Abstract - A hot issue in the area of database management is to provide a high level interface for non-technical users. An important research direction is the application of natural language interface. The paper presents an interface module that converts user's query given in natural language into a corresponding SQL command. After clustering the input sentence, a pushdown automaton is used to verify the syntax. The corresponding SQL code is generated by a semantic matcher module.

Keywords: NLP, NLI, SQL, formal grammar.

I. INTRODUCTION

One area of research efforts in the query interfaces is focused on improving the usability. The main goal is to provide a high level interface that can be used by non-technical users without any requested DBMS oriented knowledge. An important area in this direction is the application of natural language interface for databases (NLIDB). The NLIDB means that a user can use some natural language to create query expressions and also the answer is presented in the same language. The history of NLIDB goes back as early as 1960's [2]. The era of peak research activity on NLIDB was in the 1980's. In that time, the development of a domain and language independent NLIDB module seemed as a realistic task. The prototype projects showed that the building of a natural language interface is a much more complex task than it was expected.

Regarding the usability of NLIDB, there can be found some tests in the literature that evaluates the efficiency of the NLI interfaces. In these tests the NLIDB is compared with traditional interfaces like SQL [1]. The results show that expert users can perform more efficiently the special command interface (SQL) than the NLI interface [6]. On the other hand, the un-experienced users could achieve better results with the NLI interface than with the imperative SQL interface. A similar result was experienced with the NLI interface for spreadsheet management [7] too.

In the years around the millennium the situation of NLIDB can be characterized on one hand with the decreased interest on theory of general NLIDB (due to the disappointment in the research results to generate a general NLIDB), and on the other hand with the increased number of domain specific commercial products and with the high activity on studying the natural language in general [3]. In the recent years, a lot of new related research areas has arisen and improved. The potential application area of domain specific NLIDB is unlimited. The research projects cover among others the scientific databases (chemistry, biology, geology, mathematics, physics,...), the libraries and the WEB queries.

II. BACKGROUND

The late sixties and early seventies were an active period in database research. The first NLIDB research projects for databases used a domain specific engine like the LUNAR [2] system (1972) that contained data on chemical analysis of moon rocks. In the next decades the number of test systems increased and also the first general NLIDB applications appeared. The RENDEZVOUS (1977) [4] system was one of the first general purpose NLIDB modules. A key element of the developments was to provide database independence (see LADDER [2]) and large flexibility in the grammar’s usage (see ASK [2]).

Based on the success of Chomsky’s transformational language model, the grammar oriented approaches have gained a great importance. Related to the viewpoint of generative linguistics, the most appropriate tools to process the sentences are the declarative logical programming languages. One of the first members of this group is the CHAT-80 [1] project. One of the commercial NLIDB products is the ELF [5] system. It provides an interface to the Access desktop database. The system understands plain-English queries and transforms it into SQL commands. A popular NLIDB interface is the English Query [5] from the Microsoft Its language repository is open, the mapping to the underlying database is generated manually by the developers. Its semantic modeling system stores the relationships between the database objects and the language elements. The natural
language commands are translated into the corresponding SQL commands. Beside the mentioned systems, there are a lot of pilot NLIDB systems like INTELLECT, ASKjeeves or Ianywhere.

Our methodology is related to the current approaches of Giordani [12] and Tikk [13]. In the model of Giordani the sentences are represented by parsing trees. The training pool consists of pair of parsing trees: one tree (NLT) for the sentence in natural language, the other one (SQT) is for the sentence in SQL. There is a knowledge base to store the relationships between the nodes of NLT and SQT. For a new input NL sentence, a similar NLT is searched from the knowledge base. To measure the syntactic similarity between the pairs of trees tree kernel structures are used which computes the number of common substructures. The significance of work [13] is that it creates an efficient NL module for the Hungarian language. The system accepts only simple, well-formed interrogative sentences with a question word from a given list. The engine incorporates several sub-modules to perform a deeper analysis of the sentences. The morphological parser identifies the multi-word tokens of the input sentence and assigns part of speech labels to the tokens. The recognition of multi-word tokens is performed based on decreasing size of expression. The second part of the NL module groups related tokens in brackets. The context recognizer gets bracketed sentence alternatives as input. This module generates SQL like Context Language sentence alternatives. The main information elements during the context recognition are the attribute names, entity names type of entities, verbs used in the query and attribute values.

These approaches show the importance of two base components: first, a deep linguistic and morphologic analysis is required in the case of Hungarian language and second, the similarity based schema matching is an effective way to reduce the computational costs of the engine.

III. GRAMMAR MODEL

The test language of the investigation is the Hungarian language which has a very complex grammar. The Hungarian language belongs to the family of agglomerative languages, where a stem word can be extended with several suffixes. During the joining of suffixes the chaining of tags may cause inflection of the root part. For example, the word

kutyáimmal

can be translated into the following expression: with my dogs,

where

kutyá: stem(dog),
kutyá-im: plural + genitive (my dogs),
kutyá-im-mal: preposition (with).

The second difficulty of the target language is the free word order, the ordering of the words within a sentence has only few constraints. The sentences

Én olvasok egy könyvet a szobában,
Olvasok egy könyvet a szobában,
Egy könyvet olvasok a szobában,
Könyvet olvasok a szobában,
Egy könyvet olvasok én a szobában,
A szobában olvasok egy könyvet,
A szobában én olvasok egy könyvet,
A szobában könyvet olvasok,
A szobában egy könyvet olvasok

are all grammatically correct and have only slight differences in the meaning (I am reading a book in the room).

Chomsky introduced four types of formal grammars in terms of their generative power known as Chomsky-hierarchy. A hotly contested issue over several decades has been the question where natural languages are located within this hierarchy. Chomsky showed [8] that NLs are not regular and he also presumed that NLs are not context-free. On the other hand, context sensitive languages are not adequate for practical use, as they can take up to exponential time to simulate on computers. Thus, the most approaches are based on grammar between context-free and context-sensitive levels. The traditional grammar formalism like TAG [9], HMM [10] are usually effective for languages with strict word ordering and with low set of acceptable words, but they are inefficient for larger size problems. The grammars like dependency grammar[10] or word grammar are strong on handling flexible structure but their implementation details are not well explored yet.

To cope with the complexity problem, the probabilistic context free grammar was selected. A context-free grammar G=(A,V,P) over an alphabet of terminals A is composed of a finite alphabet V of nonterminals and a finite set P ⊆ V ∉ V × (V∪A)* of production rules. The production rules are given in the form u → v where u is nonterminal symbol and v is a sequence of terminal and nonterminal symbols. The context-free grammar can be represented with a pushdown automaton. The push-down automaton is based on the LIFO processing model and has the following formal description:

P(Q,S,G,P,q,F),

where

Q: set of states
S: the alphabet of the language
G: the alphabet of the automaton
P: set of transition rules
q: initial state
F: final states.

At each phase of the sentence processing, the state of the
automaton is given with a triplet \((w,q,s)\), where \(w\): the input sequence to be processed, \(q\): state of the automaton, \(s\): content of the stack.

If for a given \(v\) terminal symbol several production rules exist, the model is called probabilistic CGF model (PCGF). The main benefits of PCGF model is that it can be learned from positive data alone and it provides a robust grammar. Although the averaged entropy related to the PCGF model is higher than of n-gram models, a combination of PCGF and HMM models should superior to the traditional models [11].

IV. CONVERSION OF NL INTO SQL

The NLIDB module has the task to convert a command given in natural language into SQL statements. This transformation is done usually in several distinct steps. The main components of the module are [3] shown in Fig 1.

The main goal of the engine is to convert the user’s input given in natural language into an SQL command. The conversion usually based on four different base repositories:
- language dependent grammar base,
- domain specific semantic repository,
- database specific semantic repository,
- SQL specific grammar base.

The conversion engine consists of four main modules to perform the conversion steps. The first module takes the user’s input sentence and converts it into a sentence of the controlled language. The second module is for the checking the this generated sentence. The elements of the syntactically correct sentences are mapped into the concepts of the database domain in the third conversion module. The fourth module generates the SQL command from the semantic description.

The main module of the conversion engine performs a syntax checking of the incoming sentence. The syntax checking is based primary on the PCFG grammar. As the grammar tree of the full language is too complex, the full grammar can not be involved into the parser module. In order to cope with the complexity problem, the module involves only the grammar of a controlled Hungarian language. The restriction is based on the following elements:
- limited word pool,
- restricted ordering of words,
- limited inflection.

The PCFG grammar is stored in a normalized Chomsky format using the XML standard. The Chomsky normal form means that the right side of the production rule consists of only one or two symbols. The grammar is stored in a grammar tree where the parsing of sentence uses a top-down and left-to-right traversing of the tree. The stack stores the path to the current node under investigation. A rule node has a form
\[ v \rightarrow w^* \]
where \(w^*\) expression can contain some wildcard symbols to define
- type of inflection
- type of stem
- type of matching

For example, the rule
\[ (1,"FBN","FN\{NOM\} FN|NM\{NOM\}","2") \]
has the following meaning:
- \(FBN\) is terminal symbol
- It should match either to \(FN\{NOM\}\) or to \(FN|NM\{NOM\}\)
- The internal checking routine with id \#2 should be called for extra constraint validation
- \(FN|NM\{NOM\}\) means that the stem is noun or pronoun and is in nominative case.

The PCFG parser module is based on the word stemmer module. The Humorph parser is used to determine the stem part and the different inflection components for a given input word. For example, for the input word ‘fizetése’ (his salary), the following output is generated:
\[ fizetés\{FN\}+e\{PSe3\} + \{NOM\}. \]
The list of stems and morphemes can be used to determine the semantic roles of a given word.

As the applied PCFG repository describes only a subpart of grammatically and semantically valid sentences, the incoming sentences should first converted into controlled format. The mapping is based on a clustering approach. The cluster centers are sentence schemas where each schema is a parameterized sentence. The rules have the general form
\[ s \rightarrow s' \]
where \(s\) is a normal parameterized input sentence and \(s'\) is the parameterized sentence of the controlled language. Let
us take the following sample:

"hogyán nevezik #1#E[ACC][PL] nevezzük\[DET]\[ACC]\[PL]\n" \(\rightarrow\) "kérem a #1#tanárók\[ACC]\# nevét\"

In this sentence, the input sentence should consist of four words:

- first word: fix word ‘hogyán’,
- second word: fix word ‘nevezik’ or ‘nevezzük’
- third word: determinant ,
- fourth word: #1#E[ACC][PL]: a parameter with id number 1, it should be of type E (entity name) and it is in a plural and accusative case.

The output sentence consists of four words, where the # separator symbol denotes the parameter substitution. The substitution expression may contain some additional inflection rules and a default value too. Taking the input sentence:

*Hogyan nevezik a tanárokat?*

is converted into the output sentence

*Kérem a tanárok nevét.*

Having the sentence of the controlled language, the sentence elements will be mapped to the concepts of database domains. There are several tables for semantic level mapping:

- synonymms for the database concepts
- synonymms for the relationships
- relationship between the question words and database concepts
- relationship between basic question sentences and database concepts

The mapping for question words is given in the form

\[ w \rightarrow (d,w') \]

where w is a question word, d is the domain of interpretation and w’ is a list of substitution concepts. For example, in the rule

"mi\[ACC]\", "TANTARGYAK\", "TARGYNEV"

the word ‘tantargyak’ denotes a table name (a domain) and the word ‘targynev’ is a fieldname (a concept name). The word ‘mi’ denotes a question word (what).

The SQL command generator application is developed in Java. The input of the program is the NL sentence, and there are output fields for the sentence of the controlled language and for the generated SQL command. The developed SQL generator program can be used for several purposes. First, it can be used as a module in a e-learning tool to train the SQL commands. The second application area is the intelligent database query interfaces for non-technical users. In domains like tourism, public transport ad-hoc and flexible queries should be supported.

In the current prototype system, the domain independent and domain specific repositories are all generated on manual way. This is a major restriction regarding the extension of the method to larger domains. In order to cope with this efficiency limits, the next phase of the project focuses on automated repository generation from external ontology databases.

**CONCLUSION**

In this paper, some results on development of an NLP engine for transforming natural language sentences into SQL commands were presented. The novelty of the approach relates to combination of the following characteristics: processing of the Hungarian language, multi-level stages of command generation and similarity based sentence processing. The generated system provides a flexible and efficient commend generation for a predefined application domain.

**REFERENCES**


