INTELLIGENT MULTIDIMENSIONAL DATABASE INTERFACE

Mona Gharib Mohamed Reda Zahraa E. Mohamed

Faculty of Science, Mathematics Department, Zagazig University

Zagazig, Egypt

ABSTRACT
In the present computing world, most new-generation database applications need for intelligent interface to enhance efficient interactions between database and the users. Database query language SQL could be difficult to the non-expert users and learning these formal queries takes a lot of time. In this paper, we discussed mapping of natural language queries to SQL rather than building normal query and a user simply poses the question in everyday verbiage language. Furthermore, we introduce a dynamic approach to determine the tables and attributes involved in the query by using database metadata schema. Our approach will minimize the time that used to build the queries thus minimizing the code size and effort for building query. We also propose a dynamic component that helps in building any NLIDB this component is linked to all the NLIDB system's components and helps to build the dictionaries and database schema graph.

KEYWORD
Intelligent Multidimensional Database Interface (IMDI), Natural Language Interface to Databases (NLIDB), Database Management System (DBMS), Structured Query Language (SQL).

INTRODUCTION

Natural Language Interface to Database (NLIDB) systems have provided an easy access to database system without the need for the user to use formal query languages [1], such as SQL. Database query languages can be difficult to the non-expert users and learning these formal queries takes a lot of time. In NLIDB, users can type a question or a sentence in their natural language. Then it will be converted through a special natural language interface interrupter into formal database query. A major problem that faces the NLIDB designer is the identification of the tables that contain the required information and the desired attributes in query. In [2], previous work used static built-in templates of possible production rules for the possibly introduced queries. In which tables' names are embedded in the template. However, this requires large code that considers all possible query templates. The standard approach to database NLP systems relies on creating a 'semantic grammar' for each database, and uses it to parse the NL questions [2]. The semantic grammar creates a representation of the semantics of a sentence. After some analysis of the semantic representation a database query can be generated to SQL. In the existing systems, they are some problems such as: the requirement of using additional knowledge to extract meaningful information, the input can have many choices and it is not easy to choose the correct choice among target representations (one-to-many mappings), the complexity of mapping in NL sentences if you change a single word, the entire structure can be changed, which is called the quantifier scoping problem. Words such as "the," "each," or "what" can have several meanings in different situations. Another problem is the identification of tables required to build the query. In this work, we will solve some these problems by using semantic analysis, lexical dictionary, interactive query formulation and syntax-based system.

In this work, we provide user friendly query interface for non expert users and this interface will have a graphical user interface; user should have knowledge about data source structure and contents which are essential to the construction of intuitive interface. The transformation of natural language interface to database divide into two steps: Query interpretation and Query translation.

Query interpretation: a natural language query enters by the user is parsed into a logical representation.

Query translation: the logical representation of a query is mapped to a database querying language.

It is evident that query interpretation process requires both extensive linguistic resources for understanding query, whilst the query translation step requires semantic resources for mapping query terms to database entities.

BACKGROUND AND RELATED WORK

NLIDBs permits users to formulate queries in natural language, providing access to information without requiring knowledge of programming or database query languages. A general information management system that is capable of managing several kinds of data, stored in the database is known as Database Management System (DBMS).In the real world we obtain information by asking questions in a natural language, such as English. Supporting arbitrary natural language queries is regarded by many as the ultimate goal for a database query interface when it comes to using a multidimensional database for internal business purposes; the main advantage is the ease of obtaining data quickly and succinctly. Any question that contains a request for information that is not found within the database will not

result in a direct response. DATABASE systems are being used in almost all aspects of our lives, in banks, universities, hospitals. Etc. To manage databases; users have to learn a formal query language. Formal query languages are difficult to learn and Master; at least by non-computer specialists. Casual users need an application interface through which they can manage the database. There are several major reasons why Understanding is a difficult problem [1]. These problems include: the requirement of using additional knowledge to extract meaningful information, the input can have many choices and it is not easy to choose the correct choice

Among target representations (one-to-many mappings), the complexity of mapping because in NL sentences if you

Change a single word, the entire structure can be changed, which is called the quantifier scoping problem. Words

Such as "the," "each," or "what" can have several meanings in different situations [1, 12, 13]. Another problem is the Identification of tables required to build the query. Previous work used static built-in templates of possible production Rules for the possibly introduced queries. This will require the careful analysis of the user requests and huge code for Transformation of these requests into queries. There are many applications that can take advantages of NLIDB [2]. In PDA and cell phone environments, the display screen is not as wide as a computer or a laptop. Filling a form that has many fields can be tedious: one may have to navigate through the screen, to scroll, to look up the scroll box values, etc. Instead, with NLIDB, the only work that needs to be done is to type the question similar to the SMS (Short Messaging System).The NLIDB 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 a friendly environment to interact with users in the sense that computer does not require any programming language skills to access the data; only natural language is required. The research of Natural Language interface to databases (NLIDB) has recently received attention from the research communities. The area of NLIDB research is still very experimental and systems so far have been limited to small domains, where only certain types of statements can be used. When the systems are scaled up to cover larger domains, it becomes difficult due to vast amount of information that needs to be incorporated in order to parse statements. Although the earliest research has started since the late sixties, NLIDB remains an open search problem.

Intelligent system overview

To translate a natural language query into SQL Query expression, we first need to identify words/phrases in user question which will be mapped into the corresponding SQL query. Unclassified terms can't be mapped in our system. We will use query interpretation to fill both semantic dictionary from user question and lexical dictionary from database metadata. We will define query tables and attributes by using semantic and lexical dictionaries. Then, we will use query generator to map data from semantic dictionary with data from lexical dictionary then test if we have required data to build SQL query component (Tables, Attributes, Conditions) then execute query on database and retrieve data to user.

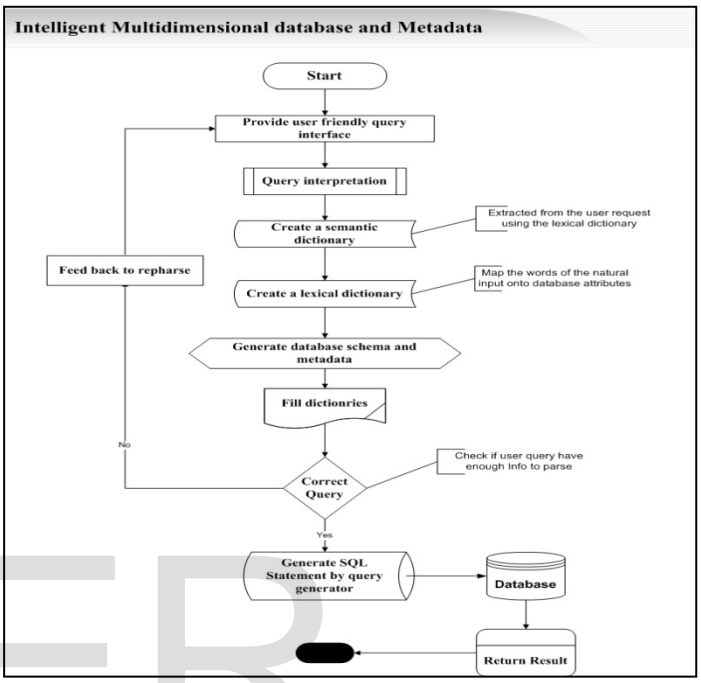

Fig (1): Intelligent database interface.

Intelligent System Components

Our system consists of four main components: User Friendly Interface (UFI) to provide user with friendly graphical user interface, Query Interpretation (QI) used to fill semantic dictionary from user question and fill lexical dictionary from database using Metadata, Query Generator (QG) used to map semantic with lexical dictionary and generate SQL statement and system outputs (RV) used to view returned result from database.

User Friendly Interface

We will provide user graphical friendly interface to facilitate user query by writing question in natural language with some user guides to ensure that we have enough data to generate database query

Query Interpretation

After receiving user question we have ability to check if user poses this question before to get SQL query that are saved in our system to increase performance by providing a method to save question if user poses new question we will go further to generate corresponding SQL query by using our approach.

Semantic analysis

Once the words are extracted from the user request using the semantic analysis, they are mapped to the database

attributes. This implies that the meaning of each word needs to be defined. Each word in user question may have many Synonyms words that could be used in user query each word in semantic dictionary may have many Synonyms words. We will semantic dictionary words for mapping with lexical dictionary words. This process transforms user question from normal language SQL query. Both the lexical analysis and the semantic analysis are domain dependent modules. Many NLIDB systems are based on syntactic grammar or semantic grammar like in. This approach creates a 'semantic grammar' for each database. The semantic grammar creates a representation of the semantics of a sentence. For example "name" word may have many synonyms (names, nam and Names).

Lexical analysis

This dictionary is used to store the most common words expected to be used in the user request for this specific database system which makes this dictionary domain specific. These words consist of all the elements of the database (tables, attributes, relations, and valued attributes), which we called root words, and their possible synonyms along with some misspelling words. This step uses a lexical dictionary which holds the definition of all the words that may occur in the user request. We will fill lexical dictionary data from database using Metadata schema that are generated from our system database, each word in semantic dictionary should have one word in lexical dictionary, then we define word type in lexical dictionary, every word should have one of these types (Tables Names, Attributes, Condition). Keywords are defined to be words or phrases that have particular meaning within the domain of work. These words consist of all the elements (relations, attributes and values) of the database and their synonyms. For example, if we have a database that has the attributes name and age of students, the lexical dictionary will contain the keywords found in the database like: "name: name, names, named, student: students, pupils, pupil, age: age, ages, old, more: more, larger, much, and numbers like 20, 22, 21, . . ." where the bold words are the keywords substituted instead of its following words. A request such as: "find the names of students aged more than 21" when lexically analyzed will be compared to the lexical dictionary and will be converted to: "name, student, age, more, 21".The idea behind adding the misspelling words to the dictionary is to minimize the rejection rate of the user request. The user request can be rejected if it does not contain meaningful parts that could be used to build the query. If the system can determine every keyword and their corresponding meaning from the user request, then the sentence can then be converted easily to an appropriate query.

Query Generator (QG)

Our approach to generate SQL query will use templates: Select Attributes from Table Name or Select Attributes from Table Name Where Condition or Select Attributes from Table Name Where Conditions. First template will define Attributes and Tables Name, Second Template will define Attributes, Tables Name and one condition, Third will define Attributes and Tables Name and more than one condition.

Query Formulation

The mapping process from natural language to SQL Query requires our system to be able to map words from semantic dictionary with related words from lexical dictionary. Each word in semantic dictionary should have one word in lexical dictionary then define word type in lexical dictionary (Tables, Attributes, Conditions) Due to the limited vocabulary understood by the system; certain terms cannot be properly classified. Clever Natural language understanding systems attempt to apply reasoning to interpret these terms, with partial success. To ensure that this process proceeds smoothly for the user, we provide the user with specific feedback on how to rephrase.

Syntax-Based Systems

In syntax-based systems the users question is parsed (i.e. analyzed syntactically) and the resulting parse tree is directly mapped to an expression in some database query language. Syntax-based systems use a grammar that describes the possible syntactic structures of the users' questions. Syntax based NLIDBs usually interface to application-specific database systems that provide database query languages carefully designed to facilitate the mapping from the parse tree to the database query.

System Process

The two dictionaries are built empty of data, and are filled/updated through the database management component. The database metadata schema graph is also built through the database management component during the building of the database tables. Extract the database schema graph from it, and filling/updating the two dictionaries used. Finally, the result from this preprocessing step are a set of tokens of words after the removal of the un-wanted characters (like,*, &, .etc.). For using more than on condition we will use Query logical and numeric operators the role of this function is to return a string that contains the condition part in the query. The valued attributes found in the request represent the condition. This function takes the processed request, identifies the valued attributes, matches each with the proper attribute then returns a string that contains the condition statement used in the query. This function can deal with both simple and multiple conditions on different attributes. Logical operators (And, Or, Not) numeric operators (Greater or more than, Less than, Equal)
Greater or more than converted to (>=).
Less than converted to (<=).
Equal converted to (=).
Not Equal converted to (=).

Query Attributes built in SQL Functions

We will check if user question has SQL built in functions by using semantic dictionary data if there exists we will get corresponding function name to determine which function category belong to in our system we will cover some categories such Mathematical and statistics function Such (Count, Sum, Max, Min, Avg, Mean, etc), we will search for some words in user question should be mapped into semantic dictionary with corresponding word in lexical dictionary for example how many in semantic dictionary will be converted into Count in lexical dictionary, we have already predefined data in semantic and lexical dictionaries.

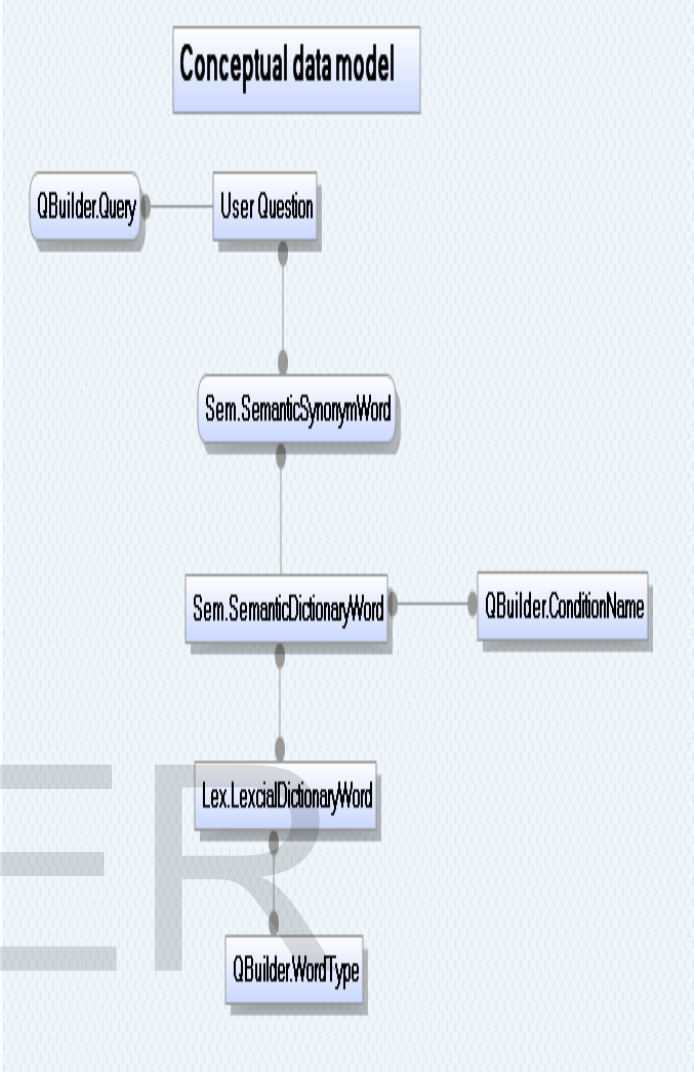

Fig (2): Intelligent system - Conceptual Model

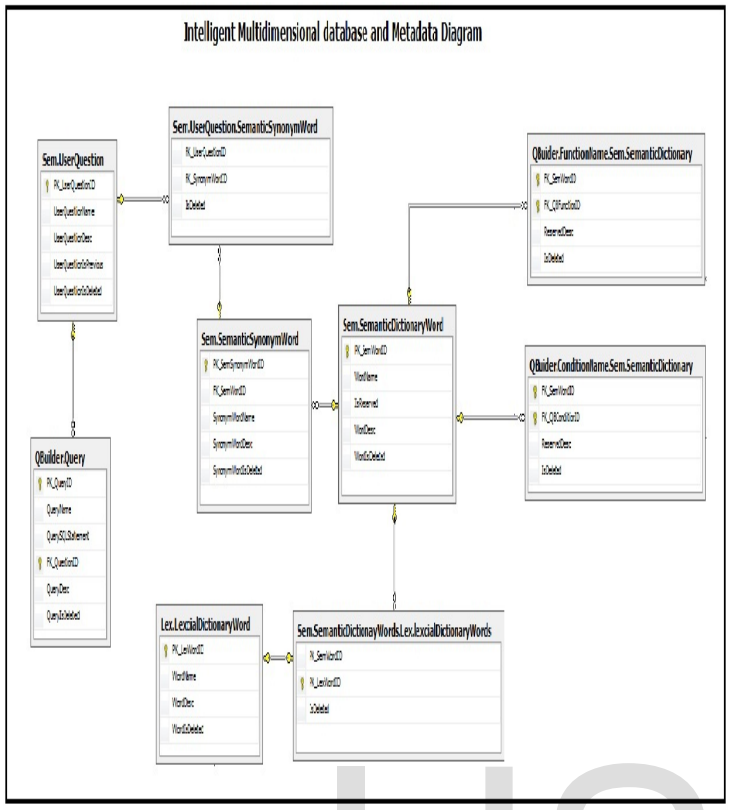

Fig (3): Intelligent system Database Diagram

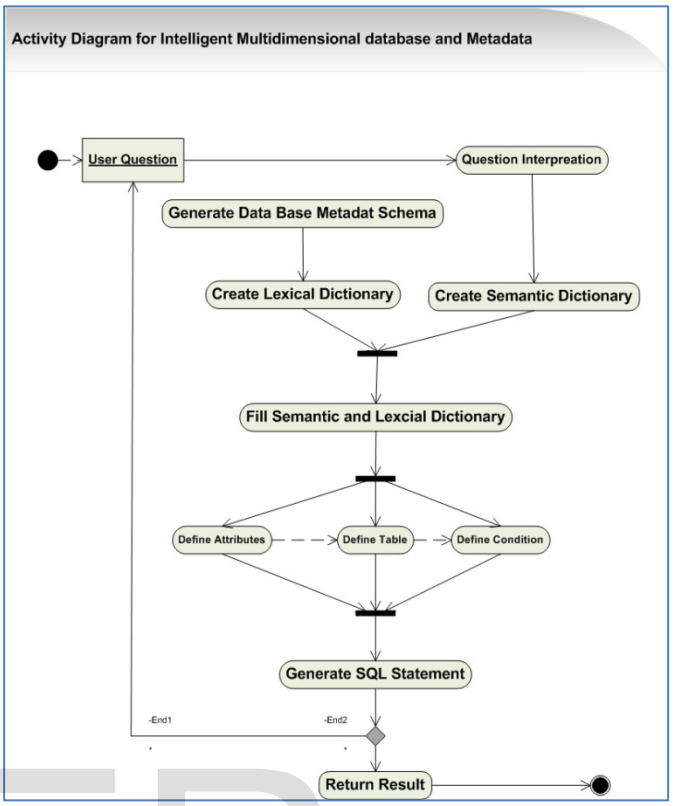

Fig (4): Intelligent system activity diagram

Intelligent system design

We will have some system artifacts to describe our system approach in different forms such

Conceptual data model

This is the highest level ER model in that it contains the least granular detail but establishes the overall scope of what is to be included within the model set. The conceptual ER model normally defines master reference data entities that are commonly used by the organization. Developing an enterprise-wide conceptual ER model is useful to support documenting the data architecture for an organization.

Our conceptual model will be used to describe our approach for building SQL query first user poses a question, we will save user question and corresponding query to return query if user ask for this question again to increase system performance by calling corresponding query. Extract words from user question and check for synonym words then fill semantic dictionary, check if question have condition and define which template should be used then fill lexical dictionary from database and define word types in lexical dictionary.

Physical model

Our system physical model will be used to describe database tables, attributes and relationships user question will have many to many relationship with synonym words, each word in semantic dictionary have many words in synonym word one too many relationship, each word in lexical dictionary have one corresponding word in semantic dictionary one to one relationship and user question could have one or many condition one too many relationship.

Activity Diagram

Our system will provide user friendly interface for user question then using query interpretation to fill semantic dictionary from user request and lexical dictionary from database schema them mapping two dictionaries to generate SQL query by using query generator after create query we will check if we have enough data if not return feedback to user if we have correct query execute on database.

Screen Shots for Intelligent system interface

Intelligent system interface is Graphical user friendly interface help user to enter question, compile question to

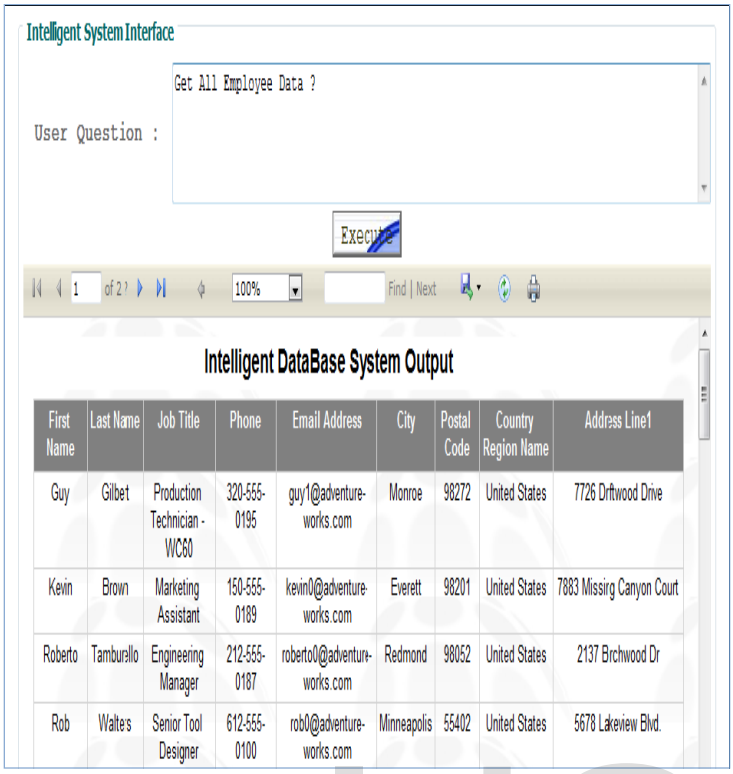

Fig (5): Intelligent Interface – User Question.

SQL query and execute query in our system database using DBMS then return result from database to view to user. Intelligent system interface for previous questions are used to view all previous question users have entered before and we have saved complied query in our system to increase performance and return result in less time from go on steps to generate new SQL query, User have ability to choose question then press Execute Question to return data and view to user.

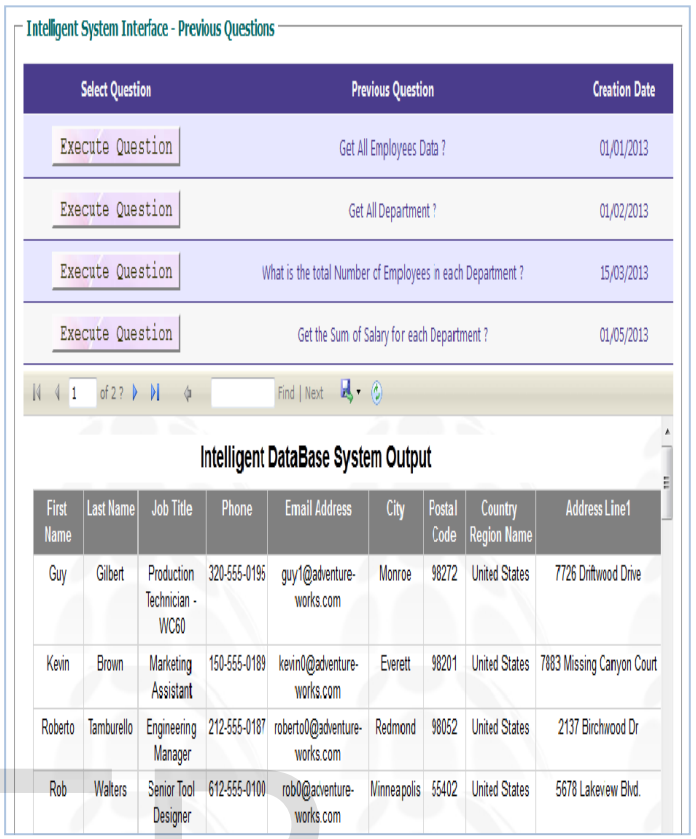

Fig (6): Intelligent Interface – Previous Questions.

CONCLUSION

In this paper, we introduced a new approach for generating SQL query using multidimensional database interface for non expert user by posing a question then converting question to SQL query. The main advantage of our system is using natural language for writing question, ability to save questions and mapped query for each question in database to increase system performance and stability. We used dynamic template to generate database query using query interpretation, semantic and lexical analysis to fill semantic and lexical dictionaries. Then get required data to build SQL query depending on our templates. We have used a dynamic approach to generate query whatever database we used by using metadata to fill lexical dictionary and have ability to communicate with different database providers by depending on standard SQL query language. And this approach will minimize the time and effort that used to build the queries.

REFERENCES

[1] Amany Sarhan. "A proposed architecture for dynamically built NLIDB systems". Knowledge-based and Intelligent Engineering Systems Journal, 13 (2), IOS Press Amsterdam, Netherlands, 2009.

[2] Zongmin Ma, "Intelligent Databases: Technologies and Applications", IGI publishing, 320 pages, 2007.

[3] Burlesan, Donald K., Joe Celko, John Paul Cook, and Peter Gulutzan. 2003. Advanced SQL Database Programmer Handbook.BMC Software and DBAZine.

[4] I. Androutsopoulos, G. Ritchie and P.Thanisch. "Time, tense and aspect in natural language database interfaces". Natural Language Engineering Journal, 4, 3, Cambridge University Press New York, 1998.

[5] C. Hallett, Generic Querying of Relational Databases using Natural Language Generation Techniques, Proceedings of the Fourth International Natural Language Generation Conference, pages 95-102, 2006.

[6] Dietmar Wolfram, "Applications of SQL for Informetric Data Processing", Proceedings of the 33rd conference of the Canadian Association for Information Science, 2005.

[7] Androutsopoulos, I., Ritchie, G. and Thanisch, P. "Natural Language Interfaces to Databases - An Introduction". Natural Language Engineering, 1(1): 29-81, 1995.

[8] Donald P. McKay and Timothy W. Finin, "The Intelligent Database Interface: Integrating AI and Database systems", In Proceedings of the 1990 National Conference on Artificial Intelligence: 677-684, 1990.