Natural Language Interface to Database Approach in the Task of Relational Databases Design

Kateryna Yalova[a] , Kseniia Yashyna[a] and Abdel-Badeeh M. Salem[b]

[a] *Dniprovsky State Technical University, Dniprobydivska str.2, Kamyanske, 51918, Ukraine*
[b] *Ain Shams University, El-Khalyfa El-Mamoun Street Abbasya, Cairo , Egypt*

Abstract

Developing general-purpose solutions that could be migrated to a variety of databases, companies, and data domains has been a big challenge for researchers for years. Today, most of the data is stored in relational databases, and extracting useful information from these databases requires special knowledge of the structured query language. Recent advances in natural language understanding and processing have led to renewed interest in natural language interfaces to database (NLIDB) approach, which provides a simple mechanism for non-technical users to access data from data storages. That is why NLIDB have attracted attention as useful tools that simplify interaction between users and computer systems. The paper describes the results of the NLIDB system design and software implementation. Created system allows to generate users-system dialogue and process its results to obtain formalized data about the data domain. The NLIDB pattern matching system approach that allows to set a template for control and clarification questions for users and form an algorithm for transforming knowledge about the data domain into a database structure was used in the research. Depending on the users' responses, the system outputs the domain business rules, database diagram, SQL queries, and database file to users. The results of the developed NLIDB system testing are presented in the paper. The testing of the system was carried out to create a fragment of the database for storing invoice data. In order to check the adequacy of the proposed project solutions, an entity relation diagram for the document was used. The main problem in generating a database structure is the implementation of data normalization rules and checking for anomalies in the created database. Checking and optimizing of the developed database is carried out after populating a database with an initial data, by using a dialogue with users and applying queries to the created database tables.

Keywords

Databases, Natural Language Interface to Databases (NLIDB), Structured Query Language (SQL), pattern-based approach

1. Introduction

Nowadays almost every software, regardless of the field of use, whether it is a mobile application for exchanging messages or services for managing companies, processes a large amount of information that needs to be stored for long periods of time. Databases are usually used to store, process and analyze these data. In order to manage the design and processing of databases, database management systems (DMS) are used. DMS is a set of software and linguistic tools for general or special purposes. Most of the data of modern information systems and programs store data in relational databases (RDB) as a set of the tables. RDB design is a time-consuming process involving domain analysis, studying the data necessary for storage, structuring and presenting data as entities and their characteristics, creating the

IntelITSIS'2023: 4th International Workshop on Intelligent Information Technologies and Systems of Information Security, March 22–24, 2023, Khmelnytskyi, Ukraine
EMAIL: yalovakateryna@gmail.com (K. Yalova); yashinaksenia85@gmail.com (K. Yashyna); abmsalem@yahoo.com (A.-B. M. Salem)
ORCID: 0000-0002-2687-5863 (K. Yalova); 0000-0002-8817-8609 (K. Yashyna); 0000-0003-0268-6539 (A.-B. M. Salem)

relations between them. The result of RDB design is its logical and physical model. A logical database model is a universal model that is independent of a specific DMS, presented in terms of the data domain. A physical database model is a model developed by the means of a specific DMS. It this case the knowledge of the structured query language (SQL) is required to provide information operations of the addition, deletion, updating and data searching in RDBs. Shortcomings in the stage of RDBs design lead to the impossibility or complexity of fixing them in the next stages of software development life cycle. Despite the existence of data normalization rules, the process of RDBs developing is poorly formalized and largely depends on the experience and knowledge of the database architect, so the task of improving the efficiency of the RDB design process remains a pressing scientific and practical task.

Achievements in Natural Language Processing (NLP) enable interaction with RDBs in natural language. The idea of using natural language instead of SQL contributed to the development of a new type of NLP called Natural Language Interface to Database (NLIDB) [1]. The concept of NLIDB is to make data stores more accessible to a wide range of non-technological users, reduce the knowledge requirements of database developers [2]. The ultimate aim of the Natural Language Interface to Database is to organize the users' communication with the database (almost) as with a person. That is, NLIDB is used to simplify work and reduce the time spent to develop and process RDB data. Software tools that use the concept of NLIDB exclude the stage in which knowledge of the SQL language, data normalization rules or RDB design rules are directly used. Using their own software mechanisms, they allow users unfamiliar with database development rules [3]:

- perform queries to create, update, delete the database and its tables in the Data Definition Language (DDL);
- get actual, complete and reliable information from the database as a result of executing the generated SQL queries.

The main goal of the paper is to present the results of the NLIDB system development, which allows to organize a dialogue with the user in such a way that a database is designed as a result of this interaction. To achieve this goal, the following design objectives have been identified and successfully completed:

1. Development of the NLIDB system architecture.
2. Design a dialogue model as a tool to acquire data domain knowledge.
3. Creating an algorithm to transform a verbal domain description into commands to create RD.
4. Implement a NLIDB system with a high-level programming language.
5. Using a data domain document as a complex sample data to test a developed system.

Integrating the NLIDB as an abstraction layer between users and databases provides the main benefit of simplified natural language-based interaction [4]. The main difficulties in solving this problem arising from linguistic failures and the impossibility of general-purpose solutions developing, motivate researchers to continue searching for the effective methodologies and approaches to create NLIDB systems applicable to various data domains.

1.1. Related works

Over the past 50 years, many attempts have been made to create an NLIDB to generate database queries. The first NLIDBs appeared in the late sixties and early seventies. Among the first authors which described an approach of NLIDB implementing were W. Woods, R. Kaplan and B. Webber who in 1972 described the Lunar Sciences Natural Language Information System. Over the next 40 years, scientists such as: I. Androutsopoulos, G. D. Ritchie, P. Thanisch, E. Buchholz, A. Düsterhöft, B. Huang, G. Zhang, J. Mark, N. Nihalani, A. Kumar, A. Kaur, I. S. Bajwa, N. Stratica, S. Maisonneuve devoted their works to the development of effective mathematical and software mechanisms and systems for translating natural language queries into SQL queries.

Two main approaches are used to implement the NLIDB concept:

1. Rules based approaches are the approaches that aim is to design models for automatically mapping natural language semantics into SQL-queries.
2. Neural approaches [5] are approaches that form the neural architecture consists of an encoder and a decoder component for translating natural language sentences into SQL-queries and verse vise.

In turn, rules based approaches can be divided into:

- Pattern-based. The main idea of the approach is to use keywords that allow you to respond to complex requests in natural language. The goal of the approach is to provide the ability to process specific language templates to identify commands. The works of M. D. Agatonovic, H. Cunningham, Zheng [6], X. Xu, C. Liu, D. Song [3] are devoted to the implementation of the pattern-based approach and its application in software systems;
- Parsing-based. The main idea of this approach is to use a natural language parser to analyze the input query and analyze the grammatical structure of the query. Then the grammatical structure can be used to improve understanding of dependencies between query markers. Such scientists as F. Li, H. V. Jagadish, D. Saha, A. Floratou, K. Sankaranarayanan [7], A. Kopp, D. Orlovskyi, S. Orekhov [8] devoted their works to using the parsing-based idea.
- Grammar-Based. The main idea of this approach is to use a set of rules and grammar, to determine how the system can build, understand and answer questions. These rules can then be used to help users enter their queries by auto-completing. Scientists such as D. Song, F. Schilder, C. Smiley, C. Brew, C. Zielund, S. Ferre [9], A. K. Deshpande, P. R. Devale, B. Huang, J. Guo, Z. Zhan [2] created semantic trees to implement the Grammar-Based approach.

Recently, a lot of attention has been paid to the use of conversational interfaces [4] for data analysis, which allows business owners and non-technical users to quickly get database views.

Also, an interesting approach of the NLIDB implementing is presented in [10], where authors describe a method for effectively automating of the queries conversion from a natural language to SQL-query, using data storage objects of an online analytical processing (OLAP) cube.

The works, listed above, are mainly devoted to the creation of NLIDB systems for generating queries for already created databases. The task of designing and creating RDBs based on the verbal description of the data domain transmitted by the user by the means of dialogue with the program remains relevant and has high practical value. The paper, unlike the ones mentioned above, presents the results of the development of the system based on NLIDB approach for converting user commands submitted in natural language to commands for creating RDB. For users, dialog with developed system looks like communication with a colleague about a future database, answering short questions in natural language. The results of user interaction with the system are:

- list of the data domain business rules;
- database diagram;
- generated SQL queries;
- .mdf database file created by MSSQLServer.

2. Proposed methodology

NLIDB eliminates the need to know the database design rules and the specific syntax of SQL [11-13]. Implementation of NLIDB requires understanding the context of natural language sentences, as well as converting these sentences into meaningful executable queries. The paper suggests using rules based approach in the form of the pattern matching system to create a physical model of the RDB. Pattern-based NLIDB is an extension of rules based systems that allow to form more complex templates in native language instead of single keywords [14-16]. The basic idea is to provide the ability to process specific language patterns to identify and generate queries. The main advantage of the pattern matching approach is its simplicity, there is no need for complex parsing and interpretation modules, and systems can be easily implemented [17,18]. The problem with implementing NLIDB is how to deal with the ambiguity of user information submitted in natural language. Author propose to implement the pattern matching NLIDB system so that the user-system interaction is minimized. It is proposed to form a pattern of control and clarifying questions, the answers to which allow to describe the main business rules of the data domain and convert them into SQL commands. This approach is valid for creating queries to create the database needed to store domain data.

At the request of the user, the created database can be populated with data. After populating the database with an initial data it can also be checked for anomalies or violations of normalization rules. To minimize anomalies in the developed databases, data normalization rules will be automatically used by the system. The developed RDB will be normalized to the third normal form.

3. Results

Since the ultimate goal of the NLIDB system is to generate a RDB, it is necessary to describe the generalized mathematical model as follow. RDB is a data structure with the following mandatory components [19]:

- Table – a set of rows and columns filled with values;
- Attribute – a named table column;
- Domain – a set of acceptable values for one or several attributes;
- Tuple – table row.

In generalized form, the relational database model RBD can be presented in the form of tuple:

$$RBD = < T, R >, \tag{1}$$

where $T=\{T_1,\dots T_n\}$ is a set of database tables; $R=\{R_1, R_2\}$ is a set of connections presented in in the database, R_1 – 1:1 connection, R_2 – 1:N connection.

Then the database table set can be described as:

$$T = < A, D, K >, \tag{2}$$

where $A=\{a_1,\dots a_i\}$ is a set of attributes of the n-th table in the database; $D=\{d_1,\dots,d_j\}$ is a set domains of the n-th table containing all the values for each i-th attribute; $K=\{k_1,\dots,k_m\}$ is a set of tuples of the n-th table that combine all attribute values for m-th row.

For each i-th attribute there is a corresponding j-th domain. Let us suppose that the data described by i-th attribute is defined as $t(a_i)$, then the following constraint shall be fulfilled: $t(a_i) \in d_j$.

Further, the attribute set A can be described as:

$$\mathrm{A} = < N, TA, TD >, \tag{3}$$

where $N=\{n_1,\dots n_l\}$ is attribute name set of the n-th table; $TA=\{ta_1, ta_2, ta_3, ta_4\}$ is a set of attribute types: $ta_1=\{ta_{11}, ta_{12}\}$ – key attribute: primary and foreign key, respectively; ta_2 – dynamic attribute, for which it is not possible to prevent the value, ta_3 – static attribute, which value can be repeatedly used in the n-th table , ta_4 – attribute that can be calculated on the values of other attributes, according to the normalization rules such attributes should not be stored in a database; $TD=\{td_1,\dots,td_c\}$ is a set of attribute datatypes for n-th table in the database.

A synthetic attribute $ak \in A$ of type ta_{11} is created as the primary key, so that for any data t_p of attribute ak the condition $t_p(ak) \neq t_{p+1}(ak)$ is true, which provides the following advantages:

- unambiguous identification of table data;
- prevention of duplicated key values;
- speeding up database query execution;
- generation of referral integrity constraints.

To design a qualitative RDB, data normalization rules were used. Normalization rules are a systematic approach to develop database tables in such a way to eliminate data repeatability and avoid insertion, update, and deletion anomalies. Normalization of data in the developed system will be carried out up to the third normal form.

The first normal form (1NF) sets the following main rules:

- attributes must not be multivalued. The most common multi-valued attributes are: addresses, name, dates, phone number, etc. All these attributes can be divided into several single value attributes;
- values in the column must belong to one domain;
- all columns must have unique names;
- the order of data storage is not important.

The second normal form (2NF) states that the table should not be partially dependent on the key fields. The rules of the 2NF allow to form rules for choosing or creating a primary key in each database table.

The third normal form (NF3) dictates the rule that tables should not have transitive dependencies. All non-key fields which contents can refer to multiple external table must be placed in a separate table. NF3 allow to form rules for dividing data between tables and creating correct relationships between them.

Each of the listed rules is important for the full functioning of the database. The relationship between the attribute and the table, as seen from the business rules perspective, describes how the table contains the attribute, for example the conjunction “has a” or “contains”.

The architecture of the developed NLIDB system can be represented as a combination of two components: a user dialogue module and a SQL query generation module. Authors propose to supplement the user dialogue module with a pattern of questions, with the help of which a managed user dialogue will be formed in the system. A generalized architecture diagram is shown in Figure 1.

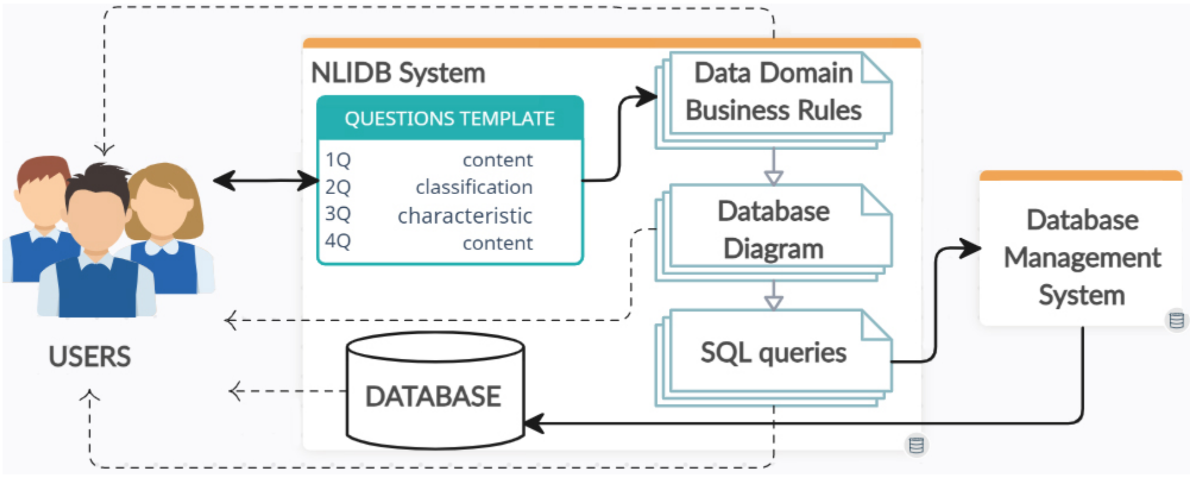


Figure 1: Architecture of the NLIDB system

The generalized algorithm of users’ interaction with NLIDB system modules includes the following main steps:

1. Acquiring information about the desired database as a whole.
2. Getting information about the described entity as a whole.
3. Gets information about the set of attributes of the selected entity.
4. Perform a managed dialogue to find dependencies and relationships between entity attributes, establish their type and a multiplicity of these relation. On the system side, the received data is normalized, primary keys are added if necessary, new tables are formed from subsets of attributes.
5. Performing a managed dialogue, the purpose of which is to find dependencies between entities and establish a multiplicity of these relations. If it is necessary, the system adds foreign keys and additional tables.
6. Obtaining a data domain abstraction of an entry-level.
7. Check, edit, adjust business rules and inputted data.
8. Generate SQL queries and create a database file.

3.1. Dialogue model

The input to the NLIDB system is the verbal knowledge of the user about the data domain. To create a conversation with users, a glossary of terms is used to simplify mutual understanding between the user and the system. The main terms of the glossary of the system are:

- A data domain is a part of the real world to be studied in order to organize management and, ultimately, automation of it.
- A data domain entity is the result of abstracting a real object by highlighting and fixing a set of its properties (attributes).
- A domain entity attribute is a characteristic of an entity that can be represented as a numeric value (a quantitative property) or a description (an indicative property).

The goal of a dialogue with a user is to obtain information about the data domain for which the database will be created, to specify the entities and relations between them. In computer systems where any typing errors can lead to significant problems, it is recommended to minimize the amount of text entered by the user. Instead of prompting the user to enter entire sentences, only new information, such as entity and attribute names, was left to be entered. All other interaction with the system should be done through markers, drop-down lists, and other components that do not require manual input of information [20]. Therefore, the dialogue model was implemented in the form of a question-answer interaction. The model of the dialog was based on the pattern of the control and clarifying questions and the algorithm for transitions between them, depending on the already received answers. Tree types of questions were defined to create a pattern:

1. Q_{con}: content question. The users' response for the content question is the text entered from the keyboard. Obtained data allows to form a set of tables T, a set of attributes A, a set of attribute names N, a set of values K of the RBD.

2. Q_{clas}: classification questions. The answer to the classification questions allows to determine the type of connection between attributes inside entity or between entities and establish the multiplicity of such connection (R_1 or R_2), determine candidates for foreign keys, form a set of relationships R of the RBD, bring the data to normal forms. The system receives the users' response through drop-down lists.

3. Q_{char}: characteristic questions. The characteristic questions are aimed at obtaining information on the attributes data type, amount of stored data, obligation of their filling. The response to the characteristic question makes it possible to determine candidates for primary keys, allows to form a set of domains D, a set of attributes types TA and their data types TD.

Table 1 provides an abbreviated pattern of the main control questions that are used to form a user-system dialogue.

Table 1
Pattern of the main control questions

Type	Question	User activity
$1Q_{con}$	What is the data domain	Enter
$2Q_{con}$	What is the data domain entity name	Enter
$3Q_{con}$	What is the name of the entity attribute	Enter
$4Q_{con}$	Do you want to set a set of values?	Choose: Yes/No
$5Q_{con}$	What is the set of values for the current attribute?	Enter
$6Q_{con}$	Do you want to add another attribute of the selected entity	Choose: Yes/No
$7Q_{con}$	Do you want to add another entity	Choose: Yes/No
$1Q_{clas}$	Whether an entity consists of other entities	Choose: Yes/No
$2Q_{clas}$	How many instance of the shown entity belongs to the selected entity	Choose: 1 / many
$3Q_{clas}$	Is it possible to calculate the attribute value from values of another attribute?	Choose: Yes/No
$4Q_{clas}$	Can the same entity attribute values be repeated	Choose: Yes/No
$5Q_{clas}$	Whether the value of the attribute can be divided into separate parts	Choose: Yes/No
$6Q_{clas}$	Whether the same attribute is present in another entity	Choose: Yes/No
$1Q_{char}$	Can the attribute value be empty/missing	Choose: Yes/No
$2Q_{char}$	Whether the attribute values are unique/unrepeated	Choose: Yes/No
$3Q_{char}$	Whether the entity characteristic is quantitative or indicative	Choose: Yes/No
$4Q_{char}$	What data type can the selected attribute acquire	Choose: numeric, String, DateTime
$5Q_{char}$	Can a numeric value contain fractional values	Choose: Yes/No
$6Q_{char}$	What is the maximum number of characters can be contained	Enter
$7Q_{char}$	Which date format is suitable for attribute values	Choose DateTime format

In addition to describe attributes and objects by the user, the system has a mechanism for adding key attributes, references to external entities in accordance with the data normalization rules in automatic mode. Key attributes are added depending on the answers to questions $1Q_{char}$, $2Q_{char}$. If no primary key candidates are found, a synthetic primary key is added to the table according to the NF2. If the answers to the questions $1Q_{class} = =$ yes & $4Q_{class} = =$ repeating value for i_{th} attribute, the system will change the attribute name to the foreign key to external entity.

In the NLIDB system, users can choose the level of abstraction for the described data domain:

- list of data domain business rules. Business rules are the verbal definitions of database objects with their attributes, relationships and constraints.
- database diagram. In this case, the user is given a database diagram on the screen, presented in the form of a set of tables with shown relations.
- generated SQL queries will be presented in the form of text with the ability to save as the MSSQLServer query file.
- .mdf database file created by MSSQLServer.

The entry-level of data domain abstraction is the set of business rules formed based on the users' answers to the questions presented in the Table 1. Converting acquired information into business rules, and demonstrating these business rules to the user, makes the users' interaction with the system more convenient and understandable, because it is possible to analyze business rules of the data domain and if there are some mistakes or inaccuracies user can return to the dialogue with system and make the corrections. After verifying the correctness of the generated business rules, the user can request a database diagram from the system. The system outputs the database diagram in order to allow the user to make sure that the data domain description is correct. At this stage, the database diagram is not created by DMS, but rendered on the form as a graphical element. The business rules list and the database diagram are the logical level database model. After that, the system generates queries and creates a physical database model.

3.2. NLIDB system implementation

The C # language is selected as the programming language. The system is designed in accordance with an object-oriented approach. All dialogue data will be temporary stored in the format of class instances. The SQL language and database access tool will be used to interact with the DMS and execute queries by means of the ADO.NET technology. For the interface part of the system, WPF application will be used as a flexible and universal way to create a convenient simple interface. The system generates SQL queries, as well as a database file. The MSSQLServer DMS is used for this purpose. Although the user can use the created queries in various DMSs that support relational approaches and can use SQL.

3.2.1. Architecture of the NLIDB system classes

The following classes are used to temporarily storage of the database structure:

- Table class, that contains the name of the table and a list of attributes of this table. The table class contains methods for interacting with the database. For example, methods that generate a query to create a table in the database.
- Attribute class, that contains information about the attribute name, its data type, size, and other properties. The attribute class has a method returns string with information about the attribute to construct database queries.
- Inherited Key attribute and Calculated attribute classes. These classes contain specific attributes and methods and provide the ability to store primary and foreign keys, as well as to store the calculated attributes.
- Database class for working with the entire database. The class contains information about database name and the value of the connection string.

The class diagram of the NLIDB system is shown in Figure 2.

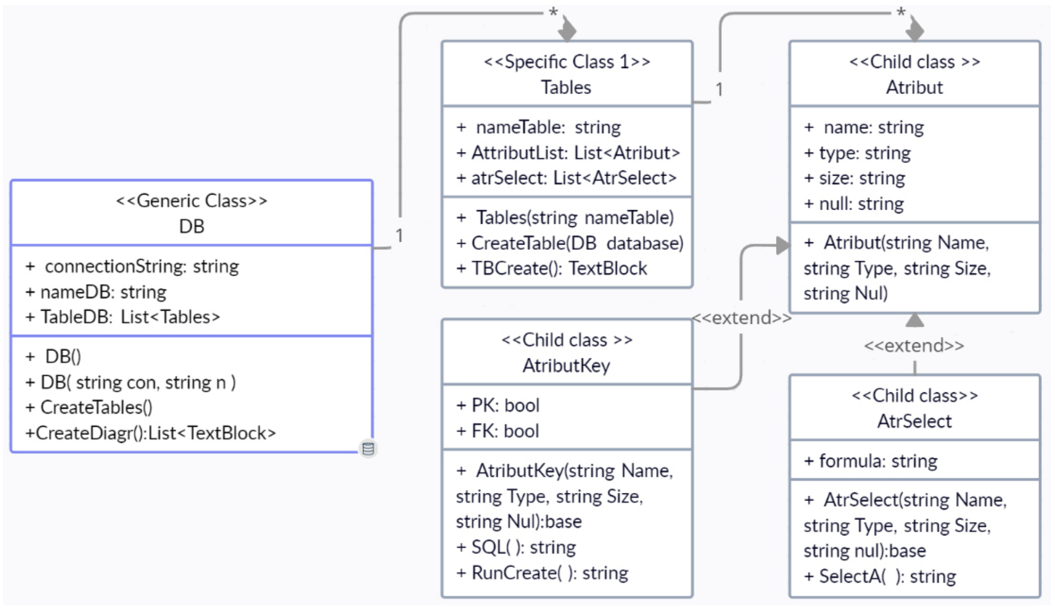


Figure 2: Class diagram of the NLIDB system

Information stored as instances of classes is used to create business rules, construct queries and create a database file. For example, in the last phase of the users' interaction with the system it sends queries for the DMS to create a database. An example of a database table creation function for already existed database is as follows:

1. Create a query string. To do this, use the word "use" and add the corresponding name of the database. Then add the statement "create table". Extract the name of the table from the user inputted data stored as a Tables class instance. Extract the data about attribute from the list of attributes. Complete the "create table" query statement.
2. Open the connection to the database.
3. Execute the command created in the first step.
4. Output a message to the user that the table has been created.

3.2.2. NLIDB system user interface

The application interface is built by means of the WPF graphics subsystem from .NET, which uses the XAML language. It is used to create GUI modules.

Since most of the system is transferred to classes, the application has one main window with different tabs. The transition between tabs is implemented by using additional buttons, based on the algorithm of interaction with the user. That is, the transition to the next step of the user interaction algorithm occurs only after filling the data in the fields, and their storage for further work with them. Direct navigation between tabs as provided in WPF is blocked to prevent unexpected pathway of the users.

3.2.3. Test example

To demonstrate the work of the developed NLIDB system, it is proposed to consider the result of the development of a database fragment for storing invoice document. Invoice is a primary document used in the transfer of inventory from one person/ company to another. The invoice document was selected as test data because:

- it is a composite entity that contains and references external entities;
- contains data to be normalized, for example: address and phone attributes;
- contains attributes of different types: static (customer name, supplier name, good name, etc.), dynamic (date of creation number, amount, etc.), calculated (tax, subtotal, total), keys (references for the external data domain entities).

In order to present the object model of the invoice document the entity-relation diagram of an invoice document was designed. The developed entity relation diagram is shown in Figure 3 and used to assess the adequacy of the data obtained from the system.

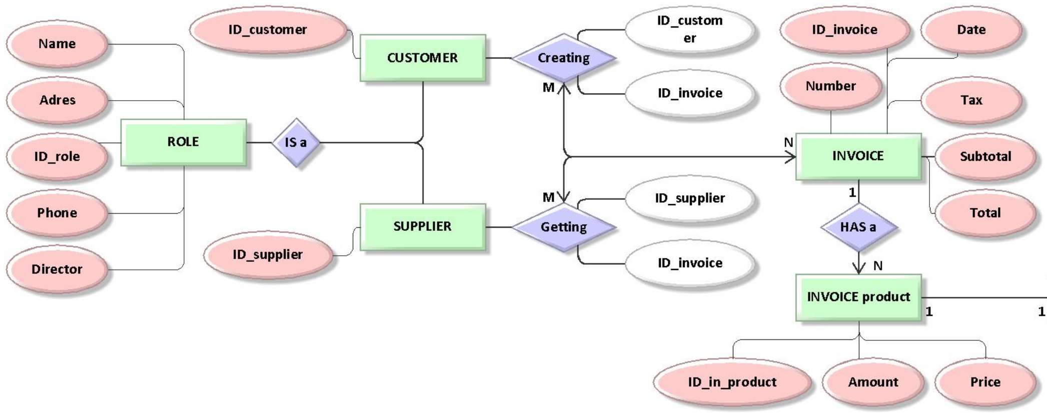


Figure 3: Entity relation diagram for an invoice document

The structure of the document is complex, so the dialogue with user will have several iterations with control and clarifying questions. It allows to test the pattern fully and check the adequacy of the system in a whole.

To generate data domain business rules, a template and algorithm converting user responses to a domain business rule were developed and used. The business rule is a brief, structural formalization of the data domain, presented verbally in natural language. The business rule template used verbs:

- *has a* – to describe a situation when an entity contains or references another entity. Such a verb is used if the question $1Q_{class} == yes$ & $4Q_{class} ==$ repeating value. The answer to the question $2Q_{class}$ makes it possible to determine the type of relationship between entities: composition or aggregation. Composition is a situation when external entity cannot be performed as independent part of the data domain. Aggregation demonstrates a possibility to refer and use attributes of the external entity.
- *contain* – to describe all the attributes that characterize the current entity. In this case business rules are formed using answers to questions $2Q_{con}$, $3Q_{con}$, $5Q_{con}$, $6Q_{con}$, $7Q_{con}$.
- *is calculated as* – to describe characteristics that can be defined based on other attributes by using specified calculation formulas. The answer to the question $3Q_{class} ==$ yes, makes it possible to form business rules with such a verb.

Table 2 shows the list of business rules generated during the users-system dialogue for describing the Invoice document.

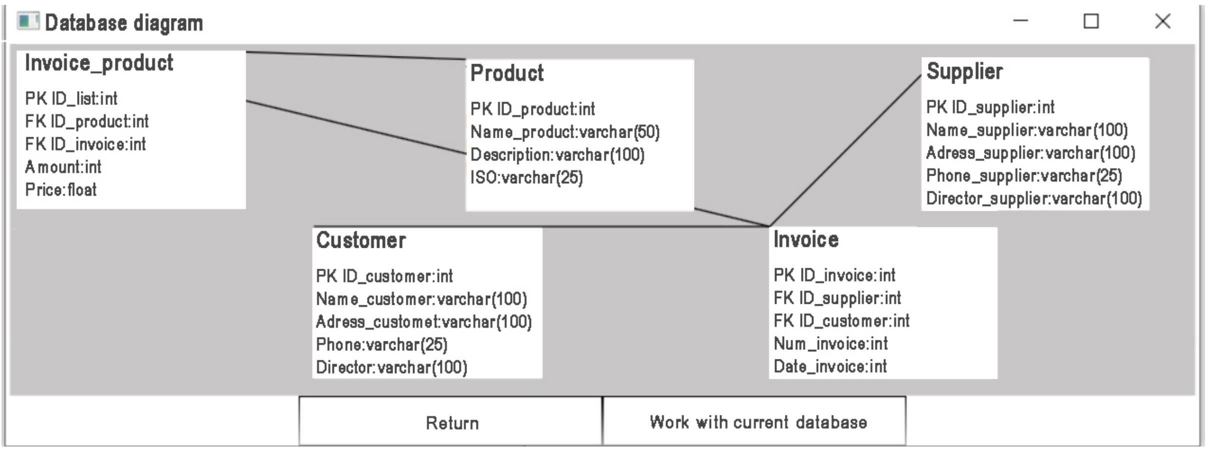
Table 2
Generated data domain business rules

Rule	Description	Type
BR1	Invoice has products	Composition 1:N
BR2	Invoice has a customer	Aggregation 1:1
BR3	Invoice has a supplier	Aggregation 1:1
BR4	Invoice product has a product	Aggregation 1:1
BR5	Invoice is described by <Number, Date, Customer, Supplier, List<Product>, Amount, Price, Subtotal, Tax, Total >	Attribute
BR6	Subtotal is calculated as SUM(Amount*Price)	Calculated Attribute
BR7	Tax is calculated as Subtotal*0,20	Calculated Attribute
BR8	Total is calculated as Subtotal + Tax	Calculated Attribute
BR9	Customer is described by <Name, Address, Phone, Director>	Attribute
BR10	Supplier is described by <Name, Address, Phone, Director>	Attribute
BR11	Product is described by <Name, Description, ISO>	Attribute
BR12	Invoice product is described by <Product, Amount, Price>	Attribute

The generated list of business rules provides a possible to obtain information about the structure of the data domain as the entry level of abstraction on the way to the database. If desired, the list can be saved and imported. There is no feedback in interaction with the list of business rules. If users need to make changes to the rules, it is necessary to update the entered information through the main dialogue with the system.

If the user agrees with the presented description of business rules, he can proceed to the graphical display of the future database. The NLIDB system implements TextBlok objects that are created and used to map the data of each entity to database diagrams in a separate application window.

Figure 4 shows an example of a database diagram created to display the structure of invoice document in terms of the future database. This is not a database diagram generated by DMS tools, it is only a graphical representation of the database content, created by the NLIDB system to improve visibility and for the verification.


Database diagram

Invoice_product
PK ID_list:int
FK ID_product:int
FK ID_invoice:int
Amount:int
Price:float

Product
PK ID_product:int
Name_product:varchar(50)
Description:varchar(100)
ISO:varchar(25)

Supplier
PK ID_supplier:int
Name_supplier:varchar(100)
Adress_supplier:varchar(100)
Phone_supplier:varchar(25)
Director_supplier:varchar(100)

Customer
PK ID_customer:int
Name_customer:varchar(100)
Adress_customet:varchar(100)
Phone:varchar(25)
Director:varchar(100)

Invoice
PK ID_invoice:int
FK ID_supplier:int
FK ID_customer:int
Num_invoice:int
Date_invoice:int

Figure 4: Database diagram generated by the NLIDB system

After receiving information about data domain entities and attributes, the user is prompted to set values for each entity attribute. The system uses the Insert command to add values to the creating database. The entered values are also used by the system to check the correctness of the generated database structure or to check the presence of anomalies in the database. For example, the presence of multiple space characters in the values of a string data type can signal that the user has created a multivalued field. And the system will issue a clarifying question by using the pattern:

Attribute <name> + $5Q_{class}$. Does not it?

If the user answers yes, the user is prompted to repeat the information entry process for each part of the multivalued attribute separately.

Another important clarifying question is the question with the following pattern:

Attribute <name> + $1Q_{char}$ + $2Q_{char}$. Does not it?

Such a clarifying question will be generated by the system in the case when the validation mechanism finds repeated values in the field declared as a candidate for the primary key. In this case, the user will be given information about the inadmissibility of duplicating the value in the specified field, otherwise a synthetic primary key with an auto-increment mechanism will be added to the table instead of the selected candidate. The system also provides a mechanism for validating the values entered by the user in accordance with the selected data type.

Absence of system comments means the finish of the users-system dialogue. As a result, the user can save SQL commands or load the generated .mdf file. The system constructs the query to the MSSQLServer based on the CREATE, INSERT commands to create a database and its tables, as well as populate it with data if necessary. The choice of this particular DBMS is not a limitation for the system, since the generated SQL queries are universal and can be used to create a physical database model in different DMS.

If users answer the questions correctly, the system adequately generates results. The figure 4 shows a database diagram generated by the system to discuss the shortcomings of the system using a concrete example. The main barrier for the efficient use of the NLIDB system is a significant dependence on the user. Despite the implemented templates and validation mechanisms, the user may not understand the question or enter deliberately untrue data. In this case, the inaccuracies will be appeared in the results generated by the system. Such inaccuracies can be seen by comparing the entity relation diagram (Figure 3) and the constructed database diagram (Figure 4). The address and phone fields were marked by the user as indivisible attributes and only at the data validation stage, after the database population, the system identifies the problem of 1NF violation. Another example is a misunderstanding of the user when answering $6Q_{clas}$ question. The incorrect user answer leads to the fact that the system does not optimize the structure of tables and in the database diagram, there is a subsets of attributes that can be brought into a separate table.

Among the advantages of proposed approach and the NLIDB system, the authors distinguish the following:

- unnecessary to have knowledge of the syntax SQL, due to the fact that the system operates only with natural language;
- convenient, intuitive, simple interface;
- possibility of obtaining an intermediate level of abstraction on the way to the database to make it possible to check and update inputted data;
- minimizing the amount of the entering data from the keyboard operations, by using short and clear instructions to the user through the question-answer dialogue.

4. Conclusion

NLIDB is a system that allows users to communicate with database by entering queries expressed in some natural language. Integrating the NLIDB as an abstraction layer between users and databases provides the main benefit of simplified natural language-based interaction. The purpose of the NLIDB is to enable users to compose questions in the natural language and to receive answers in the natural language as well. The authors propose to apply an NLIDB approach and develop the NLIDB system. In this case the aim of the user-system interaction is to acquire verbal knowledge about the data domain and transforming it into a corresponding database structure.

The developed NLIDB system allows to convert the description of the data domain presented in natural language to the commands for creating RDB. RDB is created in accordance with the data normalization rules. Users enter detailed information about the entities, attributes and relations in natural language, and template of controlling and clarifying questions allow to convert this description into components of SQL queries. In order to ensure the efficiency of the process of converting the data domain verbal description into SQL statements the intermediate-level abstractions are generated by the system. Intermediate-level abstractions are presented as a list of the data domain business rules, database diagram and SQL queries. The final result is the database physical model. Each abstraction can be evaluated and, if necessary, improved by user through the main dialogue with the system.

The presented approach, unlike the existing ones, allows reducing the threshold for users' knowledge in database theory, reducing the process of database development and increasing the efficiency of this process as a whole. Developed NLIDB system architecture, designed dialogue model with pattern for controlling and clarifying questions are displayed in the paper. The NLIDB system was implemented as desktop application with interface built by means of the WPF from .NET. ADO.NET technology and SQL are used to provide operations with database inside MSSQLServer MDS. The results of the NLIDB system are also described in the paper. As the test sample data invoice document was used. Generated results were compared with the entity relation diagram. Various system decision making results have been described to reflect the advantages and disadvantages of the proposed NLIDB system.

The scope of the following researches is to implement NLIDB module that propose users to interact with an already created database to construct queries in natural language and obtain the results of these queries execution.

5. References

[1] D. Radovanovic, Introducting natural language interface to database for data-driven small and medium enterprises, in: P. Haber, T. Lampoltshammer, M. Mayr, K. Plankensteiner (Eds.), Data Science – Analytics and Applications, Springer – Vieweg, Wiesbaden, 2021, pp. 11–15. doi: 10.1007/978-3-658-32182-6_3.

[2] J. Guo, Z. Zhan, Y. Gao, Y. Xiao, J. Lou, T. Liu, D. Zhang, Towards complex text-to-sql in cross-domain database with intermediate representation, in: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, AMACL '19, Association for Computational Linguistics, Florence, Italy, 2019, pp. 4524–4535. doi: 10.18653/v1/P19-1444.
[3] X. Xu, C. Liu, D. Song, Sqlnet: Generating structured queries from natural language without reinforcement learning, Commutating and language, 11 (2017) 1–13. doi: 10.48550/arXiv.1711.04436.
[4] A. Quamar, V. Efthymiou, C. Lei, F. Özcan, Natural Language Interfaces to Data, Foundations and trends in databases 11 (2022) 319–414. doi: 10.1561/1900000078.
[5] R. C. Iacob, F. Brad, E.S. Apostol, C.O. Truică. I.A. Hosu, T. Rebedea, Neural approaches for natural language interfaces to databases: a survey, in: Proceedings of the 28th International Conference on Computational Linguistics, ICCL '20, International Committee on Computational Linguistics, Barcelona, Spain, 2020, pp. 381–395. doi: 10.18653/v1/2020.coling-main.34.
[6] W. Zheng, H. Cheng, L. Zou, J. Yu, K. Zhao, Natural language question/answering: let users talk with the knowledge graph, in: Proceedings of the Conference on Information and Knowledge Management, CIKM '17, Association for Computing Machinery, Singapore, Singapore, 2017, pp. 217–226. doi: 10.1145/3132847.3132977.
[7] D. Saha, A. Floratou, K. Sankaranarayanan, F. Minhas, A. R. Mittal, F. Özcan, ATHENA: an ontology-driven system for natural language querying over relational data stores, VLDB Endowment 9 (12) (2016) 1209–1220. doi: 10.14778/2994509.2994536.
[8] A.M. Kopp, D. L. Orlovskyi, S. V. Orekhov, An approach and software prototype for translation of natural language business rules into database structure, in: Proceedings of the 5th Internation Conference on Computational Linguistics and Intelligent Systems, COLINS '21, CEUR WS, Lviv, Ukraine, 2021.
[9] A. Ferre, Sparklis: an expressive query builder for SPARQL endpoints with guidance in natural language, Semantic web 8(3) (2017) 405–418. doi: 10.3233/SW-150208.
[10] F. H. Hazboun, M. Owda, A. Y. Owda, A Natural Language Interface to Relational Databases Using an Online Analytic Processing Hypercube, Artificial Intelligence, 2(4) 2021 720–737. doi: 10.3390/ai2040043.
[11] A.H. Kumar, A.R. Kunar, P. Harshitha, D.N. Sachin, providing natural language interface to database using artificial intelligence, International journal of scientific and technology research, 8 (2019) 1074–1077.
[12] P. Anand, Z. Farooqui, Rule based domain specific semantic analysis for natural language interface for database, International journal of computer applications, 164 (2017) 21–28. doi: 10.5120/ijca2017913502.
[13] C. Tapsai, P. Meesad, Natural Language Interface to database for data retrieval and processing, Applied science and engineering progress 14 (2021) 435–446. doi: 10.14416/j.asep.2020.05.003.
[14] S. Walelign, D. Tesfaye, T. Kebebew, Modelling and Designing of NLP Interface to Database for Afaan Oromoo, International Journal of Innovative Research in Computer Science & Technology, 9 (2021) 6–13. doi: 10.21276/ijircst.2021.9.2.2.
[15] W. Wang, Y. Tian, H. Xiong, H. Wang, W.-S. Ku, A Transfer-Learnable Natural Language Interface for Databases, Computer Science 1 (2018) 1–28 doi: 10.48550/arXiv.1809.02649.
[16] N. Choudhary, S, Gore, Pattern based approach for natural language interface to database, International journal of engineering research and applications, 5(1) (2015) 105–110.
[17] W.Wang, Y.Tian, H. Wang, W.Ku., A natural language interface for database: achieving transfer-learnability using adversarial method for question understanding, in: Proceedings of the 36th International Conference on Data Engineering, ICDE '20, IEE ICDE, Dalas, USA, 2020. doi: 10.1109/ICDE48307.2020.00016.
[18] F. Nartey, F. Amavi, I. Nyameamah, An approach for building natural language database using pattern matching technique, International Journal of Computer Applications 177 (2019) 19–22. doi: 10.5120/ijca2019919720.
[19] A. Alghamda, M. Owda, K. Crockett, Natural language interface to relational database (NLI-RDB) trought object relational mapping (ORM), Advances in Computational Intelligence Systems, 1 (2017) 449–464. doi:10.1007/978-3-319-46562-3_29.
[20] F. Hazboun, M. Owda, A, Y. Owda, settings Order Article Reprints
[21] K. Affolter, K. Stockinger, A. Bernstein, A Comparative Survey of Recent Natural Language Interfaces for Databases, Very large databases, 28 (2019) 793–819. doi: 10.1007/s00778-019-00567-8.