

NLP using database context

Zheni Mincheva¹, Nikola Vasilev¹, Dr. Anatoliy Antonov¹, Dr. Ventsislav Nikolov¹

¹*Eurorisk Systems Ltd, 31 General Kiselov Street, 9002 Varna, Bulgaria*

jmincheva at eurorisksystems dot com, nvasilev at eurorisksystems dot com, antonov at eurorisksystems dot com, vnikolov at eurorisksystems dot com

Abstract

The usage of natural language in the industry has become more prevalent in recent years. Nowadays it is much easier to operate with complex infrastructures using natural language. Feature semantic parsing represents one of the tasks of converting natural language utterances into structured logical parts that can be used as queries to generate responses. This paper introduces an algorithm that transforms natural sentences in order to obtain structural results. Such a functionality is effective for Question and Answering (Q&A) and allows for spoken language understanding (SLU). Queries are being executed on specific tables or databases. The process of generating queries implies different operational steps, such as recognition of different types of words, synonym detection, feature grammar parsing, etc.

Introduction

Problem definition

Nowadays, more and more systems are introducing speech assistants. It's commonplace for people in this day and age to use voice commands, i.e. make calls, schedule appointments, change settings, options and access information using simple everyday words of their natural language. For that reason, it is necessary to simplify modern technology and build tools that make it possible to access large amounts of information easily and everywhere. With the evolution of the industry, more and more tools will be replaced with simpler ones. And what could be simpler than a sentence? This paper proposes an approach that automatically translates natural language into SQL syntax queries. In this way, users will be able to obtain the required information from database tables without the nuisance of technical details. The overview of this approach is shown in Figure 1.

The suggested solution is intuitive and automatic. It is easily accessible and can be applied to any context. The only input information required is an OLAP table. All other details and specifications

are generated automatically from the context of the provided table.

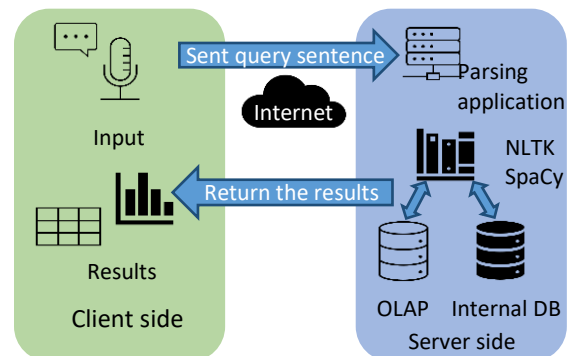


Figure 1. Overview of the approach

NLP explanation

Natural language processing (NLP) combines linguistics and computer science. It is related to the interaction between the natural language of humans and computers. This paper observes the request for information, using limitations and clauses for the grouping and ordering of results. The request is given in a natural language and is being translated into an operation understandable to computers. Currently, this request can be obtained from voice commands or written sentences.

Technical environment

Python

The majority of modern solutions regarding natural language processing are created using Python. Python is high-level programming language that supports a variety of well-developed and widely used frameworks for language processing, including spaCy, NLTK and others. The solution presented in this paper is based primarily on Python 3.7, along with other Python libraries described below.

Natural Language ToolKit (NLTK)

Even though Python can perform natural language processing tasks on its own, NTKL is a good extension that enables the execution of any type of NLP tasks. NLTK contains various modules and has an open-source license. Additionally, it offers a functionality for importing grammar and using it to parse text. [1]

SpaCy

SpaCy is an open-source library for advanced natural language processing. There is a functionality that perfectly defines numbers, time periods and exact dates given in conversational form. For example, it can detect “last year” as a time period and 29th of September as a date.

Structured Query Language – SQL

SQL is a standard language for dealing with databases and is used in programming and for managing data in relational databases. SQL has specific syntax and grammar. The purpose of the proposed application is to automatically translate sentences as expressed by humans into queries in SQL that are understandable to computers.

Intent recognition with RASA

The first step in this process is intent recognition. This is important because the assistant can be connected to more than one OLAP table. Since those tables include information for different contexts, intent recognition is used to determine which table to access. RASA is a great tool for intent recognition. In order for intent recognition

to function properly, it is necessary to train the RASA neural network. This is achieved by giving it examples for each intent. After being trained, the network is able to calculate a percentage for the probability of an input having the encounter intents. Thereafter, an intent is recognized and the input can be used in the key replacement phase. [2]

Observed data

The OLAP table is a flat virtual Cartesian table created from other real tables that are linked together and is used by OLAP reporting systems, such as QlikView and QlikSense. OLAP tables contain two types of data – measures and dimensional data. Dimensional values are usually enumerable and are used for grouping and filtering. Measures can be represented in the form of dates or numerical values. Numerical measures are generally used for aggregation and conditions. Date measures are used for the representation of date points or periods. A data scientist must first define an OLAP table (or use an existing one) using the corresponding format in order to proceed, which is fundamental to the data analysis. OLAP tables provide fast access to large data sets, as all analytical operations are performed on already fetched data and operations are completed in the memory. There is no need for all data to be pre-fetched from the database, providing operations “on the fly”. The module also requires a configuration that includes the selected columns and synonyms for the column.

Methodology

Pre-processing

Date parsing

Dates are complicated data structures for language processing. In the presented module, dates are differentiated into two types – an exact date that specifies a particular date point, and a date range, representing a time period. Dates can be expressed in two forms – implicit and explicit. The implicit form is a self-evident format, e.g. ‘27 of April’. The explicit form defines more complex structures, such as ‘2 years ago from last

Sunday’ or ‘for the last four years’. All dates are masked with specific keywords that can easily be recognized later in the grammar parsing. Additionally, dates are stored into a common format in order to avoid errors and be more general.

Number replacement

In contrast with dates, numbers are simpler for recognition. All numbers are normalized according to the same format and masked with a specific keyword, i.e. „NUMBER”. Later, the masked values are obtained and used to form an SQL query.

Key word replacement

Key words are used in the sentence when asking the system for information. Those words do not cover stop words, which will be considered later. Key words consist of five word sets. The first includes synonyms for column names. When a word from this set is detected in the input and is known, it is referred to the corresponding column. The second set of words contains all distinct values from every enumeration column within the OLAP table. The third set is used for all words that express comparison, e.g. bigger, higher, smaller, less. The last two sets contain synonyms for grouping and ordering, which represent very important parts of the query, as they prioritize information according to the user criteria.

Query parsing

After the key words are loaded, each word from the input is being searched for in the sets and, if found, is replaced with a corresponding word that provides information on the word meaning. For example, all column synonyms are replaced with the key word that contains the column name and has a “SYNONYM” suffix. If a distinct value is found in the input, it is replaced with the corresponding column name. For instance, the input ‘*French*’ would be replaced with

region_value, because it is a distinct value from the column region.

Removing of stop words

The SpaCylibrary provides a list of stop words that include words that have no meaning to the application, such as ‘*also*’, ‘*the*’, ‘*to*’, ‘*are*’, *etc.* Such words are removed. All words that are left unmapped and do not represent stop words are listed as incorrect words. Incorrect words are processed by the edit distance algorithm [3]. It is assumed that those words are not recognized by the microphone or misspelled if written in a query.

Edit distance algorithm for incorrect words

The edit distance algorithm generates the probability of a word being confused with another word or combination of letters similar to the word in question. The algorithm calculates the percentage difference between the words, based on misplaced letters. It takes into account the closeness of the syllables, of letters, as well as the position of keyboard keys. The algorithm distributes negative points according to the error made and depending on what must be done to make the necessary corrections to match it to the desired word. The final percentage is calculated using those points. [4] [5]

In this particular case, three threshold values can be distinguished.

The first is used for an automatic replacement of words. This value is the highest. If the value of similarity between the words is higher than this automatic replacement threshold, the word is replaced without human interaction. The formula (1) for this calculation is

+

$$T_1 = 1 - \frac{1}{l} \quad (1)$$

where T_1 is the threshold value and l is the number of characters in the incorrect word. If T_1 is bigger than 0.92, it is set to 0.92. If it is lower than 0.75, it is set to 0.75.

The second threshold value limits the possible choices presented to the user when choosing the correct word. It is calculated using the following formula (2):

$$T_2 = 1 - \frac{3}{l} \quad (2)$$

where T_2 is the threshold value and l the number of characters in the incorrect word. If T_2 is higher than 0.75, it is set to 0.75. If it is lower than 0.5, it is set to 0.5.

All words with a similarity bigger than T_2 and smaller than T_1 are compiled into a list and presented to the user to choose from them.

The third threshold value is 0.25. If the similarity between the incorrect word and any of the distinct and key values is lower than 0.25, the word is marked as irrelevant and is removed.

Values 0.92, 0.75, 0.5 and 0.25 are preferred and are chosen because of the high accuracy detected during the testing process.

Grammar Parsing

After the pre-processing, the query is parsed and words from the input sentence are replaced with key words that are recognized by the grammar. Parsing of queries is a process of analyzing the sequence of key words in a sentence, that have met a previously defined set of rules. Those rules define the parsing grammar.

Since each SQL select query is composed of separate parts, the input should contain structured information that can be mapped to any of those parts. The grammar should not only recognize them, but take into account the natural language order.

The main structure of grammar rules is disintegrated into smaller rules that are reusable. Each rule is explained later.

Sentence ->
Sentence OrderingPart | ColumnPart
WherePart GroupPart | ColumnPart |
ColumnPart WherePart | ColumnPart GroupPart

The main part of an SQL select query is the column part, in which the required columns are enumerated. It has the following syntax: “SELECT revenue, revision_year, name, ect”. This is the only required part of the query. All other parts are optional. Generally, columns are described using synonyms. If a user wants to see specific information from columns, he uses one of the corresponding synonyms. Synonyms are obtained from sources studying language synonyms and are carefully prepared by people that are familiar with the natural language and the database (data scientist). **ColumnNames** list synonyms for all columns.

ColumnPart -> AllWord | ColumnNames

The **where** part describes the conditions in the WHERE statement. Users can ask for data in several ways using conditions. Since OLAP tables have two main column types, the grammar depends on those types. The *where* part points to each clause, that describes a request, to a column. In the end, the *recursion* part is added, which is a very important and elegant way of calling more than one condition, in any particular order.

The *where* part contains restrictions and filters that specify the requested information. In other words, it describes the conditions in the WHERE statement, which is the most complex part. It contains rules, combined into different parts, for each column of the OLAP table. Those parts are generated automatically using a supportive database. Such database contains descriptions of different columns and defines the way the grammar is supposed to operate with them. Since OLAP tables have two main column types, the grammar depends on those types. The *where* part points each query, that describes a filtering request, to a column. For each column type there are several approaches for requesting information.

The statements are pre-processed and specific parts are replaced by others that are recognizable

by the grammar. Following is an example of the description of a measurement type column. Examples for the covered cases are “*revenue is less than 100*” or “*revenue is 50 or more*”. This part is also generated.

```
revenueClause ->
revenueSynonyms Comparative Number |
revenueSynonyms Number Comparative
```

Measurement conditions use comparative words. Comparatives are loaded from the same database as the synonyms. A lot of words are replaced in the pre-processing. They are replaced with one of the four possible comparatives:

- ‘positive’ are clauses for words ‘greater’ (*more than, above, greater, etc.*)
- ‘negative’ are keywords that contain most of the words for ‘lower’ (*less than, to, behind, etc.*)
- ‘equality’ (is equal to) ‘
- difference’ (is different than, is not)

```
Comparative[SEM='COMPARATIVE'] ->
'positive' | 'negative' | 'equal' |
'different'
```

Dates represent measures, but the syntax for their query is somewhat different. Sometimes comparative words might be omitted and their meaning then refers to an exact point. In other cases, ‘greater’ (*positive*) or ‘smaller’ (*negative*) synonym words are predefined.

```
revisionDateClause ->
revisionDateSynonyms Comparative Date |
revisionDateSynonyms Date
```

In the pre-processing phase of the input text, all numbers are replaced with this word so they can be parsed in the grammar. Input numbers are stored as attributes for later use in the final query.

```
Number[SEM='NUMBER'] -> 'NUMBER'
```

In the same manner as numbers, dates are normalized and replaced by keywords in order to

avoid the mismatching of formats, as well as to benefit the lightweight syntax of the grammar. Dates are stored as attributes for later use in the final query. In this case, the UTC format will be used for the definition of the query.

```
Date[SEM='DATE'] -> 'DATE'
```

If the referred column is from a dimensional type, e.g. region, then it is limited by the following rule in the grammar:

```
regionClause ->
regionNames regionSynonyms |
AllWord regionSynonyms | regionNames
```

Following is an example of the description of an enumerable column. It illustrates the case where the value “all regions” is selected, or any enumeration of column values, with or without a definitive word. The covered cases are:

- *regionNames regionSynonyms* most commonly used, followed by a synonym, e.g. “Italian region”
- *AllWord regionSynonyms* this is a case when all values are taken into account, e.g. “all regions”
- *regionNames* only the value of the given distinct value is considered, e.g. “Italy”

The above explained rules refer to all columns, depending on their type. The defined rules are added to the *where* clause, thereby defining the limitations of the query.

The *where* part is followed by the *grouping* part, after which comes the *ordering* part. *Grouping* is most commonly used for columns that are defined utilizing the *dimensional* type. In cases where the information is hard to assimilate because of its magnitude, the *grouping* part uses a combination of rows. Usually, when there is a *grouping* part, there is an aggregation in the *column* part that is generated automatically. All columns containing the *measure* type are

aggregated using the sum function, except for columns that contain dates. Since the sum of dates is not in use, the depicted information shows the minimum and maximum value of the column. In the grammar, the grouping part is presented as:

```
GroupPart -> GroupSynonyms ColumnNames
GroupSynonyms[SEM='GROUPING'] -> 'GROUPING'
```

The *ordering* part is the last part and refers to the ORDER BY clause in the query, which allows the sorting of result set by one or more columns. Two sorting options are available: ascending and descending. These options are optional. By default, the sorting is in the ascending order. In the grammar, this part has the following syntax:

```
OrderingPart ->
OrderingSynonyms ColumnNames |
OrderingPart OrderingStyle |
OrderingStyle OrderingPart

OrderingStyle[SEM='ORDERING_STYLE'] ->
'ASC' | 'DESC'

OrderingSynonyms[SEM='ORDERING'] ->
'ORDERING'
```

In this context, the grammar can be automatically generated. Since the information is retrieved from an OLAP table, this makes it easier to generate clear grammar.

Creating the Table Query

Once the parsing of the input using grammar is performed, it is certain that the input is convertible to a data base query. The output is represented in the form of a tree which, when iterated, provides a query of a size equal to the input query size. It contains labels of corresponding positions that describe the semantics of the word for this position in the input query. Semantics and inputs are used in the making of the query. This is achieved by extracting the recognized words and placing them in an output query in the proper construction, containing, for example “SELECT WHERE”.

In the cases described above, generated queries will have a syntax as shown below:

*What is the **revenue** and the number of **employees** of companies **Lenovo** and **Sony** in the Dutch and French region, grouped by **region**?*

Common SQL syntax query:

```
SELECT revenue, employees_num,
company_name, region
FROM OLAP_TABLE
WHERE (company_name = 'sony' OR com
pany_name = 'lenovo')
AND (region = 'italian'
OR region = 'french')
GROUP BY region
```

Figure 2. Example of a chart of the query results.

Result can be illustrated in the form of:

- Tables

revenue	employees_num	company_name	Region
....	sony	French
....

- Text (suitable for single row results)

revenue = and employees number =

*Show me **region** and **company** for all companies with revenues bigger than 100*

Common SQL syntax query:

```
SELECT region, company_name, revenue
FROM OLAP_TABLE
WHERE revenue > 100
```

Here the results can be shown in the form of:

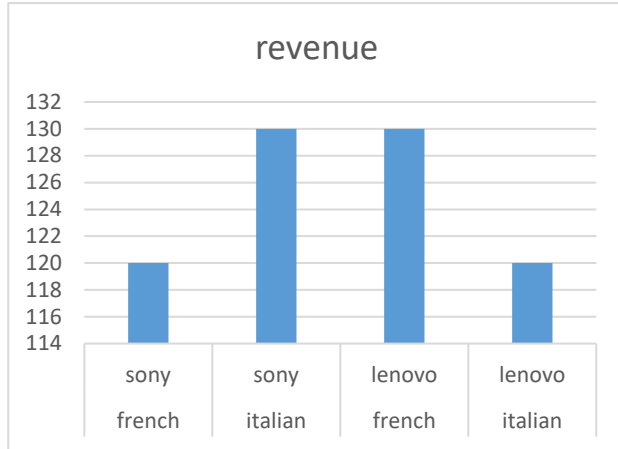
- Tables Table 1

Table 1 Example of the query results.as a table.

region	company_name	revenue
french	sony	120
italian	sony	130
french	lenovo	130

italian	lenovo	120
---------	--------	-----

- Charts (suitable for more complex results) Figure 2



A query is successfully parsed when all words are recognized, and are in order that is acceptable by the grammar.

Figure 3 shows the main steps of the application's algorithm. First, the information is loaded from an OLAP table, which represents the main input for the generation. The distinct values from the OLAP table are loaded and recorded in an internal structure used later in the process. This information, provided in the structure, is used for the generation of the grammar. Once the grammar is generated, it is ready to parse sentences that are obtained from voice or text. Each word in the sentence must be recognized and labeled. The words that are recognized represent distinct values and synonyms, which are added to the input query for more flexibility. Unlabeled words are marked as incorrect and are sent for corrections. After the corrections are made, the words are replaced in the input. Once the sentence is completely corrected, it is parsed and used in the process of generating queries. All words that are found in the input should be recognized and labeled, so the grammar can parse them. Parsing of the grammar represents a process of checking the order of words in a sentence. The order must match the predefined rules of the grammar. Consequently, the query is generated and, if valid, it can be executed and the retrieved results

can be displayed in various formats, such as tables, charts or even voice.

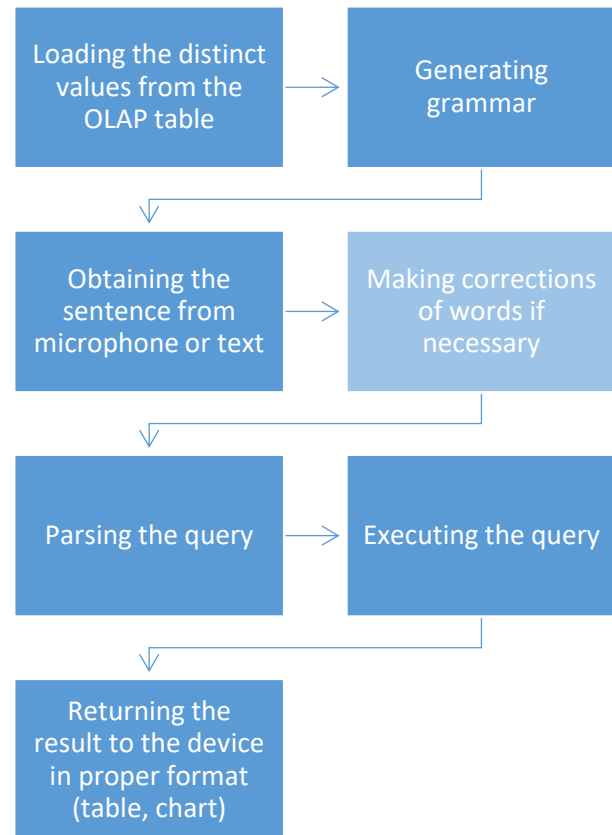


Figure 3. Overview of the steps of the algorithm.

Results

The OLAP table is comprised of five columns, which contain the name of the company, region of the located branch, revenue of the corresponding branch, the revision year and number of employees. Company name and region columns are *dimension type* data, while all others represent *measure types*. The table contains 10000 records.

The internal database contains supporting information that are required for the grammar generation. Figure 4 shows the model of the internal database.

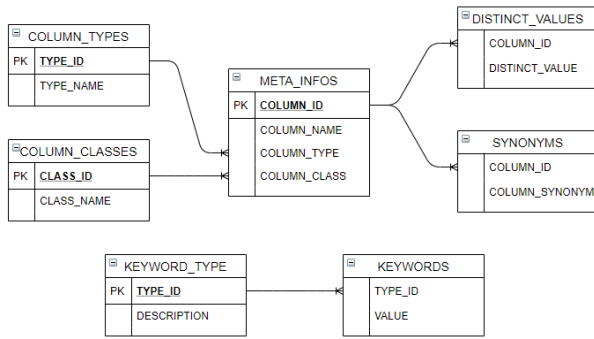


Figure 4. The internal database relational model.

META_INFO includes the properties of each column, and contains the column name, types and classes. Types provide information on the kinds of values that can be found in this column, which are. String, Number and Date. These values are described in the COLUMN_TYPES table. Classes represent types of data that are stored in columns and are classified into dimensions and measures. They are located in the COLUMN_CLASSES table. Table DISTINCT_VALUES contains distinct values from all columns. The SYNONYMS table stores synonyms that can be used to designate columns. There are two additional tables that contain information on key words, such as “more”, “bigger”, “smaller”, “less”, etc. Those key words are used throughout the entire application and are not connected specifically to columns.

In order to test the described algorithm, a Python prototype application is build. It uses a Python Qt5 library for visualization and basic functionalities. The input for the sentence can be in the form of voice or text. After that, the natural sentence is automatically converted to an SQL query. Additionally, an intermediate result of the parsing is provided showing only the meaningful word by removing the stop word and other unnecessary information. Then, if all the parsing and validation stages are successful, an SQL query is displayed and results are shown in table format. Here the given sentence is ‘What is the revision and revenue of Nvidia and Disney in the Italian and British region, ordered by revision’. After the parsing of results, the algorithm provides the

following query, which matches the given criteria:

```

SELECT revision_year, revenue FROM
OLAP_TABLE WHERE ( name = 'nvidia' OR
name = 'disney') AND ( region =
'australian' OR region = 'british')
ORDER BY revision_year
    
```

The application needs to show the input taken from the user, the corrected sentence that is “understood” by the application, the final result of the parsing, as well as results from the query. Figure 5 illustrates a prototype for the desktop application. The algorithm can be used from a desktop application, a web browser or mobile application. [6]

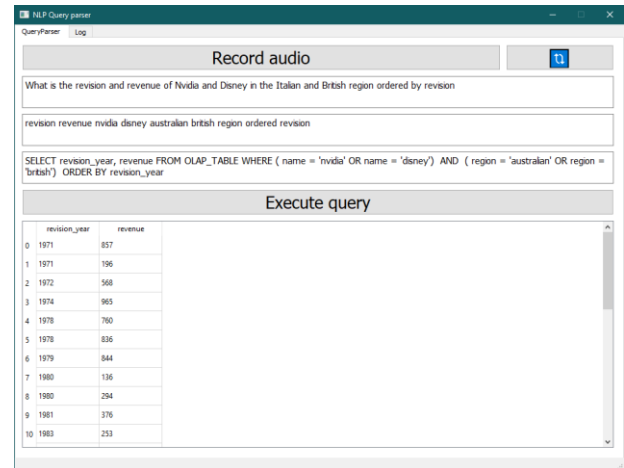


Figure 5. Prototype the application.

However, if the input is misspelled, the application requires an additional interaction with the user in order to correct the incorrect word. Consider the following example: ‘give me names~~s~~ and re~~w~~enues for ciss~~c~~o sorte~~e~~ by region’. This sentence has several errors. The algorithm automatically substitutes some words, such as re~~w~~enues, names~~s~~, sorte~~e~~, since they come very close to their correct counterparts. With others, additional interaction is needed. Figure 6 shows a window in the application, where users can choose from a list of alternative words, according to the given probability of similarity.

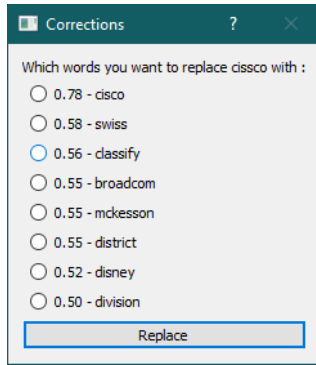


Figure 6. Prototype of the corrections suggestions.

For further details, a log is provided, where the automatic actions and performance benchmarks can be seen. A screenshot of the log is shown in Figure 7 below. All parsing results are performed almost immediately, within less than a second.

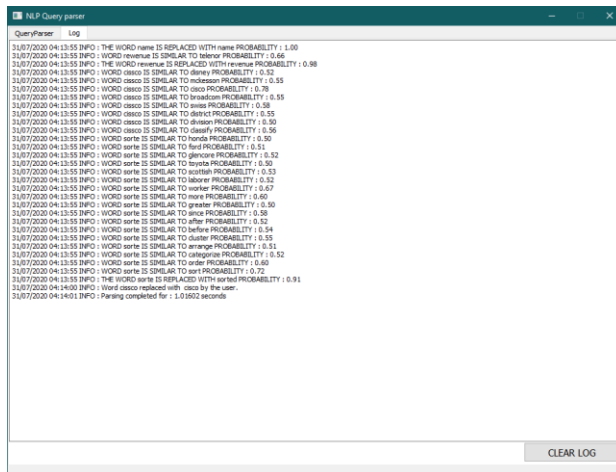


Figure 7. Prototype of the log.

Future developments

The main logic of the application runs on a server, so it can be accessed from anywhere. The proposed solution provides information in a user-friendly and flexible manner. Since smart phones have become such an indispensable part of our lives, it can be made possible for the application to be accessed from mobile devices. It could also be accessed via a web browser or desktop application, depending on the user needs.

Further developments will include multi-language support and interactive conversation,

thereby enabling the usage of aggregation functions, such as sum, average, max and min, as well as ordering and chart type selection of the result table for the primary data request.

Conclusion

Natural language represents an easy way to obtain desired information without the necessity of spending additional time to learn the required technical details. The solution presented in this paper can be used in different contexts. For example, obtaining needed information “on the fly”, using only simple sentences.

References

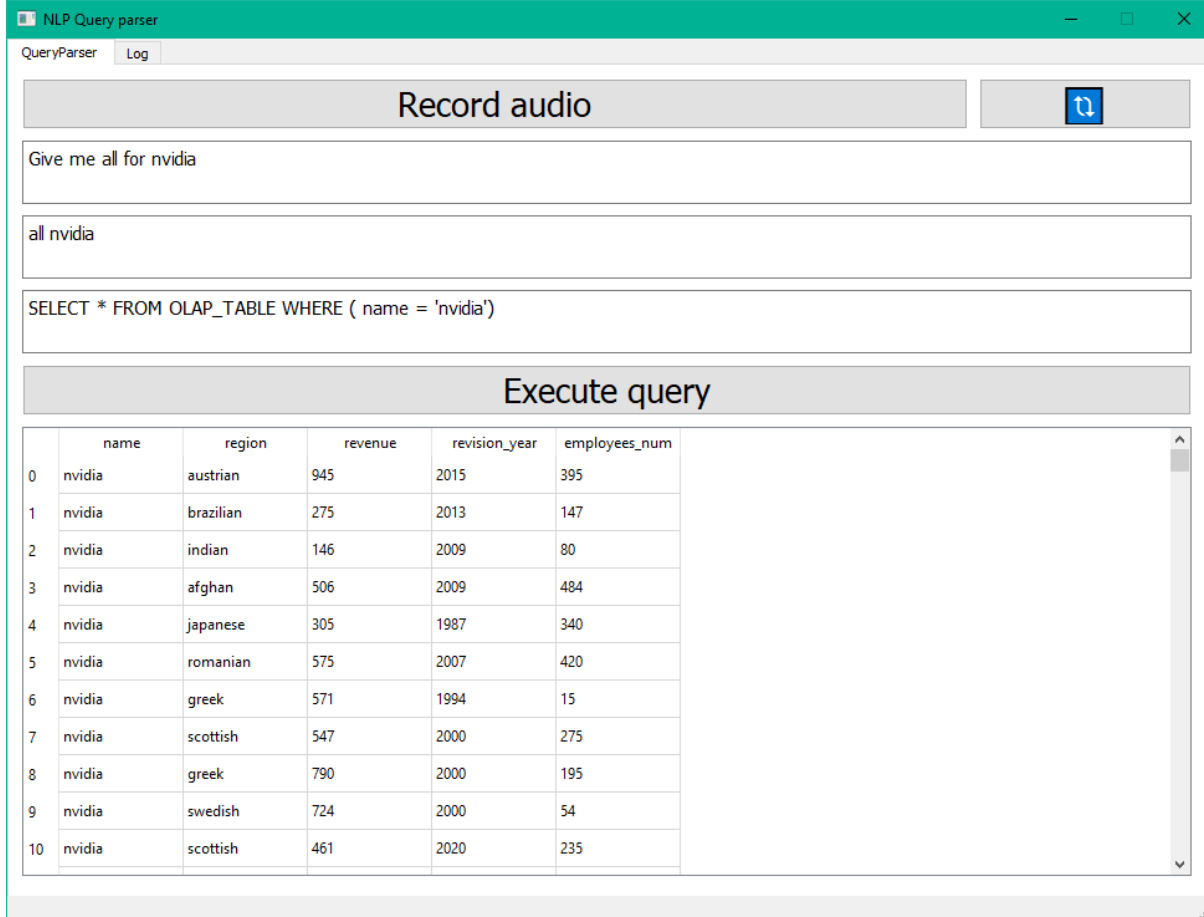
- [1] J. Perkins, Python text processing with NLTK 2.0 cookbook over 80 practical recipes for using Python's NLTK suite of libraries to maximize your natural language processing capabilities, Birmingham, U.K.: Packt Pub., 2010, p. .
- [2] T. Bocklisch, J. Faulkner, N. Pawlowski и A. Nichol, „Rasa: Open Source Language Understanding and Dialogue Management,“ p. .
- [3] A. A. Plamen Paskalev, „Intelligent Application for Duplication Detection,“ *CompSysTech '2006*, 2006.
- [4] P. P. Anatoliy Antonov, „Increasing the performance of an application for duplication detection,“ *CompSysTech '2007*, 2007.
- [5] D. Skurzok и B. Ziółko, „Edit distance comparison confidence measure for speech recognition,“ в *Lecture Notes in Electrical Engineering*, 2013.
- [6] M. Aiello, Y. Yang, Y. Zou и L.-J. Zhang, Artificial Intelligence and Mobile Services – AIMS 2018 7th International Conference, Held as Part of the Services Conference Federation, SCF 2018, Seattle, WA, USA, June 25-30, 2018, Proceedings, 1st ed. 2018.. ред., M. Aiello, Y. Yang, Y. Zou и L. Zhang, Ред., Cham: Springer International Publishing : Imprint: Springer, 2018, p. .

Appendix: Examples

1. Example of the usage of the word 'all' in the column part.

Input: Give me all for Nvidia.

Output: SELECT * FROM OLAP_TABLE WHERE (name = 'nvidia')



The screenshot shows the 'NLP Query parser' application window. It features a 'Record audio' button and a 'Log' tab. The input text is 'Give me all for nvidia'. The generated SQL query is 'SELECT * FROM OLAP_TABLE WHERE (name = 'nvidia')'. Below the query, there is an 'Execute query' button and a table displaying the results.

	name	region	revenue	revision_year	employees_num
0	nvidia	austrian	945	2015	395
1	nvidia	brazilian	275	2013	147
2	nvidia	indian	146	2009	80
3	nvidia	afghan	506	2009	484
4	nvidia	japanese	305	1987	340
5	nvidia	romanian	575	2007	420
6	nvidia	greek	571	1994	15
7	nvidia	scottish	547	2000	275
8	nvidia	greek	790	2000	195
9	nvidia	swedish	724	2000	54
10	nvidia	scottish	461	2020	235

2. Example of the grouping criteria.

Input: Give me all for Nvidia and Lenovo, grouped by company and region.

Output: SELECT name, region, sum(revenue) , min(revision_year), max(revision_year) , sum(employees_num) , count(*) FROM OLAP_TABLE WHERE (name = 'nvidia' OR name = 'lenovo') GROUP BY name, region

NLP Query parser
- □ ×

QueryParser Log

Record audio

↻

Give me all for nvidia and lenovo grouped by company and region

all nvidia lenovo grouped company region

```
SELECT name, region, sum(revenue) , min(revision_year), max(revision_year) , sum(employees_num) , count( * ) FROM OLAP_TABLE
WHERE ( name = 'nvidia' OR name = 'lenovo') GROUP BY name, region
```

Execute query

	name	region	sum(revenue)	min(revision_year)	max(revision_year)	m(employees_num)	count(*)
0	lenovo	afghan	4851	1977	2019	1391	7
1	lenovo	african	3334	1971	2016	2114	11
2	lenovo	algerian	4941	1975	2017	1526	7
3	lenovo	angolan	4104	1973	2020	1340	8
4	lenovo	argentine	1904	1977	2012	1856	5
5	lenovo	asian	4903	1972	2016	2607	9
6	lenovo	australian	4391	1992	2018	1479	7
7	lenovo	austrian	3332	1976	2015	1754	7
8	lenovo	belgian	818	1986	2017	458	3
9	lenovo	bolivian	7256	1973	2020	4057	12
10	lenovo	brazilian	2985	1988	2015	2087	8

3. Example of the ordering criteria.

Input: What is the revenue and revision of Adobe and Cisco in the German and Spanish region, sorted by region firm?

Output: `SELECT revenue, revision_year, region, name FROM OLAP_TABLE WHERE (name = 'adobe' OR name = 'cisco') AND (region = 'german' OR region = 'spanish') ORDER BY region, name`

NLP Query parser
— □ ×

QueryParser
Log

Record audio
🗣️

What is the revenue and revision of adobe and cisco in the german and spanish region order region firm

revenue revision adobe cisco german spanish region order region firm

```
SELECT revenue, revision_year, region, name FROM OLAP_TABLE WHERE ( name = 'adobe' OR name = 'cisco') AND ( region = 'german' OR region = 'spanish') ORDER BY region, name
```

Execute query

	revenue	revision_year	region	name
11	859	1988	german	cisco
12	35	1982	german	cisco
13	960	2010	spanish	adobe
14	559	2018	spanish	adobe
15	805	1980	spanish	adobe
16	299	2018	spanish	adobe
17	975	1975	spanish	adobe
18	694	1976	spanish	adobe
19	566	1998	spanish	adobe
20	66	2017	spanish	cisco
21	171	1980	spanish	cisco

4. Example of the ordering criteria

Input: Give me the revenue region and name for Sony and Nvidia in the Brazilian, Swedish and Swiss region, ordered by revenue in desc.

Output: `SELECT revenue, region, name FROM OLAP_TABLE WHERE (name = 'nvidia') AND (region = 'brazilian' OR region = 'swedish' OR region = 'swiss') ORDER BY revenue DESC`

NLP Query parser
_ □ ×

QueryParser Log

Record audio

Give me the revenue region and name for sony and nvidia in the brazilian swedish and swiss region ordered by revenue in desc

revenue region name company nvidia brazilian swedish swiss region ordered revenue desc

`SELECT revenue, region, name FROM OLAP_TABLE WHERE (name = 'nvidia') AND (region = 'brazilian' OR region = 'swedish' OR region = 'swiss') ORDER BY revenue DESC`

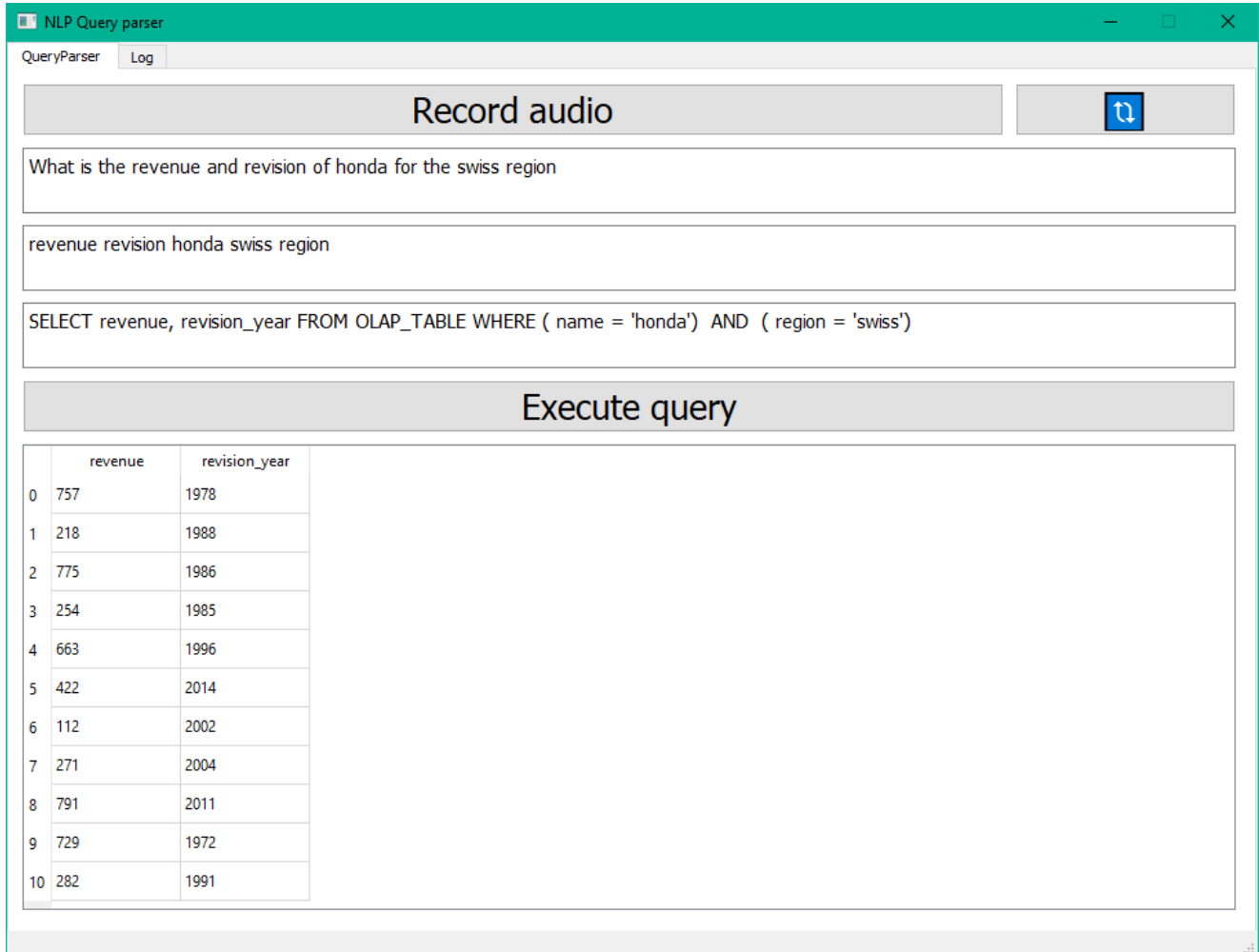
Execute query

	revenue	region	name
0	996	swedish	nvidia
1	969	brazilian	nvidia
2	959	swiss	nvidia
3	904	swedish	nvidia
4	899	brazilian	nvidia
5	888	brazilian	nvidia
6	887	brazilian	nvidia
7	869	swiss	nvidia
8	862	swiss	nvidia
9	855	swedish	nvidia
10	827	swiss	nvidia

5. Examples of asking human-like questions.

Input: What is the revenue and revision of Honda for the Swiss region?

Output: `SELECT revenue, revision_year FROM OLAP_TABLE WHERE (name = 'honda')
AND (region = 'swiss')`



The screenshot shows a web application titled "NLP Query parser". It has a "Log" button in the top left. The main interface is divided into several sections:

- Record audio:** A large grey button with a microphone icon.
- Input:** A text box containing the question: "What is the revenue and revision of honda for the swiss region".
- Processed Input:** A text box showing the extracted keywords: "revenue revision honda swiss region".
- Generated Query:** A text box showing the resulting SQL query: `SELECT revenue, revision_year FROM OLAP_TABLE WHERE (name = 'honda') AND (region = 'swiss')`.
- Execute query:** A large grey button.
- Results Table:** A table with 11 rows and 3 columns. The first column is an index (0-10), the second is "revenue", and the third is "revision_year".

	revenue	revision_year
0	757	1978
1	218	1988
2	775	1986
3	254	1985
4	663	1996
5	422	2014
6	112	2002
7	271	2004
8	791	2011
9	729	1972
10	282	1991