If the stem of the word at 'T' index is empty then
            modify this tag as 'S'
            Else if 'V1' is found in Tags
                 If the next tag is 'V1' or the distance with
                 'V1' tag is less than 2, Modify this tag as 'R'
                 Else if the nearest distance with a quotation is
                 less than 2, Modify this tag as 'A'
                 Else then modify this tag as 'S'
                 Convert the tag to tuple, zip it with words and
                 assign to tagged sentence
       If tablename is none, call FurtherAnalysis() and assign
        to tablename,column,tagged sentence
Else then call FurtherAnalysis() and assign to tablename
If tablename is still none, then inform the user to modify the
question

Method Further analysis()
Input: tagged sentence, database, the stem of words, clause,
IsTableFlag
Output: tablename, column, tagged sentence
Unzip tagged sentence into words and Tags
Create and initialize tablename to none, column to empty
If the analysis is for the table then
    For m up to the length of the words
          Take the stem of the word at index m and assign to
          tablename
          Find the root node of the tablename in the dictionary
          If tablename is  none  then go to the next record
          Else
              If 'V1' is found in Tags or ('V1' is not in Tags
              and the tablename not begins with 'vw')
                  concatenate database name,
                  tablename,'columns' and set to column
                  convert Tags to list, set the tag at m to 'T'
                  convert the Tags to tuples, zip the words and
                  Tages and assign to Tagged sentence
                  break out of the loop
  If tablename is none then
        For each of the word, find the stem of the educated
        guess and assign to EducatedguessStem
        For each of the table in the selected database
```

```
                If the frequent words of the table1 found in the
                EducatedguessStem array
                       Set the tablename to table1
Return tablename
```

Annex K- Algorithm for attribute determiner

```
Input: tagged sentence, clause, table, Stem of words, features,
database, column, IsOrder, IsEach, last, top, IsAllKey, the stem
of order key
Output:attributeR,attributeA,additionalAttribute,ordeyBy,aggrega
teFunction,groupBy,condition,TaggedSentence,attributAIndex,attri
buteRIndex,aggregateFunctionIndex,conditionIndex,LastOrderBy,
LastOrderSequence
Unzip tagged sentence into words and Tags
Create and initialize attributeA to none, attributeR to none,
If the clause is ''ምረጥ' then
     If IsOrder indicates the order key words present
           For each of the word in words
                    Find and add index of word to orderIndex if
                    the stem of the order key has a value
                    Find and add index of word to ConjIndex if
                    conjunction present between order keys
                    Find and add index of word to BejIndex if
                    attibuteA or attributeR has 'በ'
                    Find the start and end index of the order
                    attributes expected to be positioned
                    Find and add to OrderBy if either attributeA
                    or attributeR found in those indexes
                    Find and add to aggregateFunction,groupby,
                    LastOrderBy,LastOrderSequence
     Check and set ConjuctionBtnWhereClause to true if
     conjuction present between where clause
     For each of the word tagged as 'R', find it in the
     dictionary and assign it to mkupword
              If mkupword is not none and not in groupBy then
                     Set attributeR to mkupword and add the
                     index to attrRIndex
              Else then set tags 'S' and zip words, tags to
              tagged sentence
    For each of the word tagged as 'A', find it in the
    dictionary and assign it to mkupword
              If mkupword is not none and not in groupBy then
                       Set attributeA to mkupword and add the
                      index to attrAIndex
               Else then set tags 'S' and zip words, tags to
               tagged sentence
```

```
If attributeA is none and if 'W' is found in the tag
       Make Further Analysis () and find and set values to
       attributA
If attributeR is none and if IsAllKey is false
       Make Further Analysis () and find and set values to
       attributR
Find the appropriate condition and add to condition,
conditionIndex
Find the appropriate top, last word and add to LastOrderBy,
LastorderSequence  if it is not found
```

Annex L- Algorithm for where condition determiner

```
Input: tagged sentence, clause, attrA, whrValue, attrR, dtb,
table, column, stem of words, attrAIndex, groupby,
additinalAttrbute
Output: whrCondition, ConjuctionBtnWhrCondition,tagged sentence,
attrA, attrR, attrAIndex, ConjuctionORIndex
Create and initialize NoOfWhrValue to count of 'W' tags,
NoOfAttrA to count of attrA, j to 0
Unzip tagged sentence into words, Tags
If NoOfAttrA is less than NoOfWhrValue
            Make further analysis to find filter attribute and
            assign the result to attrA
            Update NoOfAttrA
If still NoOfAttrA is less than NoOfWhrValue
        Create and initialize strIndex to -1, endIndex to -1,
        conjIndex to -1
        If the conjunction is found in Tags
                Check and find the repeated attrA and add to
                PossibleWhrCond
                Check and determine to move attribute from attrR
                and add to PossibleWhrCond
                Determine whether to take attr from defined
                attributes and add to PossibleWhrCond
While NoOfAttrA is not equal to NoofWhrValue and the length of
PossibleWhrCond greater than j
           If PossibleWhrCond at j is already in attrA then
                 Increment j by 1, go to the next possible
                 where condition
           Else
                 Find possible insertion index of a
                 PossibleWhrCond at j and assign it to InsertInd
                 Insert the possibleWhrCond into attrA at
                 InsertInd index
                 Increment j by 1
If NoOfAttrA is greater than NoOfWhrValue
               For k up to a length of Tag
                  Check the validity of attrA
                  Set the tag to 'R' is attrA at k is invalid
                  Join attrA with ',' and assign to whrCondition
```

```
If NofAttrA is greater than 2
          For each word in words
                If 'ወይም' is between where condition then
                      Add index of a word to ConjORIndex, set
                      ConjBnWhrCond to true
return whrCondition, ConjORIndex, attrA,attrR, taggedSentence,
       attrAIndex,ConjBnWhrCond
```

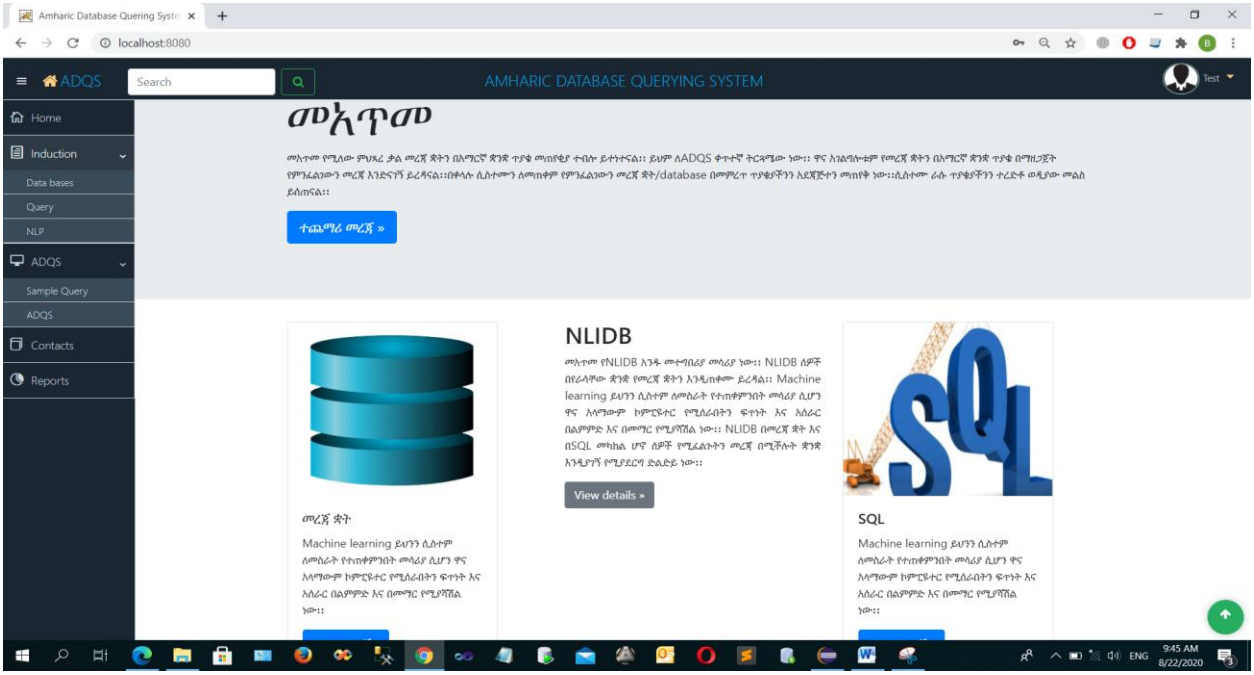
Annex M- Algorithm for query constructor

```
Input:clause,table,atrributR,attrbuteA,whrcondition,whrvalue,upd
atevalue,IsAllkey,Iseach,
OrderBy,aggrFunction,GroupBy,Condition,ConjBnWhrConditon,OrderSe
quence,
ConjORIndex,attrAIndex,Top,HavingWhrValueIndex,aggrfunctionIndex
,HavindWhrValue,
ConjIndex,whrClauseIndex,LastOrderBy,LastOrderSequence,
Output:queryEng,queryAmh
Create and initialize queryEng to empty string,queryAmh to empty
string
If lentgth of OrderyBy is greater > 0 concatenate orderBy with
OrderSeq and assign to orderBy
If length of GroupBy is greater than 0 Concatenate gropuby with
‘,’ and assign to groupby
If length of aggrFun is > 0 then Concatenate aggrFun, ‘(’,
attribute, ‘)’ based on aggrFunInd
If havWhrVal is >0 then Concatenate aggrFun, ‘(’, attribute, ‘)’
based on havValIndex
If WhrValue is not empty and HavingWhrValue is empty
     If condition is other than ‘=’
         If clause is ‘ምረጥ’
               If orderBy is greater than 0
                    Concatenate whrConditon,Condition,WhrValue
                    and assign it toWhrCondVal
                    Concatenate ‘select’,key
                    words,attrR,aggrFunc, ‘from’,table ‘where’,
                    whrCondVal,‘order by’, ordryBy depending on
                    The flag
                Else
                     Concatenate whrConditon,Condition,WhrValue
                     and assign it toWhrCondVal
                     Concatenate ‘select’,key
                     words,attrR,aggrFunc, ‘from’,table ‘where’,
                     whrCondVal  depending on The flag
          Else if clause  to ‘ለውጥ’
                      Concatenate whrConditon,Condition,WhrValue
                      and assign it toWhrCondVal
                      Concatenate attrA,‘=’,updateValue and
                      assign it toUpdCondVal
                      Concatenate ‘update’ ,table ,
```

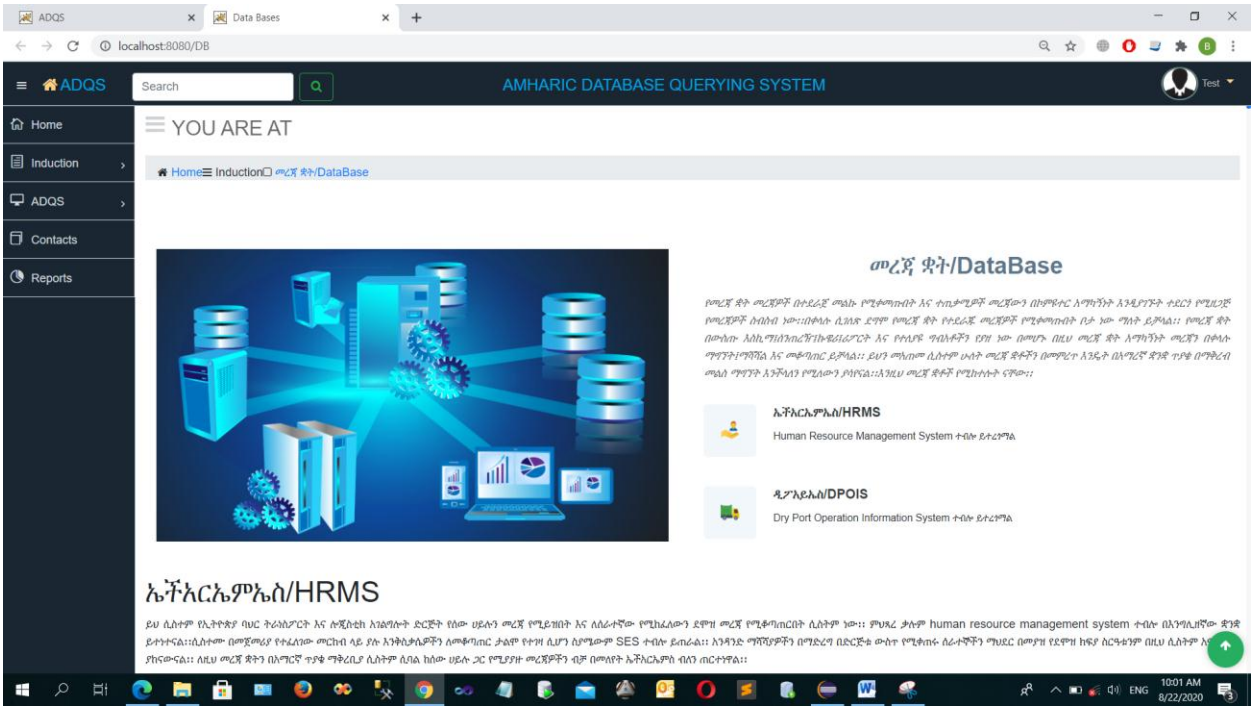
```
                    'set',updCondVal, 'where', WhrCondVal
        Else if clause  to 'አጥፋ'
                    Concatenate whrConditon,Condition,WhrValue
                    and assign it toWhrCondVal
                    Concatenate 'delete from' ,table ,
                    'where', WhrCondVal
    Else
        If the clause is 'ምረጥ'
                  If orderBy is greater than 0
                        Concatenate whrConditon,'=',
                        WhrValue and assign it toWhrCondVal
                        Concatenate 'select',key words,
                        attrR,aggrFunc,'from',table 'where',
                        whrCondVal,'order by', ordryBy
                        depending on The flag
                   Else
                        Concatenate whrConditon,=',WhrValue
                        and assign it toWhrCondVal
                        Concatenate 'select',key words,
                        attrR,aggrFunc,'from',table 'where',
                        WhrCondVal depending on The flag
         Else if the clause is 'ለውጥ'
                        Concatenate whrConditon,'=',
                        WhrValue and assign it toWhrCondVal
                        Concatenate attrA,'=',updateValue
                        and assign it toUpdCondVal
                        Concatenate 'update', table, 'set',
                        updCondVal, 'where', WhrCondVal
        Else if the clause is 'አጥፋ'
                        Concatenate whrConditon, '=',
                        WhrValue and assign it toWhrCondVal
                        Concatenate 'delete from', table,
                        'where', WhrCondVal
Else If WhrValue and HavingWhrValue are empty
           ConstructQuery()
Else If WhrValue  is not empty and HavingWhrValue is not empty
empty
           ConstructQuery()
Else If WhrValue is empty and  HavingWhrValue is not empty
           ConstructQuery()
```

Annex N- Some pages of ADQS

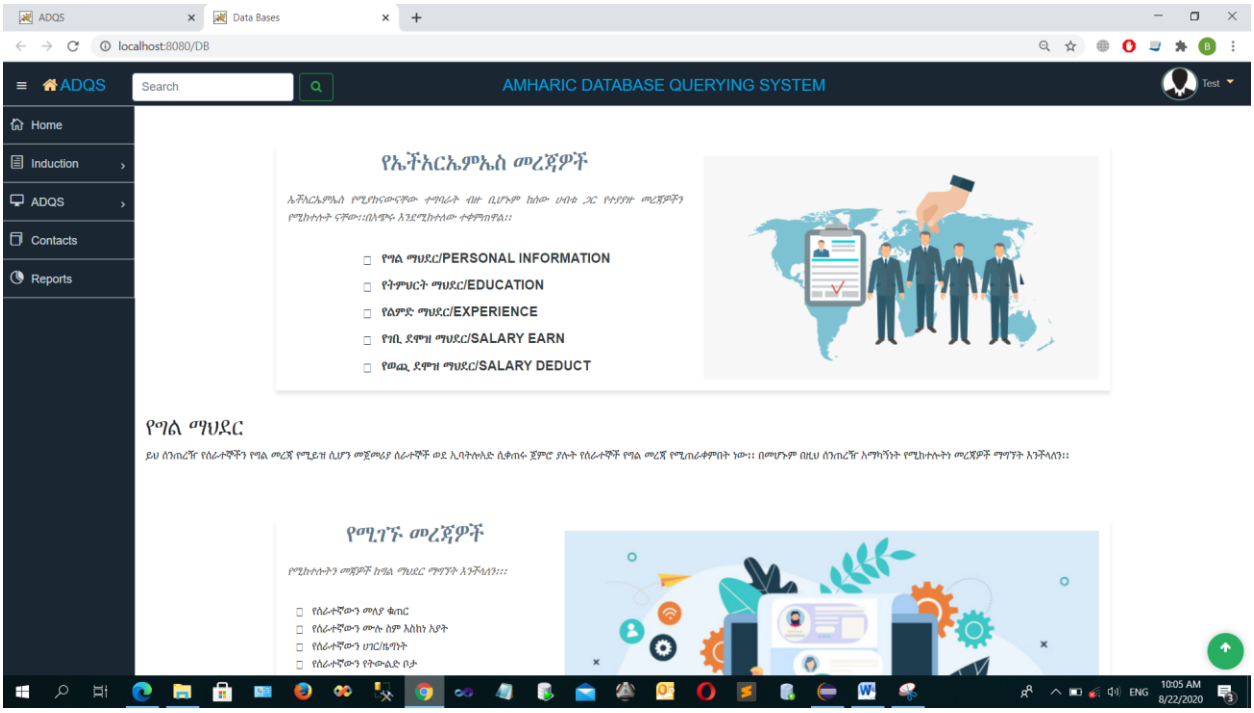
In this Annex, some page of the ADQS system which provides information to the user are indicated respectively



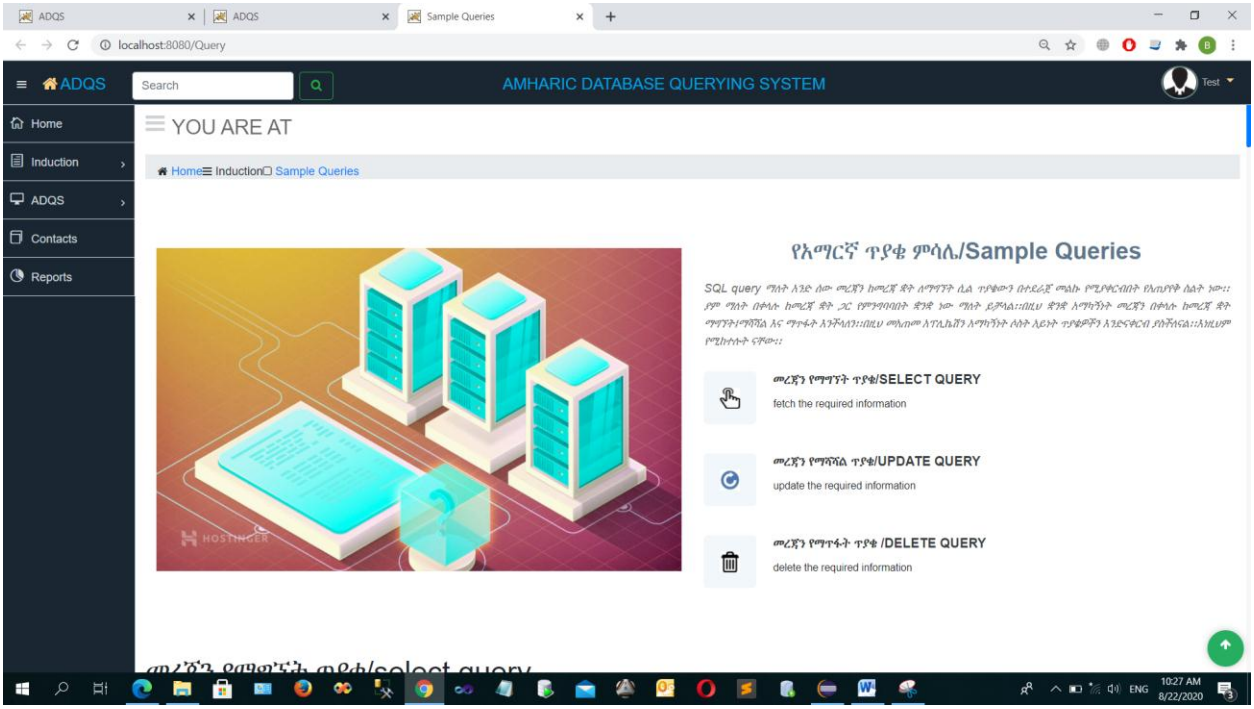
Index page with Amharic content



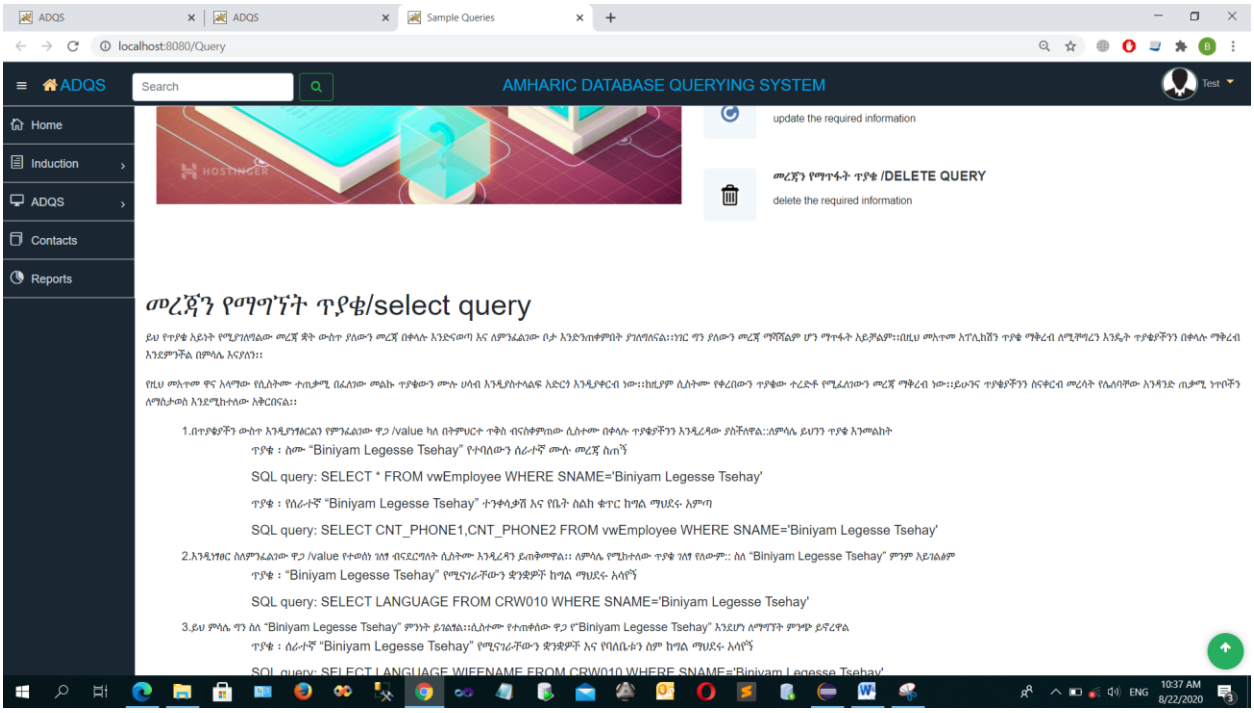
Database page



Database page giving some information to the user



Query page



Sample queries in ADQS

Declaration

I, the undersigned, declare that this thesis is my original work and has not been presented for a degree in any other university and that all sources of materials used for the thesis have been duly acknowledged.

Declared By:

Name: Beniyam Legesse Tsehay

Signature: _____

Date: _____

Confirmed By advisor:

Name: Dr. Yaregal Assabie

Signature: _____

Date: _____