An Algorithm for Translation of a Natural Language Question into SQL Query

Mariya Zhekova[1], George Pashev[2], George Totkov[3]

[1] *University of Food Technologies, 26 Maritsa Blvd., 4000 Plovdiv, Bulgaria*
[2] *University of Plovdiv "Paisii Hilendarski", 24 Tzar Asen Str., 4000 Plovdiv, Bulgaria*
[3] *National Evaluation and Accreditation Agency, 125, Tzarigradsko Chaussee Blvd., bl. 5, 1133 Sofia, Bulgaria*

Abstract:
The research presents an algorithm for creating a programming language and query language-independent software module that handles the task of extracting information from a relational database from a natural language user question. The goal can be achieved through pre-built models describing concepts from the domain area under consideration, management models, and question templates. The analytical capability of the proposed algorithm allows language units identified by the question to be mapped to objects from the database, information about which can be found and returned, in the form of a response from the system. Logical programming methods and processing in the algorithm do not depend on the programming language and technologies used. In two papers, the authors describe different software solutions in two programming languages (C# and Python) following the proposed algorithm.

Keywords:
natural language analysis, natural language processing, information retrieval algorithm

1. Introduction

Language, as a physical object of abstract structures of ordered units, serves to express and convey thoughts. The systematic description of this object, its main elements, and its technologies, make it the most powerful communicative tool. Language modelling makes the connection between reality and rules, between information flow and its meaning.

The presented research is about an algorithm (a type of methodology for creating a software module), independent of the programming language and of the query language, which copes with the task of extracting information from a relational database from a freely set user question.

With the help of natural language analysis tools, through a set of strategies, algorithms, functions, and processing, it is possible to implement a natural language interface (NLI) to an information system (IS), which extracts data from the system's database. The goal can be achieved through purpose-built models describing domain area (DA) concepts, control models and question templates for a specific IS [1].

A key point in the value extraction process is providing access to relevant structures and data designed based on the proposed models. The steps for creating a Model of DA related to an IS database, a Linguistic database of concepts and question templates have been implemented in advance and are additional resources to the system's database [1].

The process model described in [1] for the creation of NLI to IS implies the use of a software tool to automate the processes and activities included in it.

The problem of answering a user query in natural language (NL) is solved using a system of knowledge imported from external sources and newly created models, functions, and procedures. It

Education and Research in the Information Society, October 13-14, 2022, Plovdiv, Bulgaria
EMAIL: m_jekova@uft-plovdiv.bg ; georgepashev@uni-plovdiv.bg; totkov@uni-plovdiv.bg
ORCID: 0000-0002-2501-4412; 0000-0001-8148-4737, 0000-0001-9413-3565

should be noted that producing a response from the database requires the agreement of all elements involved in the process.

There is no general, reasoned consensus on the semantic representation of the input user question. The user can freely ask any NL. Input data can be longer or shorter, stand-alone sentences with a linear structure, or strings with more complex formatting.

2. Previous work

According to [2], software systems that process NL queries can be categorized based on their approach to describing and extracting knowledge.

In [4], data is accessed through database management techniques and information is extracted based on keywords. The production rule-based approach can be tuned to return only safe conclusions, limiting failures [5]. Rules are a type of predicate logic with added components that indicate how the information is used in reasoning.

In [8], the authors propose a new dataset by standardizing and correcting previous errors in existing datasets through a simple but effective baseline system. The authors introduce a slot-filling pattern.

The system presented in [9] is based on a set of rules. It accepts user text and identifies the purpose of the request and the domain (topic) that matches the request. The identification process has two phases:

- the system segments the user request into tokens;
- received tokens are tagged with additional information.

The semantic approach in [10], in addition to syntactic recognition, also offers a set of semantic expansion algorithms that use the semantics in ontologies.

The approach in [11] involves the use of a neural semantic analyzer, an SQL query executor, an error detection mechanism, and a response generator. The database schema is analyzed by a SQL parser. A strategy for detecting incorrectly asked questions and providing feedback and possible reformulation of user requests is proposed. System-side response synthesis is implemented using a pattern-based approach [11].

3. Algorithm for translating the NL text into a SQL query to the database

Understanding natural language from the software application and extracting a response from the system occurs without the help of controls, tools, and a GUI, but with NLI and software modules for NLP and transforming text into a database query. Only the type of information context is distinguished, namely to which database the queries are directed. The compliance rules are customized for the specific system, but the Model of domain area, the Linguistic Model and question templates are common to systems in the same domain area.

The analytical capability of the **proposed algorithm** allows identified and annotated language units to be mapped to objects from the database, information about which can be discovered and returned, in the form of a response from the system.

A software tool created according to the steps of the algorithm in this study consists of program methods, checks, and processing that do **not depend on the programming language and the technologies used**. In other scientific papers [6] and [7], the authors propose a different software solution in another programming language (C# and Python) applying the same algorithm. The authors aspire is to show the analytical ability and the independence of the proposed algorithm from the language of user queries and the expected responses from the system.

The overall development process of the software tool combines the following two modules S_Analyze and S_Syntheze:

The **S_Analyze module** takes care of:

- Providing metadata for language objects;
- Processing and analysis of the received linguistic data;

Module S_Syntheze is responsible for:

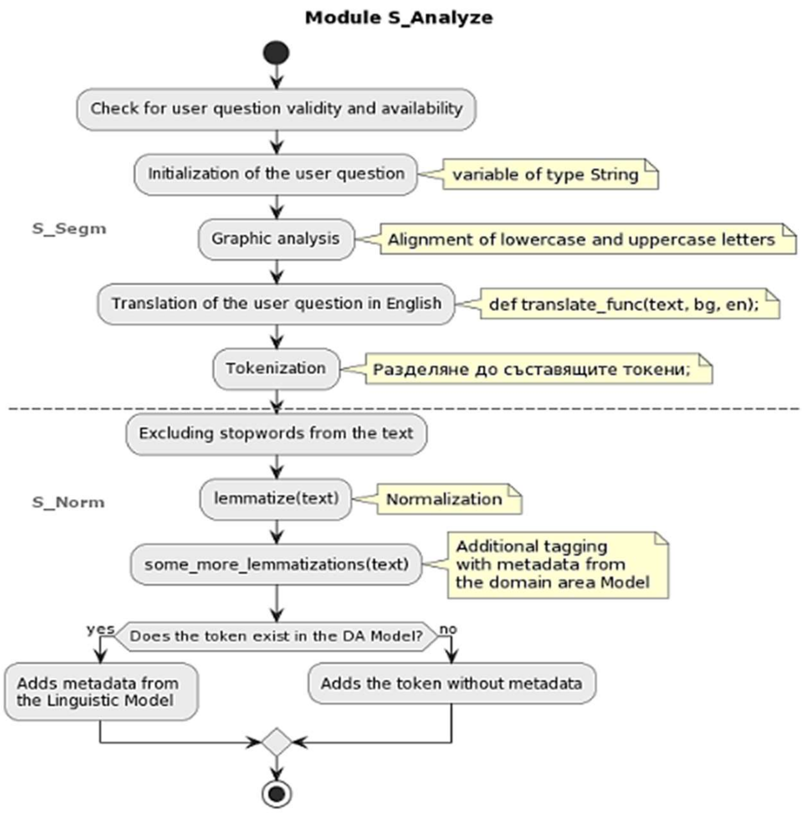
- Formation of a request from the explicit and implicit information in the question;

- Retrieve a response from the database of a specific IS.

A correct query to the database is obtained after processing user EE text in the S_Analyze module, reaching the language models encoded in its constituent parameters, analyzing the relative relationships between the parameters, and forming a query to the database in the S_Syntheze module.

3.1. Algorithm for building the S_Analyze module

- **Module S_Analyze** includes sub modules **S_Segm** and **S_Norm**.



Module S_Analyze
Check for user question validity and availability
Initialization of the user question
variable of type String
S_Segm
Graphic analysis
Alignment of lowercase and uppercase letters
Translation of the user question in English
def translate_func(text, bg, en);
Tokenization
Разделяне до съставящите токени;
Excluding stopwords from the text
S_Norm
lemmatize(text)
Normalization
some_more_lemmatizations(text)
Additional tagging with metadata from the domain area Model
yes
Does the token exist in the DA Model?
no
Adds metadata from the Linguistic Model
Adds the token without metadata

Figure 1. Logical scheme of processing in module S_Analyze

3.1.1. Algorithm for building S_Segm submodule

- The sequence of program processing in **sub module S_Segm** is as follows:

Step 1. The implementation of the processes of NLP and analysis of language units begins with the entry of a natural language text and check for its validity and availability;

Step 2. The process of creating a request to the database starts with the initialization of the user question (variable of type String), which is later used in the normalization process;

Step 3. Computer identification of the words in the string is based on the indivisibility property. A graphic analysis of the received user text is carried out, namely – recognition of the boundaries of individual sentences, words, and fragments of the input text, using punctuation marks, capital letters, spaces, indents, etc. Detected punctuation that logically separates the input string is checked. Alignment of lowercase and uppercase letters.

Step 4. Before the actual tokenization in the program code, the received NL text is translated into English words. Through the multipurpose corpus for the English language WordNet, which is a network of concepts connected by semantic and morphological relations to each other, the software solution added a universal character. From WordNet, information on basic word form, part of speech, synonym order, pronunciation, stress, etc. is automatically extracted.

If there is a morphological analyzer and lemmatizer created for the natural language, the processing is possible without translation into English.

Step 5. When the input text is composed of several sentences, segmentation (tokenization) is performed – dividing the input text into separate sentences based on punctuation marks for the end of a sentence and of the resulting sentences into their constituent units (tokens).

After distinguishing the text markers in S_Segm, grammatical identification of the linguistic units follows in the sub module S_Norm.

3.1.2. Algorithm for building S_Norm submodule

The sequence of program processing in **sub module S_Norm** is as follows:

Step 1. Remove special characters and stopwords from the string. The inclusion of unnecessary and meaningless words, such as interjections, conjunctions, pronouns, etc., increases search time, decreases accuracy, and complicates filtering methods.

Step 2. Extraction of essential information about the grammatical properties of the identified tokens, incl. normalization of the resulting word forms to their basic form (lemma). Different word forms are classified according to grammatical characteristics.

Step 3. But language segmentation and normalization alone are not enough to generate a database query. After only words relevant to the analysis are left, additional lemmatization and tagging of the received tokens are applied. The type and relationships between the received tokens are determined and additionally annotated with information about the role that the object occupies in the system's database – name of a table in the database, name of a column in a table, field value, default column in a table, relation, aggregate, built-in function or context-specific function, etc. As a result, the linguistic knowledge of the input tokens is replaced by named objects from the DB, which makes it possible to form a query for the DB.

The analysis performed in the S_Analyze module achieves the set goal to recognize, classify and transform language units in the text to query language parameters.

3.2. Algorithm for building the S_Syntheze module

Module **S_Syntheze** generally includes sub modules **S_QuestForm** and **S_GenAnsw**.

3.2.1. Algorithm for building S_QuestForm submodule

In the S_QuestForm module, additional processing is performed on the received tokens. As a result, language units are annotated with categories of tags, which in sub module S_GenAnsw form the final query to the database.

The logical sequence and order in the algorithm of program processing in the **sub module S_QuestForm** follow the following steps:

Step 1. Each token is checked for belonging to one of the properties – table name, column name, relation, value, function, constraints, etc.;

Step 2. Relative relations and associativity between the obtained tokens are searched.

Step 3. It is checked for the existence of display clauses, functions, conditions, and related mathematical operators, numbers, dates, and attribute values forming database query structures;

Step 4. The variables are labelled with the appropriate tags – tn, cn, dc;

Step 5. Check if the token is a value type. Two types of token values are distinguished – known values (from nomenclatures) and free values (stored in fields in the database, those that are extracted). The first type of values are marked with a val tag and with a cn tag for belonging to a column name, and from there to a table name;

Step 6. Search for a token of type cn to the left and/or to the right of a free value to attach to it. When a cn token is found on either side of the value, look for the closest distance (number of words) to it, and attach the value to the closer in token;

Step 7. Check for the existence of aggregate (summarizing) function aggr_fun. A token of type aggr_fun must also be concatenated with a nearby token of type cn to implement the function. Searching for the nearest left and/or right token cn to attach to the function.

The strategy for searching for a token of type cn to attach to a token value (val) or function (fun or aggr_fun) follows the following logic:

- The distance between the val value and the found column names on both sides of the token list is checked;
  - Compares the number of words between the current and the found cn token on the left;
  - Compares the number of words between the current and the found cn token on the right;
- If the distance to the left token cn is less, it prefers it;
- If the distance to the right token cn is less, it prefers it;
- If they are equally close, prefer the right one.

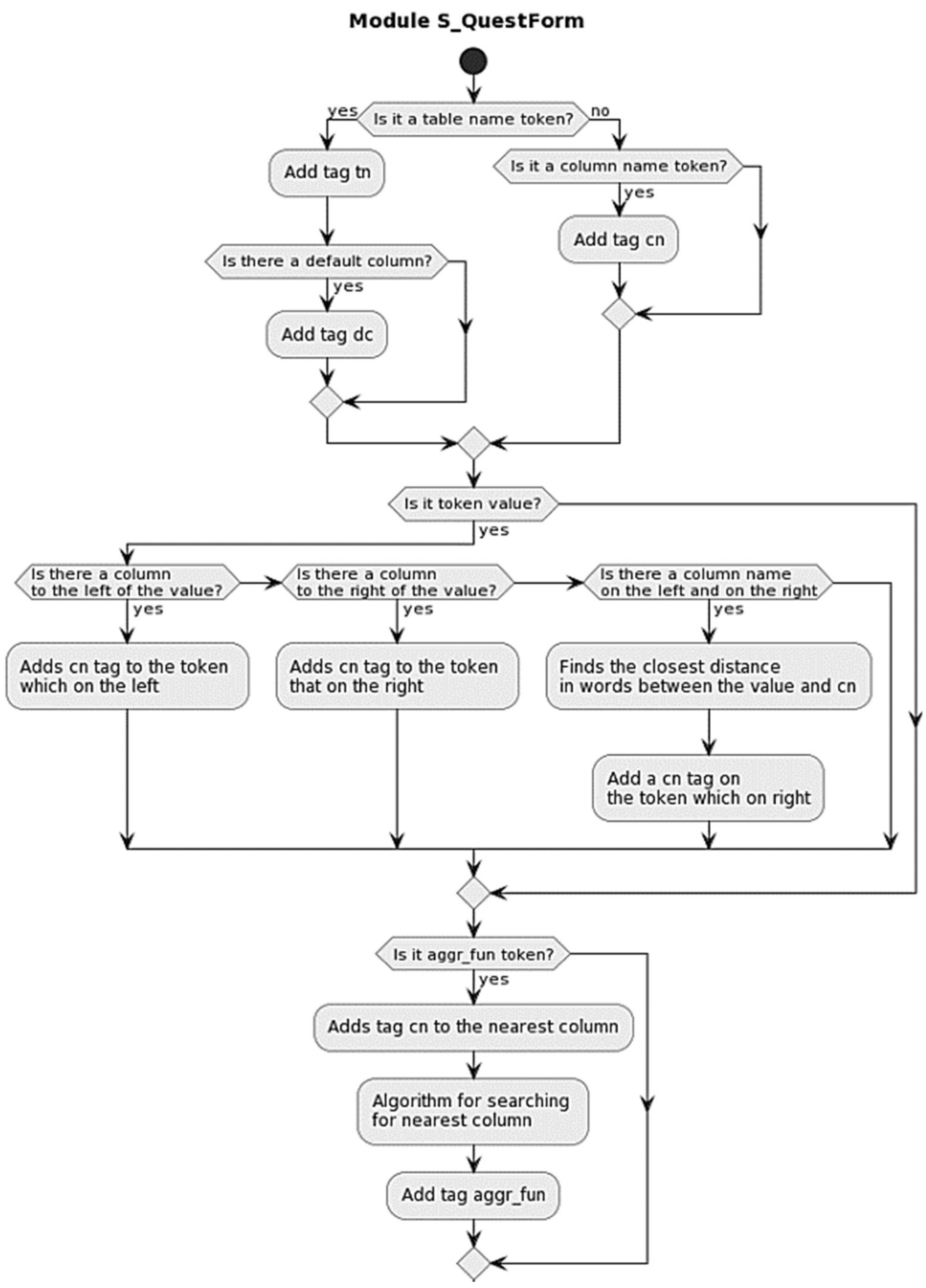

Module S_QuestForm
Is it a table name token?
yes
no
Add tag tn
Is it a column name token?
yes
Add tag cn
Is there a default column?
yes
Add tag dc
Is it token value?
yes
Is there a column to the left of the value?
yes
Is there a column to the right of the value?
yes
Is there a column name on the left and on the right
yes
Adds cn tag to the token which on the left
Adds cn tag to the token that on the right
Finds the closest distance in words between the value and cn
Add a cn tag on the token which on right
Is it aggr_fun token?
yes
Adds tag cn to the nearest column
Algorithm for searching for nearest column
Add tag aggr_fun

Figure 2. Logical scheme of processing in sub module S_QuestForm

3.2.2. Algorithm for building S_GenAnsw sub module

In the **S_GenAnsw sub module**, a final request to the database is formed, corresponding to the structure and elements of the input text and the information encoded in them. In it, the elements building the queries meet the structural templates of requests to the database. The question templates designed in [3] distinguish the form and participants of the input text. This is how different constructions of requests to the database are distinguished.

The sequence of program processing in **sub module S_GenAnsw** is as follows:

Step 1. Check for negation quantifier NOT. The position of the NOT in the sequence of elements is important from the point of view that all values, columns, and functions that are found after it must enter the query after the NOT, i.e. in a negative relation, and in case it is not detected the conditions are satisfied in a positive direction.

Step 2. Forming the dictionaries for all tokens – tn, cn and val. The found table names are stored in the dictionary – table_names[], with the key storing the named objects, and value – the order of the found table names 1, 2,... Before adding a new table, a check is first made to see if it already exists in table_names[];

Step 3. Forming a dictionary for all cn tokens. Similarly, column names are searched for and written to column_names[]. If there is a match, the names of the discovered columns are saved in column_name, and the value recorded in the key has the form tableName.columnName;

Step 4. Forming a dictionary for all tokens val. The names of the received token values are recorded in values_dict, with the data format in key being tableName.columnName, and in value – the value itself. It stores not only the value and its position but also the column the value is on and the table that contains it;

Step 5. Check for relationships between tables. Relational connections are implemented in a hash table. All possible connections, arranged in alphabetical order, form the keys, not repeating keys, a value is an expression that is placed in the body of the join. The key is used as an index to access the values. The names of the tables identified by the input string are in the dictionary table_names[]. It is checked whether there is such a key – alphabetically arranged tables and its value is taken – the expression that enters the Join construction;

Step 6. Forming the From clause from table_names[]. The number of tables in table_names and the relationships between tables based on the relatesColToTable property are influential here. In this case, the expression corresponding to the relation that connects the tables must be added to the query;

Step 7. Checking which of cn are before and after NOT, to form the Where clause;

Step 8. Formation of the Where clause of the request;

Step 9. Check for specific own or aggregate functions;

Step 10. Forming a Select clause from column_names[] and functions (if any).

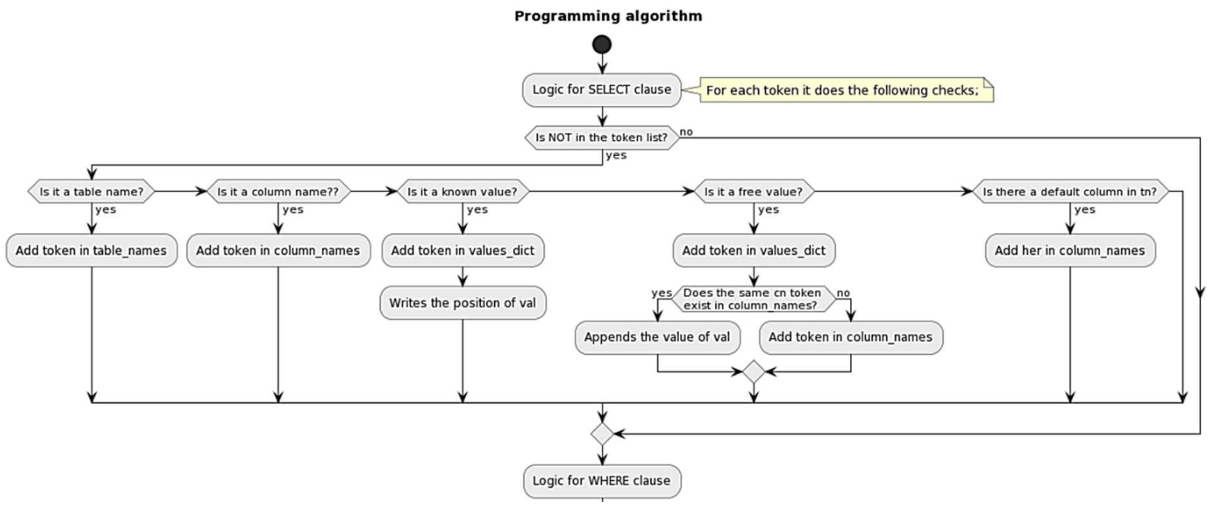


Figure 3. Logic diagram of sub module S_GenAnsw for the synthesis of Select clause

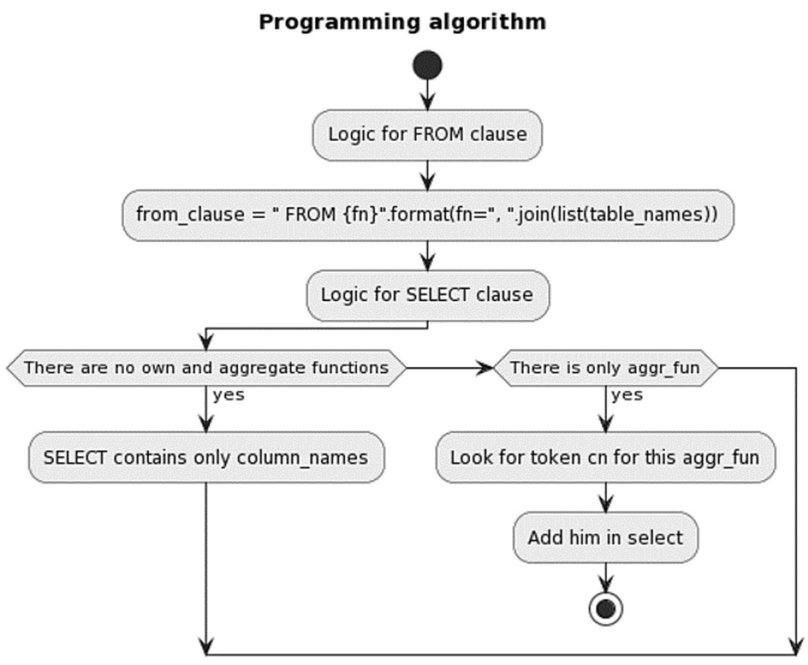

Programming algorithm
Logic for FROM clause
from_clause = " FROM {fn}".format(fn=", ".join(list(table_names))
Logic for SELECT clause
There are no own and aggregate functions
yes
SELECT contains only column_names
There is only aggr_fun
yes
Look for token cn for this aggr_fun
Add him in select

Figure 4. Logic diagram of sub module S_GenAnsw for From clause synthesis

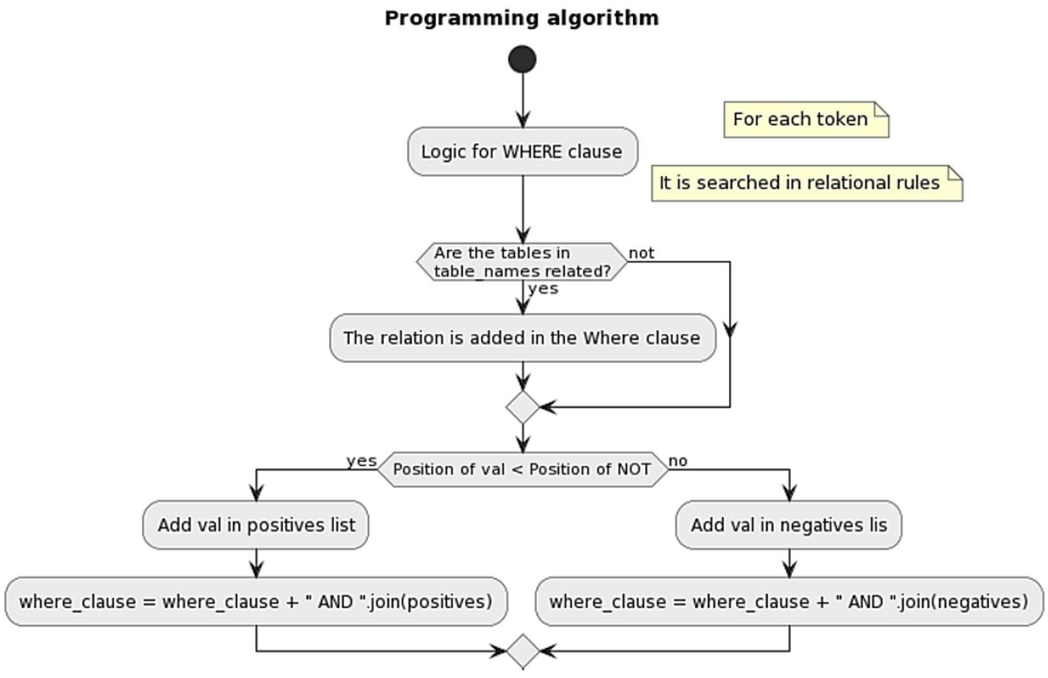

Programming algorithm
Logic for WHERE clause
For each token
It is searched in relational rules
Are the tables in table_names related?
not
yes
The relation is added in the Where clause
yes
Position of val < Position of NOT
no
Add val in positives list
Add val in negatives lis
where_clause = where_clause + " AND ".join(positives)
where_clause = where_clause + " AND ".join(negatives)

Figure 5. Logic diagram of sub module S_GenAnsw for Where clause synthesis

Different strategies have been proposed for forming the body of the **Select clause**, depending on whether there is any built-in and/or own function:

- When functions of type aggr_fun or fun are missing, the select clause contains only a comma-separated list of column names;
- If aggr_fun exists for a given token, it is stored in a list and added to the Select clause.
- If there is an own function for the type of sorting or ratio between two values (order_by / ratio), the type of the next tokens in the sequence should be checked.
  - Order_by checks if the token is last, and if not, checks if the next token is a column name (cn) to sort against.

- With ratio, the next two tokens are checked, which are of type cn and between which a division operation should be performed.

The algorithm for translating NL text to an executable SQL query seeks to solve the task of associating each input element of the input text to a specific object from the database, filling the received parameters into the query model to the database, and retrieving a response from the IS. In the final step, a feedback message is formed to the user about the type and result of a formed request.

4. Results of experiments performed with software tools on this algorithm

The presented idea and algorithm for semantic search and retrieval of information through NLI can be implemented independently in any functioning software system.

Similar tools have already been built in Python [6] and C# [7]. The modules have been tested on two test prototype databases in the field of "Higher Education" – a university database and a prototype database of the National Evaluation and Accreditation Agency of Higher education in Bulgaria.

Many tests have been done with user questions. In some of the conducted experiments, a database query was successfully generated in response to user searches. In another part, logical and technical problems were observed, which were fixed.

During testing, the intermediate results for each specific step are tracked because they lead the debugging process in the right direction. From the intermediate results, conclusions are drawn that change the direction of reasoning and improve the efficiency of the software module.

Let's look at an example of a user question: Number of students majoring in Bulgarian with German. With it, the software module returns the following intermediate results, which are indicative of its accuracy:

- lemmatized text – student number bulgarian_german;
- tagged text – aggr_fun(count) tn(student) val(bulgarian_german);
- tagged text – aggr_fun(count) tn(student) cn(faculty_number) val(bulgarian_german);
- tagged text – aggr_fun(count,cn=faculty_number) tn(student) cn(faculty_number) val(bulgarian_german);
- has_functions – False;
- has_aggregates – True;
- added column – student.faculty_number;
- added column – specialty.specname;
- table_names – student, specialty;
- column_names – student.faculty_number, specialty.specname;
- values_dict – specialty.specname=bulgarian_german;
- ordered_tables_key – specialty student;
- from_clause – FROM student, specialty;
- where_clause – WHERE student.spec_id = specialty.id AND specialty.specname like('%bulgarian_german%');
- negation_pos – inf;
- positives – specialty.specname like('%bulgarian_german%');
- negatives – [];
- final_query – SELECT count(faculty_number) FROM student, specialty WHERE student.spec_id = specialty.id AND specialty.specname like('%bulgarian_german%').

The example demonstrates the intermediate processing in the process of transforming an NL text query into an SQL database query.

The results of the experiments show a 95% success rate that the implemented tools, following the proposed algorithm, cope with the task of transforming natural language text into a query to a relational DB.

5. Conclusion

The article describes the steps to design a programming language-independent software tool that translates a question from natural language to query language and retrieves the information sought in the question. The software tool obtained in this way is part of the general process [1] of creating an NLI and receiving a response from the IS.

The proposed steps for obtaining an answer to a NL user question contribute to the development of systems with a natural language interface. A software solution implemented using the proposed algorithm could improve human-computer communication and is a means of extracting useful information in natural language from an information system database.

6. Acknowledgements

The research is partially supported by the grant No. BG05M2OP001-1.002-0002-C02 “Digitalization of the economy in an environment of Big data”.

7. References

[1] Zhekova M., Totkov G., Pashev G., Methodology for creating natural language interfaces to information systems in a specific domain area, Proc. of the International Conference on Electrical, Computer and Energy Technologies (ICECET 2022), Prague-Czech Republic (Accepted), 6 pages, DOI: 10.1109/ICECET52533.

[2] Affolter K., Stockinger K., Bernstein A, A comparative survey of recent natural language interfaces for databases, The VLDB Journal 28(5), Springer, ISSN: 0949-877X, pp. 793 – 819, DOI: 10.1007/s00778-019-00567-8, October 2019.

[3] [Zhekova M., Totkov G., Question patterns for natural language translation in SQL queries, International Journal on Information Technologies & Security, No 2 (vol. 13), ISSN 1313-8251, pp. 43 – 54, 2021.

[4] Simitsis, A., Koutrika, G., Ioannidis, Y., Précis: from unstructured keywords as queries to structured databases as answers. The VLDB Journal 17, pp. 117–149 (2008), https://doi.org/10.1007/s00778-007-0075-9.

[5] Reynolds, Robert Joshua, Russian natural language processing for computer-assisted language learning: Capturing the benefits of deep morphological analysis in real-life applications, Doctoral thesis, 2016.

[6] Zhekova M., Totkov G., Pashev G., Software Tool for Translation of natural language text to SQL query, InERIS, 14 pages, 2022 (presented).

[7] Zhekova M., Pashev G., Totkov G., Gaftandzhieva S., Automated Extraction of Values of Quantitative Indicators to a Quality Evaluation System Using Natural Language Analysis Tools. Education and Research in the Information Society (ERIS), ISSN 1613-0073, pp. 17-28, 2021.

[8] Zhong, V., Xiong, C., Socher, R., SEQ2SQL: Generating structured queries, 2017, 12 pages, https://doi.org/10.48550/arXiv.1709.00103.

[9] De A., Kopparapu S., A rule-based Short Query Intent Identification System. 2010 International Conference on Signal and Image Processing (ICSIP), DOI: 10.1109/ICSIP.2010.5697471, pp. 212-216.

[10] Melzi S., Jonquet C., Representing NCBO Annotator results in standard RDF with the Annotation Ontology, 7th International Semantic Web Applications and Tools for Life Sciences, vol. 1320 of CEUR Workshop Proceedings, Berlin, Germany, 5 pages, December 2014.

[11] Zeng J., Lin X., Hoi S., Socher R., Xiong C., Lyu M., King I., Photon: A Robust Cross-Domain Text-to-SQL System. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations, 2020, pp. 204–214.