

Interactive Natural Language Interface

Faraj A. El-Mouadib¹
Computer Science Department
Faculty of Information Technology
Garyounis University
Benghazi, Libya
<elmouadib@yahoo.com>

Zakaria Suliman Zubi²
Computer Science Department
Faculty of Science
Al-Tahadi University
Sirt, Libya
<zszubi@yahoo.com>

Ahmed A. Almagrous³
High Computer Technology Institute
Academy of Science
Benghazi, Libya
<almagroos@yahoo.com>

I. El-Feghi⁴
Electrical Engineering Department
Faculty of Engineering
Al-Fateh University
Tripoli, Libya.
<idrisel@gmail.com>

Abstract:- To override the complexity of SQL, and to facilitate the manipulation of data in databases for common people (not SQL professionals), many researches have turned out to use natural language instead of SQL. The idea of using natural language instead of SQL has prompted the development of new type of processing method called Natural Language Interface to Database systems (NLIDB). The NLIDB system is actually a branch of more comprehensive method called Natural Language Processing (NLP). In general, the main objective of NLP research is to create an easy and friendly environment to interact with computers in the sense that computer usage does not require any programming language skills to access the data; only natural language (i.e. English) is required.

Many systems have been developed to use the concept of NLP in different varieties of domains, for example the system LUNAR [19] and the system LADDER [8]. One drawback of previous systems is that the grammar must be tailor-made for each given database. Another drawback is that many NLP systems cover only a small domain of the English language questions.

In this paper we present the design and implementation of a natural language interface to a database system. The system is called Generic Interactive Natural Language Interface to Databases (GINLIDB). It is designed by the use of UML and developed using Visual Basic.NET-2005. Our system is generic in nature given the appropriate database and knowledge base. This feature makes our system distinguishable.

Keywords:- SQL, NLP, database, UML, NLIDB, DBMS, ATN.

1.0 Introduction

People via computers all around the world, access, accumulate and manipulate huge amount of data every second of the day. These huge amounts of data are located in private personal computers or remote location (i.e. the internet). Mostly, the data is stored in some kind of repository system such as database¹ and/or data warehouses². Data in database are usually managed by DBMS. The access to database is facilitated through a special interaction language called SQL or some version of it.

To override the complexity of SQL for non-professional, many researches have turned out to use Natural Language (NL) instead of SQL. The idea of using NL has prompted the development of new type of processing method called Natural Language Interface to Database systems (NLIDB). The NLIDB system is actually a branch of more comprehensive method called Natural Language Processing (NLP). The main objective of NLP research is to create an easy and friendly environment to interact with computers in natural language (i.e. English).

¹ A database is a collection of data, typically describing the activities of one or more related organizations. A database is a collection of related data. By data, we mean known facts that can be recorded and that have implicit meaning.

² A data warehouse is a repository of information collected from multiple sources, stored under a unified schema, and which usually resides at a single site. A data warehouse is a repository of multiple heterogeneous data sources, organized under a unified schema at a single site in order to facilitate management decision-making.

2.0 Early systems

The early efforts in the NL interfaces area started back in fifties [10]. Prototype systems had appeared in the late sixties and early seventies. Many of these systems relied on pattern matching to directly mapping the user input to the database [1]. Formal List Processor (FLIP) is an early language for pattern-matching based on LISP structure [16] works on the bases that if the input matches one of the patterns then the system is able to build a query for the database. In the pattern-matching based systems, the database details were inter-mixed into the code, limited to specific databases and to the number and complexity of the patterns. As the usage of databases has spread during the 1970's, the concept of user interface presented new challenges to the designers. One approach was the use of natural language processing, where the user interactively is allowed to interrogate the stored data.

2.1 LUNAR system

The system LUNAR [19] is a system that answers questions about samples of rocks brought back from the moon. The meaning of systems' name is that is in relation to the moon. The system was informally introduced in 1971. To accomplish its function the LUNAR system uses two databases; one for the chemical analysis and the other for literature references. The LUNAR system uses an Augmented Transition Network (ATN) parser and Woods' Procedural Semantics. According to [18], the LUNAR system performance

was quite impressive; it managed to handle 78% of requests without any errors and this ratio rose to 90% when dictionary errors were corrected. But these figures may be misleading because the system was not subject to intensive use due to the limitation of its linguistic capabilities.

2.2 LADDER

The LADDER system was designed as a natural language interface to a database of information about US Navy ships. According to [8], the LADDER system uses semantic grammar to parse questions to query a distributed database. The system uses semantic grammars technique that inter-leaves syntactic and semantic processing. The question answering is done via parsing the input and mapping the parse tree to a database query. The system LADDER is based on a three-layered architecture. The first component of the system is for Informal Natural Language Access to Navy Data (INLAND), which accepts questions in a natural language and produces a query to the database. The queries from the INLAND are directed to the Intelligent Data Access (IDA), which is the second component of LADDER. According to [7], the INLAND component builds a fragment of a query to IDA for each lower level syntactic unit in the English language input query and these fragments are then

combined to higher level syntactic units to be recognized. At the sentence level, the combined fragments are sent as a command to IDA. IDA would compose an answer that is relevant to the user's original query in addition to planning the correct sequence of file queries.

The third component of the LADDER system is for File Access Manager (FAM). The task of FAM is to find the location of the generic files and manage the access to them in the distributed database. The system LADDER was implemented in LISP. At the time of the creation of the LADDER system was able to process a database that is equivalent to a relational database with 14 tables and 100 attributes.

2.3 CHAT-80

The system CHAT-80 [17] is one of the most referenced NLP systems in the eighties. The system was implemented in Prolog. According to [2], the CHAT-80 was an impressive, efficient and sophisticated system. The database of CHAT-80 consists of facts (i. e. oceans, major seas, major rivers and major cities) about 150 of the countries world and a small set of English language vocabulary that are enough for querying the database. The CHAT-80 system processes an English language question in three stages as depicted in Figure-1.



Figure-1: CHAT-80 processing scheme.

The system translates the English language question by the creation of a logical form as processes of three serial and complementary functions where:

1. Words are represented by logical constants.
2. Verbs, nouns, and adjectives with their associated prepositions are represented by predicates. The predicates can have one or more arguments.
3. Complex phrases or sentences are represented by conjunctions of predicates.

These functions are being; parsing, interpretation and scoping. The parsing module function determines the grammatical structure of a sentence and the interpretation and scoping consist of various translation rules, expressed directly as Prolog clauses. The basic strategy followed by Chat-80 is to append some extra control information to the logical form of a query in order to make it an efficient piece of Prolog program that can be executed directly to produce the answer. According to [17], the generated control information comes into two forms:

1. Orders the predication for a query that will determine the order in which Prolog will attempt to satisfy them.
2. Separates the over all program into a number of independent sub problems to limit the amount of backtracking performed by Prolog.

In this way, Prolog is led to answer the queries in an obviously sensible way and the Prolog compiler can compile the transformed query into code that is as efficient as iterative loops in a conventional language.

3.0 The GINLIDB system architecture

Here, we present the architecture, design and implementation steps of the Interactive Generic Natural Language Interface to Database (GINLIDB) system. The architecture of the GINLIDB system consists of two major components:

3. Linguistic handling component and
4. SQL constructing component.

The first component controls the natural language query correctness as far as the grammatical structure and the possibility of successful transformation to SQL statement. The second component generates the appropriate SQL statement, opens a connection to the database in use, executes the generated SQL statement and returns the query's result to the user. Figure-2 depicts the over all architecture of the GINLIDB system.

3.1 Graphical User Interface (GUI)

The user interact with it GINLIDB system in a user-friendly environment where no knowledge of computers and database terms are required. The interaction with our system is via suitable visual forms, buttons, and menus.

3.2 Linguistic handling component

The Linguistic handling component consists of three parts: Lexical analysis, Parser, and Semantic representation.

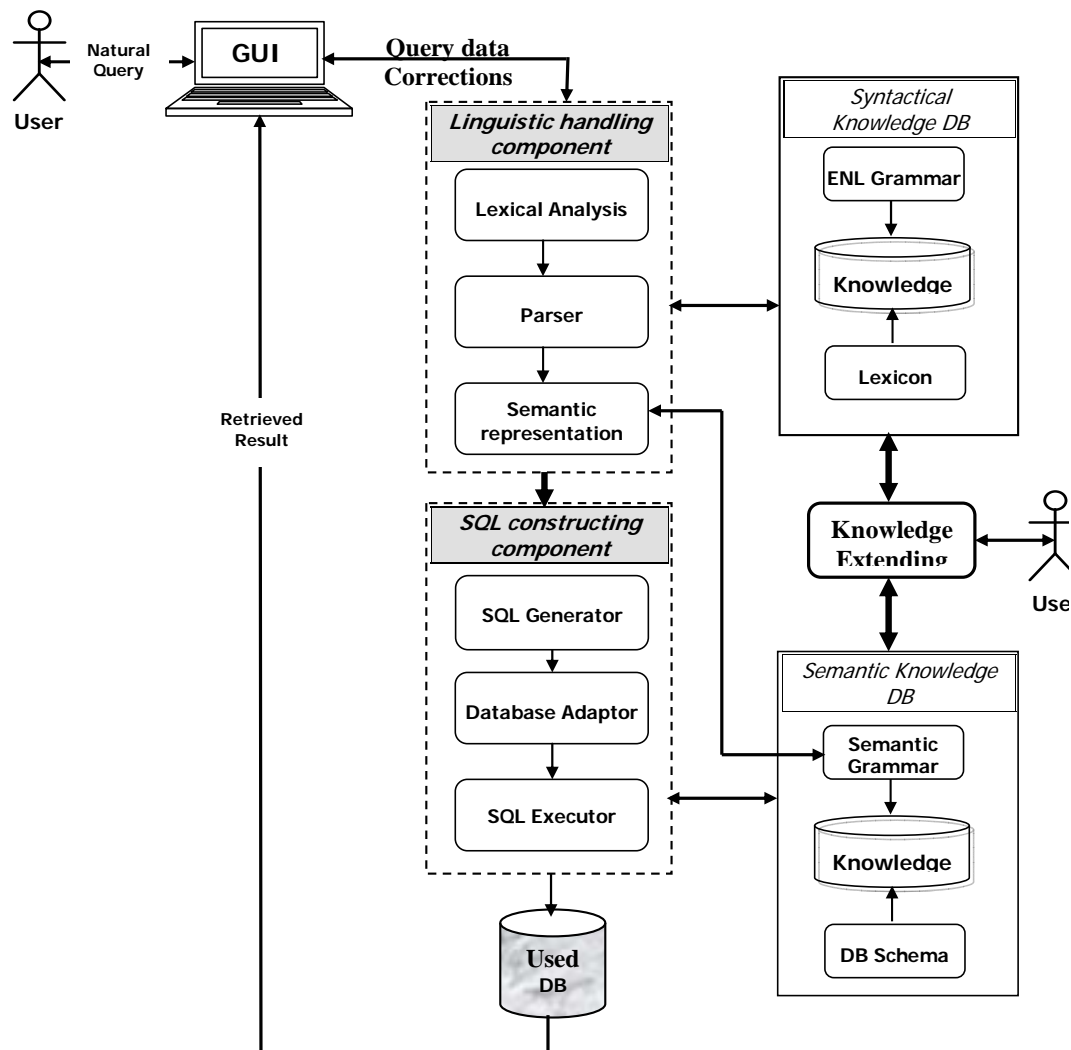


Figure-2: The architecture of the GINLIDB system.

3.2.1 Lexical analysis

This step to divide the sentence it into simpler elements that called tokens (i.e. in natural language query the elements are words and/or punctuations). This process is performed by the following functions:

- **Token analyzing** function is used to split the input string into a sequence of primitive units called tokens that is treated as a single logical unit.
- **Spelling checker** function makes sure that each token is in the systems' dictionary (lexicon) and if this is not the case then the spelling correction is performed or new words are added to the systems' vocabulary.
- **Ambiguity reduction** function reduces the ambiguity in a sentence and simplify the task of the parser, the system substitutes multiple words or symbols

with a canonical internal phrases or words, such as the ones in table-1.

Table-1: An example of canonical words substitution.

Normal words	Canonical words
First name	First_Name
, (comma)	AND
/ (slash in a none date token)	Or

- **Excessive tokens removal** function removes the additional tokens that will not affect the meaning of the query.

3.2.2 GINLIDB Parser

The Natural language queries are not easily parsed by programs, as there is substantial ambiguity in the structure of natural language queries. The GINLIDB parser is designed with two stages of grammars: lexical and syntactic. The first

stage is the token generation (lexical analysis) where the input tokens' stream is split into meaningful symbols. The second stage is a syntactic analysis based on Augmented Transition Network (ATN), which checks if the tokens' structure is in allowable grammatical structure. This is processed via the parser according to a Context-Free Grammar (CFG), which is used by our system.

```

SENTENCE= VERB_PHRASE + OBJECTS
SENTENCE=QUESTION + AUX_VERB + OBJECTS
SENTENCE=PRON + AUX_VERB +PREP+ VERB_PHRASE + OBJECTS
SENTENCE= VERB_PHRASE + OBJECTS +CONJ+ OBJECTS
SENTENCE= SENTENCE + (CONDITION) + (ORDER)
CONDITION= COND + OP + (AND CONDITION)
COND = WHERE | WHOSE | WHOM | HAVING
OP=NOUN_PHRASE + SYMBOL + VALUE+ (NOUN_PHRASE)
SYMBOL= IS | = | > | >= | < | <= | <>
VALUE = NUMERIC | STRING | DATE
ORDER = ORD + NOUN + (AND ORDER)
ORD= ORDER BY | SORTED BY | ACCORDING TO | ...
VERB_PHRASE=VERB+ (PRON)
OBJECTS= OBJECT + (AND OBJECTS)
OBJECT= (QUANT) + (PRON) + NOUN_PHRASE
NOUN_PHRASE= (DET) + ADJ_EXPR
NOUN_PHRASE= (DET) + NOUN + PREP + NOUN_PHRASE
NOUN_PHRASE= NOUN + CONDITION
PREP= OF | IN | AT | TO | ...
ADJ_EXPR= (ADJ) + NOUN | (ADJ) + NOUN + NOUN_PHRASE
NOUN= STUDENT | PATIENT | EMPLOYEE | DEPARTMENTS
QUANT = ALL | ANY | EVERY
PRON= ME | OUR | US | I |
AUX_VERB = IS | ARE |WANT
QUESTION = WHO | WHAT | WHERE | WHICH
ADJ = COLOR | STATUS |
STATUS = MARRIED | DIVORCED | SINGLE | GRADUATED | FAIL ...
COLOR = RED | BLUE | GREEN | ....
CONJ= AND | OR

```

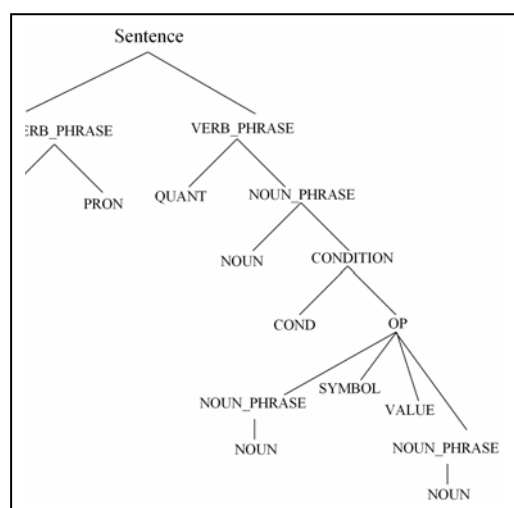
Figure-4.2: Sample of ATN.

The ATN technique is a network-like structure consists of labeled of nodes and

arcs. Every node represents different state of a process, and an arc represents the

transition from a state to another, with a label referring to word category in NLP. ATN is built on the idea of using finite state. Sentences are parsed by reaching a final state in any state graph. A sentence is determined to be grammatically correct if a final state is reached by the last word in the sentence.

- **Parse Tree** is a tree that represents the syntactic structure of a sentence according to some formal grammar approved by the ATN. A parse tree is



There is a big difference between the natural language (i.e. English) grammar and the semantic grammar of the NLI processes is in the meaning of the words. For example in the English grammar the word “employee” is a *noun*, but in semantic grammar it can be classified as a database table name, an attribute name, or a reference to a database tuple. Sometimes, a natural language phrase can be represented in our semantic grammar only by an attribute name or by a relational

composed of nodes and branches; each node is either a root node, a branch node, or a leaf node. Figure-4 depicts a sample of a parse tree. In a parse tree, an interior node is a phrase and is called a non-terminal of the grammar, while a leaf node is a word and is called a terminal of the grammar. Figure-5 depicts linguistic parse tree representing the natural language query "List me all employees having salary greater than or equal to 300 dinars".

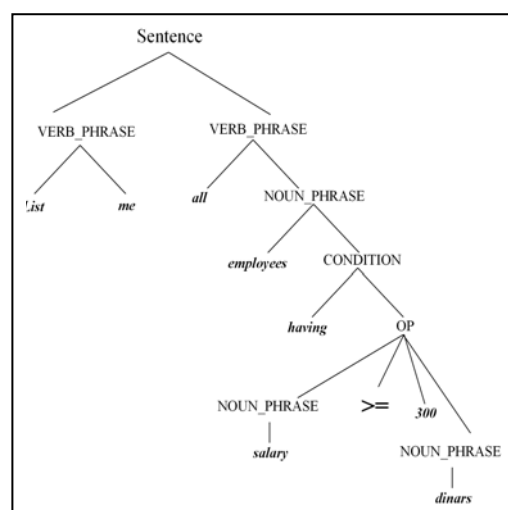


Figure-5: Linguistic parse tree

operator. Tables-2, table-3 and table-4 depicts some examples of semantic grammar representation.

The semantic grammars parser used in our GINLIDB system is designed to support a broad range of natural language statements. We used the EMPLOYEE³ database to create the semantic grammar English language queries. In GINLIDB system, there are

³ Almasri book Elmasri, R. and Navathe, S., (2007). Fundamentals of Database System. 5th ed. Addison Wesley, USA.

two types of semantic grammar, the first is a single lexicon semantic grammar, and the second is a composite lexicon semantic grammar. The single lexicon semantic grammar consists of individual words and some of their synonyms that are used in the

English language grammar. The composite lexicon semantic grammar is a combination of terminal words (terminals that exist only in the lexicon) that form phrases or sentences in a specific structure.

Table-2: Examples of semantic representation of terminal words.

Terminal words	Semantic
Employee worker	Employee
Employees workers	Employees
Salary income money	Salary
Customer patron member	Customer
Customers patrons members	Customers

Table-3: Semantic representation of terminal symbols.

Terminal	Semantic
greater larger bigger	Greater
Than	Than
Or	Or
Equal like equivalent =	equal
To near close	To

Table-4: Semantic representation of non-terminal examples.

Non-terminal	Semantic representation
greater than or equal to	>=
at least	>=
not less than	>=
at or above	>=
older than	>
less than or equal	<=
less than	<
Younger than	<

3.3 SQL constructing component

The SQL constructing component consists of three parts; SQL Generator, database adaptor, and SQL executor.

3.3.1 SQL generator

The task of the SQL generator is to map the elements of the natural query to the actual elements of the SQL the used databases. The SQL generator uses four routine, each of which manipulates only one specific part of the query. The overall SQL statement is constructed from the

concatenation of the output of the four routines. The first routine selects the part of the natural language query that corresponds to the appropriate DML command with the attributes' names (i.e. SELECT * clause). The second routine selects the part of the query that would be mapped to a table's name or a group of tables' names to construct the FROM clause. The third routine selects the part of the query that would be mapped to the WHERE clause (condition). The fourth routine selects the part of the natural language query that corresponds to the

order of displaying the result (ORDER BY clause with the name the).

3.3.2 Database adaptor

As there are many DBMS in existence that can be used. Each of them has its own techniques and interfaces that are different from each other. Our system can handle these varieties through database adaptor. Database connection, constraint, data type, and SQL format are examples of such varieties.

3.3.3 SQL executor

The purpose of SQL executor is to get the required results from the used database. In order to achieve this, the generated SQL statement would be tested to verify correctness before applied to the used database and then represent the result to the user. The testing phase involves the verification of the given valid name(s) of either table(s) or attribute(s) and also checks for meaningless queries.

3.4 Syntactical knowledge base

The syntactical knowledge base of the GINLIDB system is used by the linguistic component to determine the accepted words, provide word alternatives (in spelling correction process), and to verify the natural language query grammar.

3.5 Semantic knowledge base

This knowledge base consists of the English semantic grammar (grammar rules) and the schema of the database in use. The semantic knowledge base is used to replace words and/or phrases semantically by equivalent words and/or

phrases that are recognized by our system (according to the system capabilities).

3.6 Knowledge extension

This component extends the syntactical knowledge base by the adding new words and the semantic knowledge base by adding new rules. The importance of this component is to enlarge the system to accommodate variant domains and to strengthen the terminology and rules of existing domains.

4.0 Design of the GINLIDB system

The system is designed and implemented by the use of Object Oriented (OO) techniques. The Unified Modeling Language (UML) is an evolutionary general-purpose, broadly applicable, tool-supported, and industry-standardized modeling language [6], used to design our system. We have used number of diagrams from the UML and they are as follows:

- **Use case diagrams** are to conceptualize the functionality of the system through the systems' cases that represent different overall system scenarios.
- **Sequence diagrams** are used to show the interactions among different elements of the system in the shape of passing messages from and to each object.
- **Sequence diagrams** depict the internal behavior of the GINLIDB system.
- **Class diagram** is used to describe the static view of the system by describing the classes and relationships among them.

- **Activity diagrams** are used to capture the flow from one activity to the next.

From figure-6 to figure-10 depict samples of diagrams used in the system.

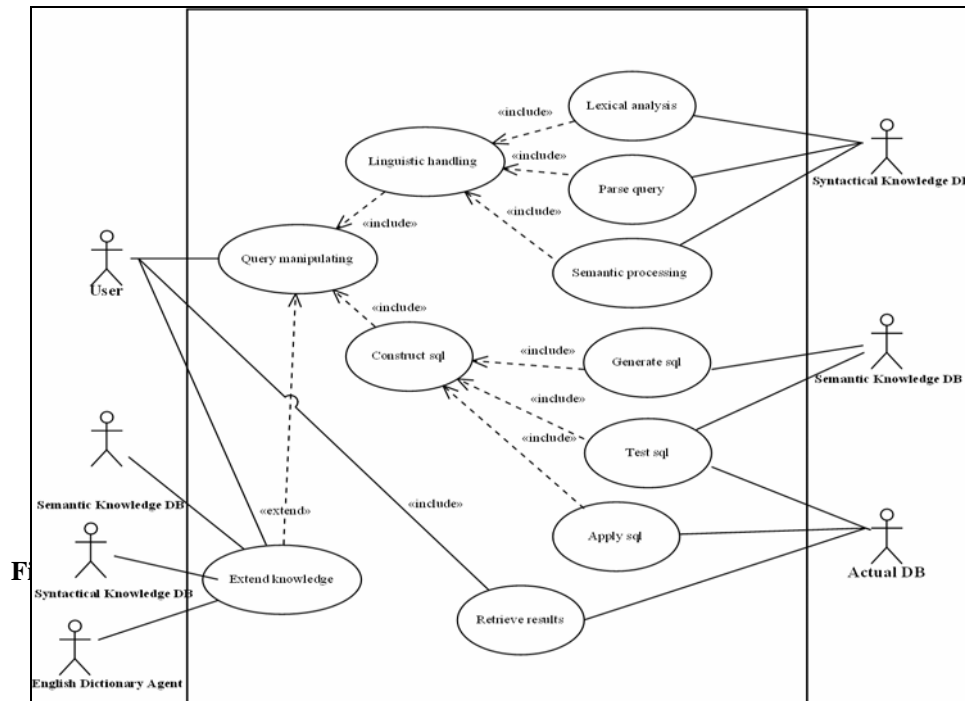
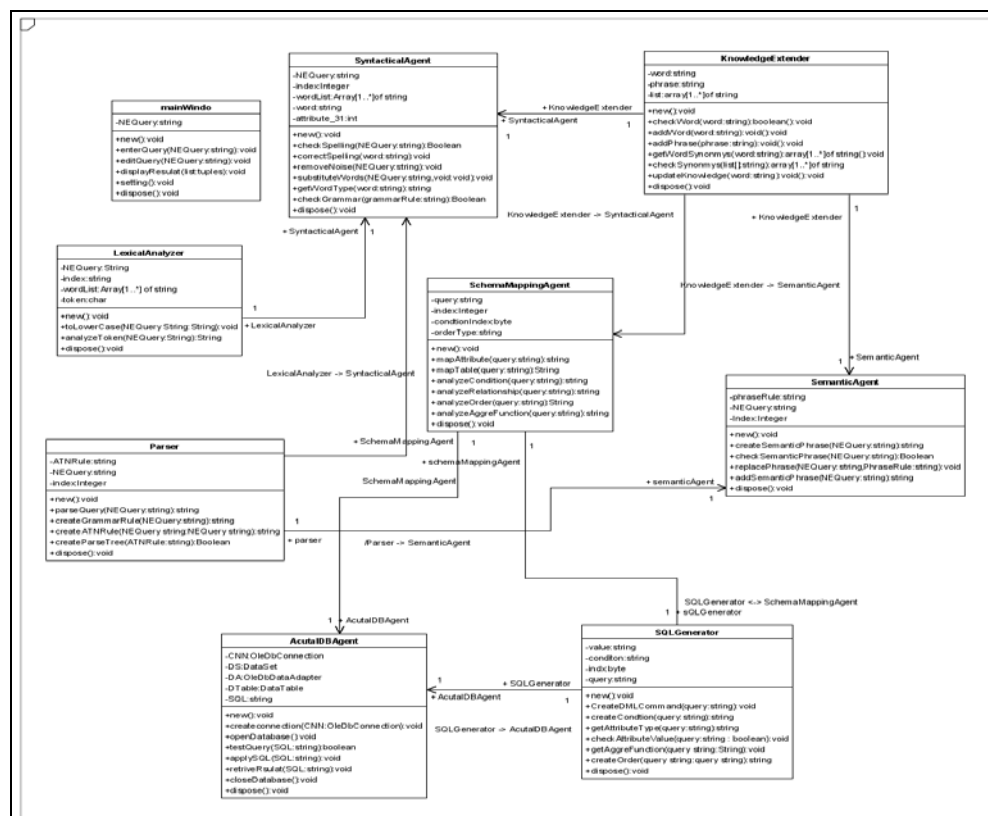


Figure-7: Class diagram of GINLDB system.



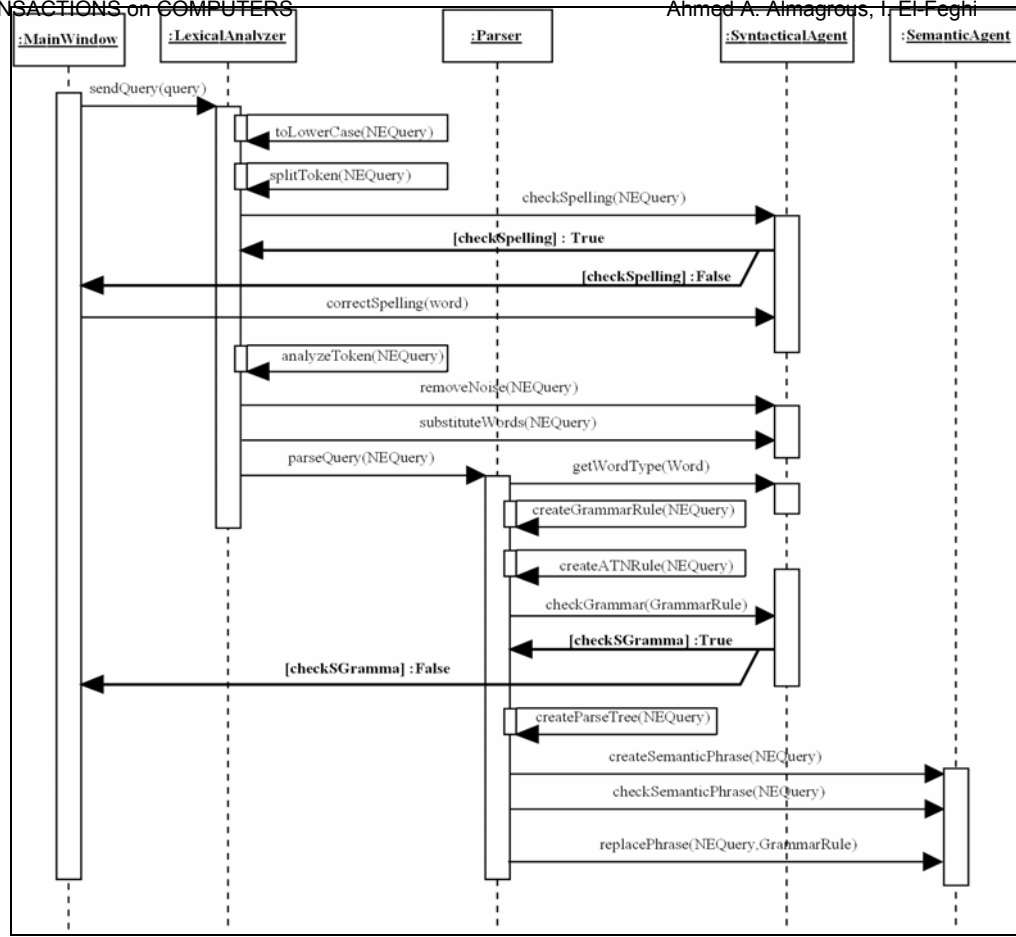


Figure-8a: Sequence of lexical analyzer.

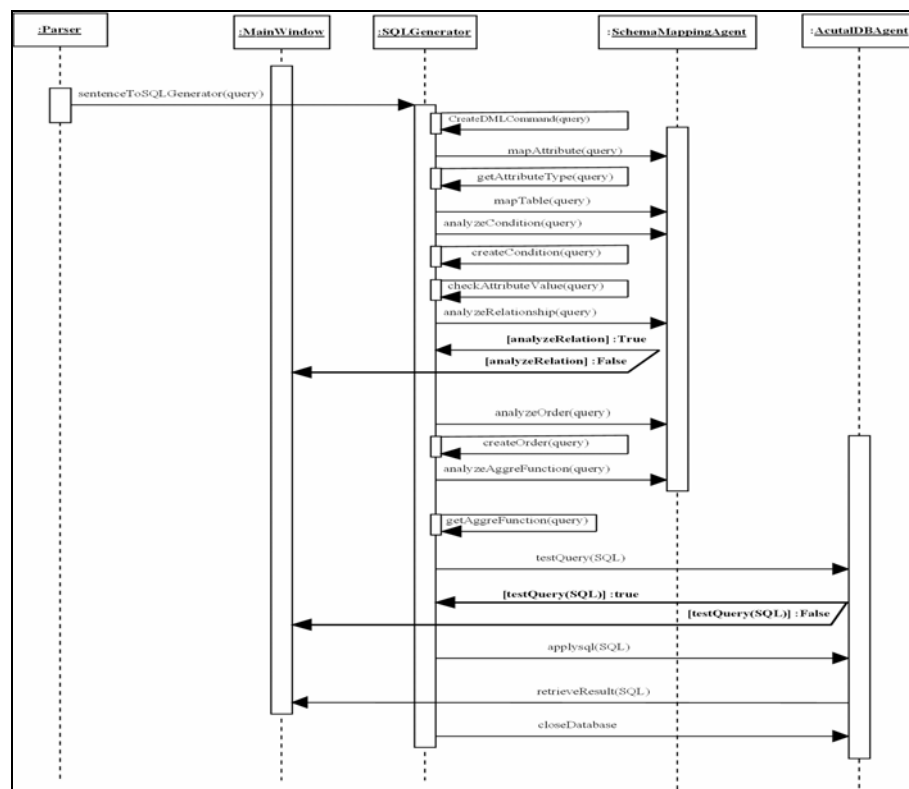


Figure-8b: Sequence diagram of parser.

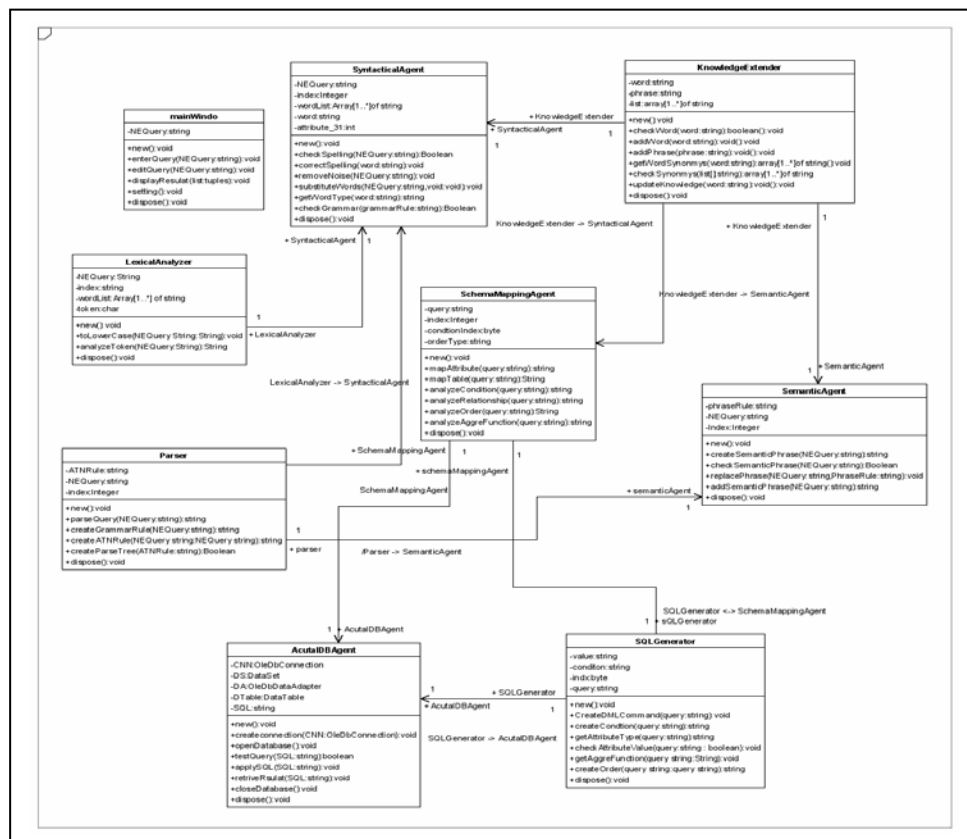


Figure-9: Class diagram of GINLIDB system.

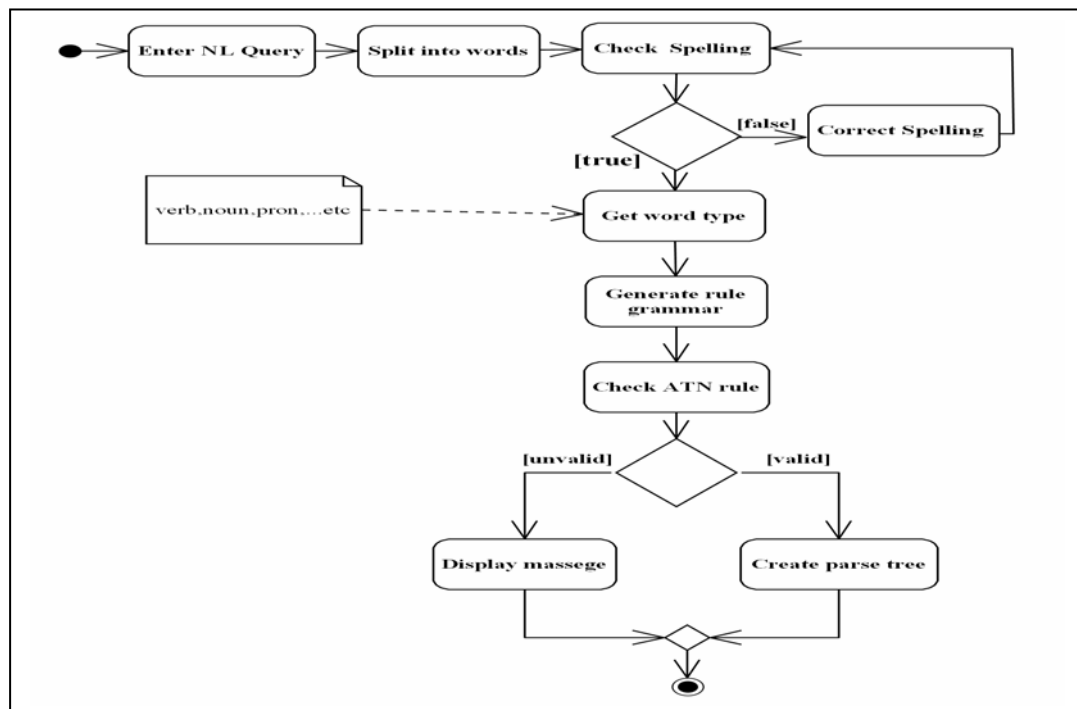


Figure-10: Activity diagram of parser.

4.2 Implementation of GINLIDB System

Here, we represent the different functionalities of the system.

- **Main window of GINLIDB system** is invoked through the GUI, where the user

provides the desired query. The system will notify the user with the correctness of the query in an interactive mode. The main window is depicted in figure-11.

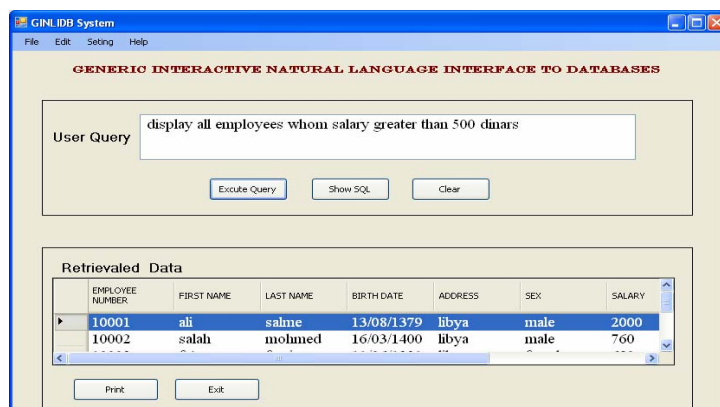


Figure-11: Main window of GINLIDB system.

- **Spelling checker** provides the user with the facility to correct the query. The

main window with the popup menu is depicted in figure-12.

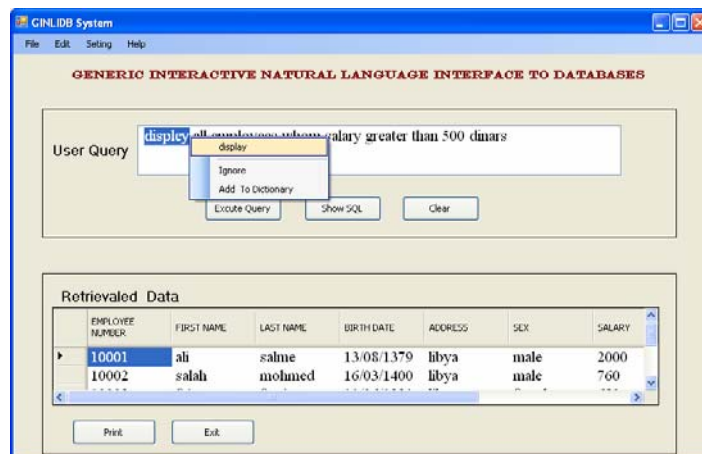


Figure-12: Spelling checker window

- **Extends knowledge** extends the knowledge base of the by adding New

Words to the existing knowledge base, as depicted in figure-13.

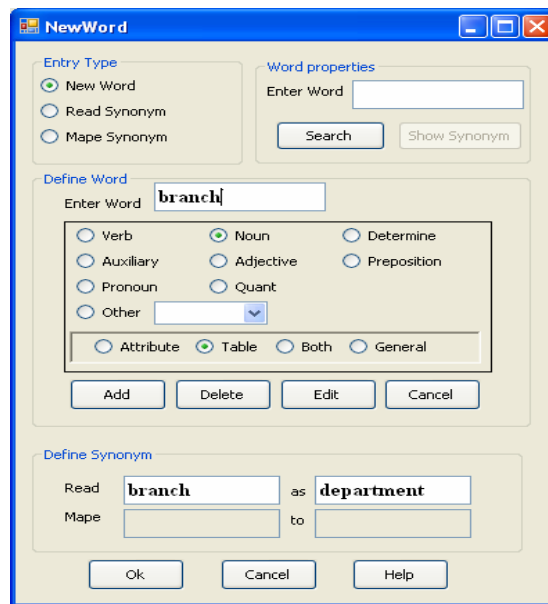


Figure-13: Extend knowledge window.

- **Database mapping** is to associate newly add words with synonyms words to be used in future queries as depicted in figure-14.

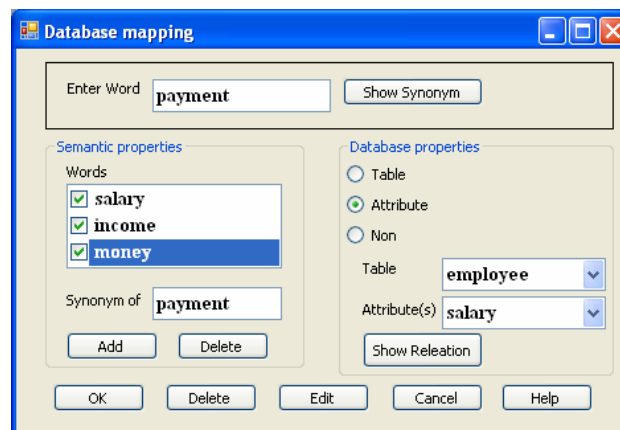


Figure-14: Database mapping window.

- **Database relationships handler** is to establish relationships between the tables of the used database as depicted in figure-15.

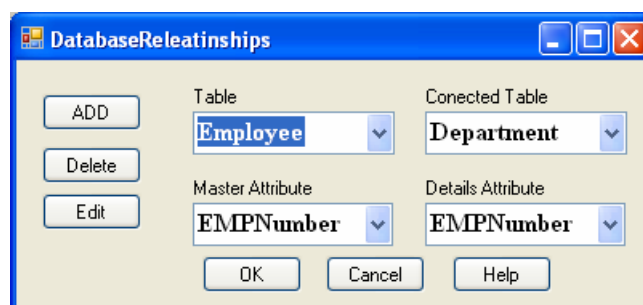


Figure-15: Database relationships handler window.

- **Tables' activation** is to activate the database tables that can be used as depicted in figure-16.

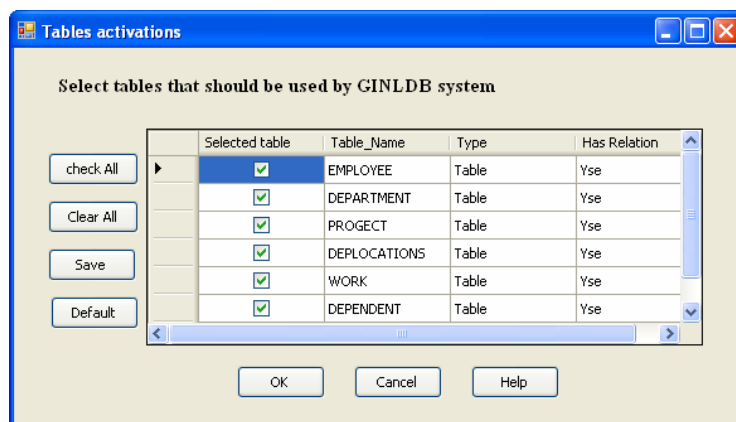


Figure-16: Tables activations window.

- **English grammar handler**

In the case of new queries that are unavailable in our ATN grammar then they can be added to our knowledge base.

This facility can be accessed through the Augmented Transition Network handler procedure as depicted in figure-17.

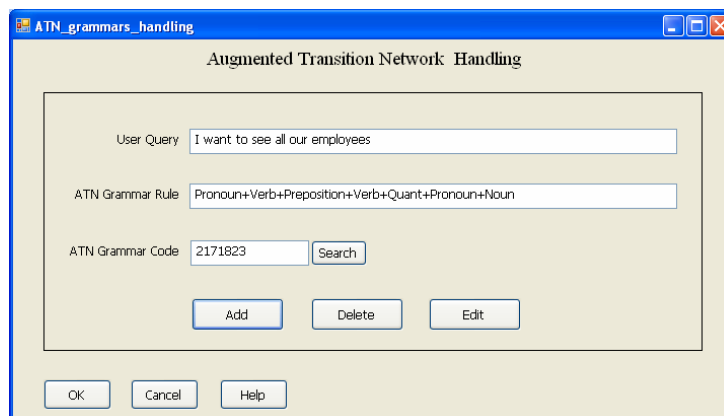


Figure-17: English grammar handling window.

5.0 Experiments and results

Here, we demonstrate the capabilities of our system via two experiments. The first experiment focuses on the natural language query syntactic correctness and the second deals with the SQL generation.

5.1 experiments

The first experiment had been conducted with the use of five different query verbs

in a sentence such as; **Show, Tell, Display, List and Give**. Each of the sentences has eight different variations to produce an ATN rules. So, the total of trials is 40. For example the eight variations that can accompany the verb **Show** are as follows:

Show employees
Show all employees
Show me all employees
Show all our employees

Show me employees
Show our employees
Show me our employees
Show me all our employees

All of the above variations give the same answer. In this manner, the user is not confined to a limited set of words that can be handled. The limitation depends on the size and content of the systems' knowledge base. When the query's ATN rule is not available in our systems' knowledge base then it will be rejected and the user has to rephrase it or the user can force the system to generate the ATN for the query and add it to the knowledge base.

The second experiment is related to the generation of the SQL statement from the English natural query. In what follows, we demonstrate our system capabilities in generating the different form of the SQL statements depending on the structure of the English natural queries.

A general query simple is a query where there are no specifics for the **attributes list**, **conditions**, **relationship**, etc...

Table-5 depicts some examples.

User query	Generated SQL
show me all employees	<i>SELECT * FROM</i> employee
tell me all our employees	<i>SELECT * FROM</i> employee
display all projects	<i>SELECT * FROM</i> projects
list all our departments	<i>SELECT * FROM</i> departments
who are our employee?	<i>SELECT * FROM</i> employee
what is our projects ?	<i>SELECT * FROM</i> projects

Table-5: Simple queries.

A specific query is a query with some certain attributes. So, the selection of the attributes is distinct to certain tables.

Table-6 depicts some examples of SQL with the DISTINCT clause.

User query	Generated SQL
tell me our employee location	None
display employee address	<i>SELECT DISTINCT</i> employee.address <i>FROM</i> employee
list all our departments locations	<i>SELECT DISTINCT</i> departments.location <i>FROM</i> departments
what are our employee name?	<i>SELECT DISTINCT</i> employee.name <i>FROM</i> employee
what are the locations of our departments ?	<i>SELECT DISTINCT</i> departments.location <i>FROM</i> departments
what are the departments ?	None
tell me our employees names and addresses	<i>SELECT DISTINCT</i> employee.name, employee .address <i>FROM</i> employee
display employees addresses and names	<i>SELECT DISTINCT</i> employee.address, employee .name <i>FROM</i> employee

Table-6: Examples of distinct attributes.

A conditioned query is a query that will select some certain tuples of the database giving some specific criteria. Table-7 depicts some examples of SQL statement with WHERE clause.

User query	Generated SQL
show me all employees whose names are "ahmad"	<i>SELECT * FROM</i> employee <i>WHERE</i> employee.name= 'ahmad'
tell me our employees whose names starts with a	<i>SELECT * FROM</i> employee <i>WHERE</i> employee.name LIKE 'a%'
display all employees whom salary between 100 and 200	<i>SELECT * FROM</i> employee <i>WHERE</i> employee.salary >= 100 AND employee.salary <= 200
list me all employees whom are younger than 38 years?	N/A
List me all employees whom birthday less than "1-1-1970"	<i>SELECT * FROM</i> employee <i>WHERE</i> employee.bdate < #1-1-1970#

Table-7: Examples with WHERE clause.

Some times the user would like to inquire about some certain attributes that satisfy some given condition(s). Table-8 depicts some examples of SQL statements with specific attributes and conditions.

User query	Generated SQL
show me names of our employees whose names are "ahmad"	<i>SELECT DISTINCT name FROM</i> employee <i>WHERE</i> employee.name = "ahmad"
what are our employee address whose name is "ali"	<i>SELECT DISTINCT</i> employee.address <i>FROM</i> employee <i>WHERE</i> employee.name = "ali"
list names and salaries of all female employees	N/A
List me names and salaries of all employees whom sex are female	<i>SELECT DISTINCT</i> employee.name , employee.salary <i>FROM</i> employee <i>WHERE</i> employee.sex = 1

Table-8: Examples with WHERE and DISTINCT clauses.

Average, Sum, etc... are used for that purpose. Table-9 depicts some examples of SQL statements on the form of:

In some case, the user is interested in the summarization of the data of numerical data. So, functions such as Count,

$$SELECT \left\{ \begin{array}{l} COUNT (*) \\ SUM (attribute) \\ AVG (attribute) \\ \dots \end{array} \right\} AS mem_var FROM Table$$

Our system can also deal with queries that need data from more than one relational database. In such case the system has the capabilities to perform the appropriate

joins to retrieve the required data. The system can retrieve the required data and display it in desired order such ascending or descending.

User query	Generated SQL
How many employees are there?	<i>SELECT COUNT(*) AS result_count FROM employee</i>
Show me the projects count	<i>SELECT COUNT(*) AS result_count FROM project</i>
How many departments do we have?	<i>SELECT COUNT(*) AS result_count FROM department</i>
How many male employees are there?	<i>SELECT COUNT(*) AS result_count FROM employee WHERE employee.sex=1</i>
Tell me how many employees have the name of "ali"	<i>SELECT COUNT(*) AS result_count FROM employee WHERE employee.name = "ali"</i>
how many employees names starts with a	<i>SELECT COUNT(*) AS result_count FROM employee WHERE employee.name LIKE "a%"</i>
how many projects are in department number 5	<i>SELECT COUNT(*) AS result_count FROM project WHERE employee.dnum=5</i>
show me the total salary of our employees	<i>SELECT SUM(salary) AS result_total FROM employee</i>
Show the total work hours in project number 3	<i>SELECT SUM(hours) AS result_total FROM work WHERE work.PNO=3</i>
show me the average salary of our employees	<i>SELECT AVG(salary) AS result_ average FROM employee</i>
what is the average salary of the employees in department number 5	<i>SELECT AVG(salary) AS result_ average FROM employee WHERE employee.DNO=5</i>
What is the average work hours in project number 3	<i>SELECT AVG(hours) AS result_ average FROM work WHERE employee.PNO=3</i>

Table-9: Examples with aggregate functions.

5.2 Results

Many experiment in a trial like had been conducted on our system. The trials had given accurate and satisfactory results where the generated SQL statements had been run against the used database. In all of our trials, we have used the Employee database.

6.0 Conclusion and further research

Our system accepts an English language requests that is interpreted and translated into SQL command using semantic-grammar technique. In addition, our system requires a knowledge base that consists of a database and its schema. The design and implementation of the system had carried out with three major concepts in mind that vital to any NLP system. The three major concepts are:

1. The users' submitted query in natural English language is analyzed from the syntactic as

well as semantic merits so the query will be correct and can be answered efficiently with respect to systems' knowledge base.

2. The construction of a valid SQL statement that represent the users' query.
3. The retrieval of the result that is required by the users' query.

The result of the number of experiments in the form of trials in a user friendly environment had been very successful and satisfactory.

We would like conclude this work by the demonstration of the capabilities and advantages of the system in the following points:

1. Our system is capable to answer common queries give the appropriate database and knowledge base.
2. Our system as any other NLP system needs knowledge base that

is tailor made for a given particular database.

3. The limitation of most of previously known NLP systems is to deal only with a limited domain and only small set of queries can be answered.
4. Our system is capable to extend its' knowledge base to cover more queries for the same database.

The researcher would like to make the following points for future work and research:

1. To put our system to vigorous test by running the system against different database to evaluate its' performance.
2. Automatic generation of the systems' knowledge base.
3. To build Arabic language front-end preprocess so the system will work in Arabic.

6.0 References

- 1 Abrahams P. W. et al. (1966). The LISP 2 Programming Language and System. Proceedings of FJCC, No. 29. USA, Pages 661– 676.
- 2 Amble, T. (2000). BusTUC - A Natural Language Bus Route Oracle. 6th Applied Natural Language Processing Conference, Seattle, Washington, USA.
- 3 Arun, A., Keller, F. (2005). Lexicalization in Cross linguistic Probabilistic Parsing. Proceedings of the 43rd Annual Meeting of the ACL, pages 306 – 313.
- 4 Elmasri, R. and Navathe, S., (2007). Fundamentals of Database System. 5th ed. Addison Wesley, USA.
- 5 Grosz, B., Joshi, A., Weinstein, S. (1983). Providing a unified account of definite noun phrases in discourse. In Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics, Cambridge, MA USA, Pages 44 – 50.
- 6 Hamilton, K., Miles, R. (2006). Learning UML 2.0. O'Reilly, ISBN-10: 0-596-00982-8
- 7 Hendrix, G. (1977). The LIFER manual A guide to building practical natural language interfaces. SRI Artificial Intelligence Center, Menlo Park, Calif. Tech. Note 138.
- 8 Hendrix, G., Sacrdoti, E., Sagalowicz, D. and Slocum, J. (1978). Developing a natural language interface to complex data. ACM Transactions on Database Systems, Volume 3, No. 2, USA, Pages 105 – 147.
- 9 Luger, G., Stubblefield, W. (1999). Artificial Intelligence Structures and Strategies for Complex Problem Solving. 3rd ed. Addison-Wesley, USA.
- 10 McCarthy, J. (1959). LISP Programmers Manual, Handwritten Draft. MIT AI Lab., Vambridge, USA.
- 11 Nogami, H., Yoshimura, Y. and Amano, S. (1989). Parsing with look-ahead in real-time on-line translation system. Research and Development Center Toshiba Corporation Kawasaki-City, Japan Volume 1, pages 488 – 493.
- 12 Palmer, M., Finin, T. (1990). Workshop on the Evaluation of Natural Language Processing Systems. Computational Linguistics, MIT Press, Volume 16, pages 175 – 181.
- 13 Sinan S. (2002). Guide To applying The UML. Springer-Verlag New York, Inc, ISBN 0-387-95209-8.
- 14 Stratica, N., Kosseim, L. and Desai, B. (2002). A natural language processor for querying CINDI. Volume

- Milutinovic ed., Proceedings of SSGRR 2002, International Conference on Advances in Infrastructure for e-Business, L'Aquila, Italy.
- 15 Tang, L. R., & Mooney, R. J. (2000). Automated construction of database interfaces: Integrating statistical and relational learning for semantic parsing. In Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-2000), pages 133 – 141, Hong Kong.
 - 16 Warren Teitelman, PILOT, (1966). A Step toward Man-Computer Symbiosis. Ph.D. Thesis, Massachusetts Institut of Technologie. Project on Mathematics and Computation (MAC). Technical Report MAC-TR-32, Cambridge, MA, September 1966.
 - 17 Warren, D., Pereira, F. (1982). An efficient and easily adaptable system for interpreting natural language queries in Computational Linguistics. Volume 8 pages 3 – 4.
 - 18 Woods, W. (1973). An experimental parsing system for transition network grammars. In Natural Language Processing, R. Rustin, Ed., Algorithmic Press, New York.
 - 19 Woods, W., Kaplan, R. and Webber, B. (1972). The Lunar Sciences Natural Language Information System. Bolt Beranek and Newman Inc., Cambridge, Massachusetts Final Report. B. B. N. Report No 2378.