Journal of Software Engineering and Applications, 2019, 12, 365-382
https://www.scirp.org/journal/jsea
ISSN Online: 1945-3124
ISSN Print: 1945-3116

A Shallow Parsing Approach to Natural Language Queries of a Database

Richard Skeggs, Stasha Lauria

College of Engineering, Design and Physical Sciences, Brunel University, London, UK
Email: richard.skeggs@brunel.ac.uk, stasha.lauria@brunel.ac.uk

How to cite this paper: Skeggs, R. and Lauria, S. (2019) A Shallow Parsing Approach to Natural Language Queries of a Database. *Journal of Software Engineering and Applications*, **12**, 365-382.
https://doi.org/10.4236/jsea.2019.129022

Received: June 5, 2019
Accepted: September 26, 2019
Published: September 29, 2019

Abstract

The performance and reliability of converting natural language into structured query language can be problematic in handling nuances that are prevalent in natural language. Relational databases are not designed to understand language nuance, therefore the question why we must handle nuance has to be asked. This paper is looking at an alternative solution for the conversion of a Natural Language Query into a Structured Query Language (SQL) capable of being used to search a relational database. The process uses the natural language concept, Part of Speech to identify words that can be used to identify database tables and table columns. The use of Open NLP based grammar files, as well as additional configuration files, assist in the translation from natural language to query language. Having identified which tables and which columns contain the pertinent data the next step is to create the SQL statement.

Keywords

NLIDB, Natural Language Processing, Database Query, Data Mining

1. Introduction

With the quantity of real-time data and the speed of data increases the need to search and extract data from multiple sources is becoming more important. Natural Language Processing can be useful for converting natural language text into a formal structure that can be processed by a computer program.

The growth in size and importance of data within society has led to the development of a new range of tools to query, examine and analyse data. Even the increasing use of tools like Siri, Bixby, Alexa and Google Assistant to perform searches is changing the way users look for information. With large quantities of data stored within databases or databased backed repositories providing an interface between a non-technical user and data is becoming increasingly important.

The use of natural language interface to a database enables non-technical users to search a database using natural language statements, whether that is the spoken or written word. The Natural Language Interface to Database (NLIDB) provides the interface between a natural query and a structured data query language like SQL. This allows for data retrieval without the need for technical knowledge or a detailed understanding of the Structured Query Language (SQL) or even knowledge of the underlying database.

A number of systems such as LADDER, CHAT-80, NaLIX and WASP [1] have all been developed to become the interface between natural language and the database but none of them have come into mainstream use. The issues these tools have struggled with revolve around natural language complexity. The most common one of these complexities has been understanding language nuance [2] [3] [4]. Other issues have revolved around the performance of the interface in converting the natural language query not only in a timely fashion but also with the accuracy of the returned results [5].

This paper is proposing a solution to solve both the language nuance [3] [6] and performance issues with the use of shallow parsing [7], which does not require an understanding of language nuances. The shallow parsing approach being proposed by this paper is the use of keywords [8] to identify characteristics that are important for the search. This paper will introduce the use of an index file containing keywords that can be used to enhance the performance. Jwalapuram & Mamidi [5] are among a number of authors who have carried out research into using keywords to enable NLIDB based systems to perform searches.

The keyword searching proposed in this paper unlike Jwalapuram & Mamidi [5] uses Part of Speech (POS) [5] processing and an index file which allows for individual words to be extracted from the natural language query. The individually extracted words can then be used to create the query for the NLIDB solution.

2. Football Events Data

To test the performance of the NLIDB application an open data set was selected for benchmarking. The website Kaggle.com has several openly available large datasets that can be used freely. The Football Events dataset was chosen and is available via the following link (https://www.kaggle.com/secareanualin/football-events). The data contains two tables which ensure that the feature to join two tables together can also be tested. The concept of being able to join two or more tables together is important as this feature is often useful when searching data repositories as data can be held across multiple tables.

The dataset comes in the form of two comma separated value (CSV) files which are labelled EVENTS and GINF. The events recorded in the tables cover 9074 football games from across Europe. The two tables are in CSV format which makes it easier to load into a database whether that is a no-SQL or RDBMS version. The EVENTS table as shown in **Table 1** contains details about each game. The data has been scrapped from bbc.com, espn.com and onefootball.com and has 941009 recorded items. The GINF table, details are shown in **Table 2** contains metadata and market betting odds for each game and contains 10,112 entries. The odds for the dataset were supplied by oddsportal.com.

The two tables can be joined using the common key ID_ODSP, which is the unique identifier for the game.

Table 1. The EVENTS table describes the structure of the events database. The details of the event types can be found in **Appendix B**.

Column Name	Data Type	Description
ID_ODSP	String	Unique id of the game
ID_EVENT	String	Unique identifier of event (ID_ODSP + SORT_ORDER)
SORT_ORDER	Number	Chronological sequence of events in a game
Time	Number	Minutes into the match
Text	String	Description of event
EVENT_TYPE	String	Primary event. 11 unique events (1-attempt (shot), 2-corner, 3-foul, 4-yellow card, 5-second yellow card, 6-(straight) red card, 7-substitution, 8-free kick won, 9-offside, 10-hand ball, 11-penalty conceded)
EVENT_TYPE 2	String	Secondary event. 4 unique events (12-key Pass, 13-failed through ball, 14-sending off, 15-own goal)
Side	String	Home or away team (1-home, 2-away)
EVENT_TEAM	String	Team that produced the event (In case of Own goals, event team is the team that beneficiated from the own goal)
Opponent	String	Opposing team
Player	String	Player involved
Player 2	String	Player involved
PLAYER_IN	String	Player that came in (only applies to substitutions)
PLAYER_OUT	String	Player substituted (only applies to substitutions)
SHOT_PLACE	String	Placement of the shot (13 possible placement locations, available in the dictionary, only applies to shots)
SHOT_OUTCOME	String	4 possible outcomes (1-on target, 2-off target, 3-blocked, 4-hit the post)
IS_GOAL	Boolean	binary variable if the shot resulted in a goal (own goals included)
Location	String	Location on the pitch where the event happened (19 possible locations, available in the dictionary)
Body Part	String	Body part ball touches (1-right foot, 2-left foot, 3-head)
ASSIST_METHOD	String	In case of an assisted shot, 5 possible assist methods (details in the dictionary)
Situation	String	In case of an assisted shot, 5 possible assist methods (details in the dictionary)
FAST_BREAK	Boolean	Did a fast break occur

Table 2. The GINF table describes the features of the GINF table.

Column Name	Data Type	Description
ID_ODSP	String	Unique ID of the game
LINK_ODSP	String	Link to odd sportal page
ADV_STATS	Boolean	Availability of advanced statistics
Date	Date	Date of event
League	String	The league the match was played in
Season	Number	The year the season finished
Country	Number	The country the match was played in
Ht	String	Home team
At	String	Way team
Fthg	Number	Full time home goals
Ftag	Number	Full time away goals
ODD_H	Number	Highest home win market odds
ODD_D	Number	Highest draw market odds
ODD_A	Number	Highest away market odds
ODD_OVER	String	Highest over 2.5 market odds
ODD_UNDER	String	Highest under 2.5 market odds
ODD_BTS	String	Highest both teams to score market odds
ODD_BTS_N	String	Highest both teams NOT to score market odds

3. Proposed Configuration

This paper is proposing to use three index files to aid the conversion from natural language query to SQL. The files being proposed are the Grammar file, Join file and Index file. The use of these files ultimately describes the structure of the underlying database which will become the target for searching, while providing an index like data structure that can be used to identify the database table(s) and table columns relevant for the database search.

The files describe in this section can be created either manually or through scripting. The grammar file should be created through the collection of queries that been used to query the underlying database. With a historic record of prior questions, the grammar file can be enhanced.

Figure 1 shows an overview of the proposed architecture for the NLIDB solution being discussed in this paper. The details of which will be expanded in this section, but the first step is to parse the incoming natural language query using the grammar file to identify parts of the query and to be able to tag individual words appropriately. The second step is to translate the natural language into an SQL statement. The join file and the index file contain the information about the database; details of this process are discussed below. The final step is the query itself. Having created the SQL query the next step is to execute the query against the database.

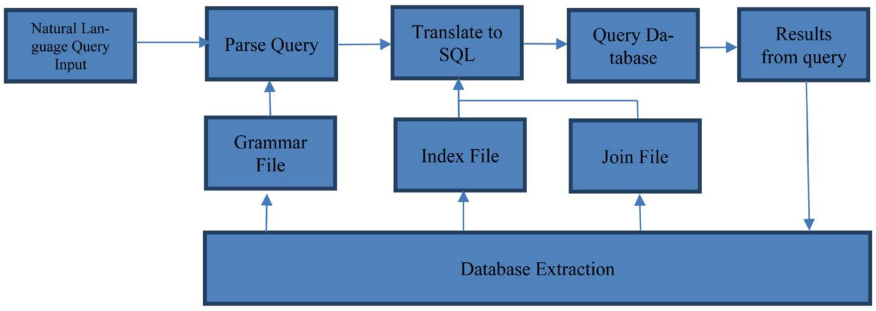


Figure 1. Shows an overview of the proposed system.

3.1. Grammar File

The database extraction process which provides data for the three configuration files manually extracts data from the target database. Thought the process is manual there is nothing about the structure of the configuration files nor the data used by the files which stop their creation from being automatic.

The first of these is the Apache Open NLP [9] grammar file which is used to identify words in the natural language query. The content from the database is used to create the grammar file, column names from the database tables and the database tables are used with the grammar file. Separate tags are assigned to each word which identifies words of importance that can be labelled as either a table name or column name. The convention for tags is that VB identifies a verb, N for noun and ADJ for adjective, a full list of tags can be found in **Appendix A**. The list of tags is used by convention rather than being statically defined, therefore custom tags can be created to fulfil a specific task. This paper uses a custom tag IRR to identify words that are irrelevant in the conversion from natural language to query language. In the example used for this paper, the grammar file is constructed from entries from both the GINF and EVENTS tables. Questions posed to the application are also used as part of the grammar file. Table 3 lists the column names from both source files that are used within the grammar file.

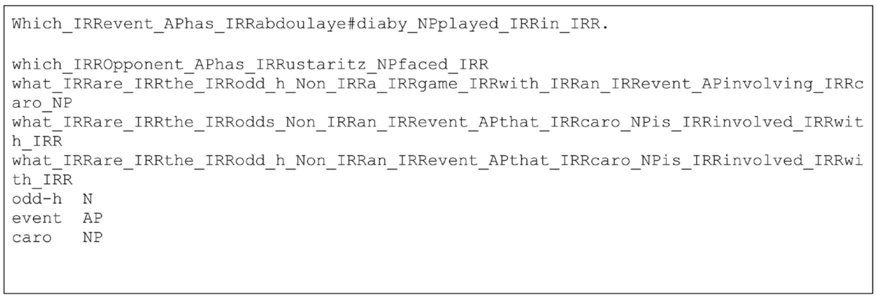
The index data extracted from the GINF table contain 10,643 entries which are made up of the original entries with some additional data. Entries from the Events table create an index file with 1201 unique entries in the data. The structure of the table is made up of potential questions that could be posed to the NLIDB application. Each word is assigned a tag representing how that word should be treated. The tags follow the appropriate word and are separated from it by an underscore.

The grammar file (an extract of which is Figure 2) for this paper uses a couple of tags, IRR which stands for irrelevant and ensures that the word will be ignored in the conversion from natural language to structure query language. The IRR tag is defined as being words or values not found within the underlying database as either table names, columns or values.

NP, which signifies that the word is important in the conversion process and states that is a value of significance and will be used within the search as this is the search criteria. Words tagged with AP signify the table that must be searched.

Table 3. Lists the entries extracted from the database for inclusion into the index file.

Events	GNIF
ID_ODSP	ID_ODSP
Side	Date
EVENT_TEAM	League
Opponent	Season
Player	County
Player 2	Ht
SHOT_PLACE	At
SHOT_OUTCOME	Fthg
	Ftag
	ODD_H
	ODD_D



```
Which_IRRevent_APhas_IRRabdoulaye#diaby_NPplayed_IRRin_IRR.

which_IRROpponent_APhas_IRRustaritz_NPfaced_IRR
what_IRRare_IRRthe_IRRodd_h_Non_IRRa_IRRgame_IRRwith_IRRan_IRRevent_APinvolving_IRRc
aro_NP
what_IRRare_IRRthe_IRRodds_Non_IRRan_IRRevent_APthat_IRRcaro_NPis_IRRinvolved_IRRwit
h_IRR
what_IRRare_IRRthe_IRRodd_h_Non_IRRan_IRRevent_APthat_IRRcaro_NPis_IRRinvolved_IRRwi
th_IRR
odd-h   N
event   AP
caro    NP
```

Figure 2. The table is an extract from the grammar file showing the data structure.

Finally, the tag N defines which column the search criteria could potentially be found in.

3.2. Index File

In addition, the grammar file is the index file. This file is currently created manually but there is nothing within the file that prevents its creation through automated scripts. The file contains elements from the database being searched; an extract from the index file is shown in Figure 3. The data is made up of three columns; the first column shows the relationship between the table, the table column and the database value. The index file uses the same tags as the grammar file to identify elements that are within the database such as the tables, columns and values. Figure 3 shows that the AP tag is assigned to the value event, this represents the table. The second value is player which is assigned the tag N, which represents the column in the table. The third column shows a value in this case the name of a player (Abdoulaye Diaby) which has been assigned the tag NP.

From this, information the query is beginning to be built and simplistically the query is "*select * from event*". The second column describes which variable from the table to use as part of the condition. In the example below, the column

```
event_APplayer_Nabdoulaye#diaby_NP

event_APplayer_Nabdoulaye#faye_NP
event_APplayer_Naboubakar#kamara_NP
event_APplayer_Nadam#federici_NP
event_APplayer_Naitor#cantalapiedra_NP
event_APplayer_Nalberto#garcia_NP
event_APplayer_Naleksandr#iakovenko_NP
event_APplayer_Nalende_NP
event_APplayer_Nalessandro#bastrini_NP
```

Figure 3. Extract from the index file.

is "*player*". This now means that the query is "*select * from event where player =*". The only element missing is the value to search on or in this case the player's name. This information comes from the third column labelled NP. From the extract in Figure 2, there is an extract of abdoulaye#diaby_NP, so the final query is now "*select * from event where player* = '*abdoulayediaby*'".

3.3. Join File

The above example shows the first step into parsing a natural language query into simple SQL statement. Not all queries are that simplistic as some will require that tables are joined to extract the required data. A key aspect is how the joins between tables can be identified not just from the natural language query but also from the table structure. One possible solution is from the configuration within the index files.

This paper suggests using a join file which lists the table and the primary key for the table. This table (see Figure 4) allows two tables to be joined. The table contains two entries which are the table name and the primary key of the table. In the example below, both the Event table and the GINF table can be joined and both share the same primary key (ID_ODSP).

The process for creating the join file is manual but as discussed above in the section titled Proposed Configuration there is the possibility of automating this process. The caveat when creating an automatic script is to identify which tables have an identifiable relationship as well as what contrives to make that relationship. In the simple case discussed within this paper, the relationship is easy to identify and easy to create as only two tables exist. In larger more complicated database environments identifying these relationships may be harder to identify. Using deep learning techniques to identify which tables are related and how that relationship exists may be required.

4. Conversion Steps

The solution proposed by this paper allows for the natural language query "What are the odds on a game involving caro?" to be converted into an SQL statement using the following steps:

```
event=id_odsp

ginf=id_odsp
```

Figure 4. The join properties file lists the table name with the primary key which allows multiple tables to be joined.

1) Tag the natural language statement. The Open NLP tagger process takes the original statement and labels each word r component with a natural language tag. An example output from the tagging process will look like
what_IRRare_IRRthe_IRRodds_NPon_IRRa_IRRgame_IRRevent_APinvolving_IRRcaro_NP.

2) Looking at Figure 1 the grammar file identifies that the word event has the tag "AP". The conversion process identifies AP as a table. Using this information, the first part of the query is "*select * from event*".

3) The next step taken by the proposed is to identify that the query should join the events and the GINF table together as the query is asking for odds from the GINF table and player (caro) from the events table. The join table specifies that the tables' event and ginf are joined by the column ID_ODSP. This creates the where clause "*where event.id_odsp = ginf.id_odsp*".

4) The final step is to identify that the player being searched for is "caro" (see above). This gives the final part of the query *where player = "caro"*.

5) The query can now be joined into *select * from events where event.id_odsp= ginf.id_odsp where player = "caro"*.

6) Currently, the select statement just uses "*select * from*". The next step is to retrieve just the requested data or columns from the database. Through the use and application of machine learning techniques it is anticipated that select everything could be reduced to selecting only relevant columns from the query.

5. Training the Model

The Open NLP toolkit model uses machine learning algorithms at its core. Having created the configuration to be used as a model, the next step is training the Apache Open NLP model. Training the model is an important aspect of the Apache Open NLP process. The mathematical models used by the Open NLP application require that the model is trained. The training allows the model to perform the word tagging using the grammar file more accurately than would have been otherwise achieved. The machine learning models used by Open NLP for training include maximum entropy and perceptron-based machine learning.

The use of a maximum entropy model as described by Ratnaparkhi [8], ensures that the model best represents the current state of knowledge. The current state of knowledge in the case of the model proposed by this paper is the training set of questions being asked by users to query the underlying data repository.

The solution allows for more questions to be added as the process evolves. The additional questions can be added as part of an automated process or manually. Each question added would need to be tagged and the process retrained. This allows for the continued evolution of the system.

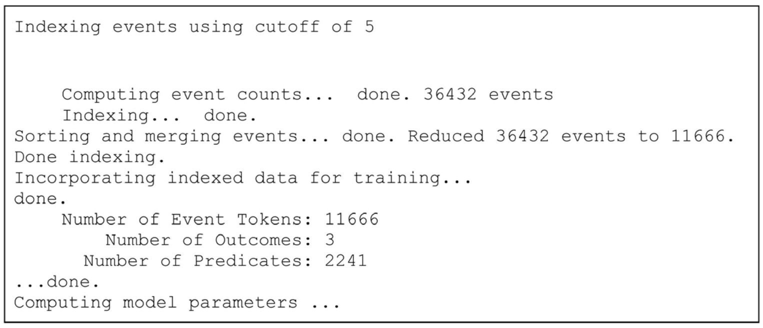
The tagging model used for this solution is the Part of Speech (POS) tagger which converts every word into a token. Each token has an associated tag. Open NLP will use a probability model to predict the correct tag for each word in the sentence. The fewer the tags used the quicker the performance, this can be seen from testing and appears to be supported by Taghipour [10] but more thorough performance testing is required. The tests that were carried out were performed on whole sentences, which included tags that can be identified as having a database related value. An example of this would be where the name of a database table or table column appears in the natural language query. In the case of the natural language query "Which event has Abdoulaye Diaby played in.", "event" is an identifiable database table. The sentence can then be processed, and relevant tags will be applied to the parts of the query (see **Table 1**), irrelevant tags will be ignored.

The Open NLP model training task process output: The output from training the model against the grammar file, which contains the list of potential asked questions that is shown in **Figure 5**.

As can be seen from the training output, the test was run against a training file with approximately 36,000 entries that were processed and indexed. From the 36,432 source entries, 11,666 were identified as either significant or unique. The number of outcomes in **Figure 5** refers to the number of possible outcomes from the model. For the shallow parsing approach proposed by this paper, the number is not significant. Though not significant for this paper the number of predicates could indicate the number of sentences in the data frame. The predicate identifies what is happening with the subject of a sentence. Though this might be helpful when trying to understand the content or meaning of the sentence for the shallow parse approach being taken by this paper the number of predicates is inconsequential.

6. Evaluation

During the evaluation phase of the proposed system, the idea was to measure the performance of the natural language conversion to SQL. The Java Virtual Machine (JVM) usage was monitored and the code profiled. The details of the proposed system performance are discussed in this section.



```
Indexing events using cutoff of 5


    Computing event counts...  done. 36432 events
    Indexing...  done.
Sorting and merging events... done. Reduced 36432 events to 11666.
Done indexing.
Incorporating indexed data for training...
done.
    Number of Event Tokens: 11666
        Number of Outcomes: 3
      Number of Predicates: 2241
...done.
Computing model parameters ...
```

Figure 5. The output from the training model.

6.1. Computer System

The computer used for the development and testing of the application is of a standard desktop configuration. The very utilitarian nature of the computer used for developing and testing this solution supports the concept that the conversion process does not require a large, expensive dedicated server. The specifications of the test machine for the natural language to SQL conversion are shown in Table 4.

6.2. Java Virtual Machine

The Java Machine used for the development and testing of the application is again a standard build. The application does run on a single JVM instance, the settings for which are shown in Figure 6.

6.3. Performance Results

The profiling of software allows for some tangible method to measure software excellence [11] [12]. The tests performed on the software show the resources used for converting a natural language query into a SQL based query. A number of tools have been employed to monitor the performance of the application which includes Java Visual VM from Oracle, YourKit Java Profiler, and the Coverage tool from JetBrains IntelliJ Java IDE. These tools highlight the computer resources used by the code in terms of virtual memory allocation and call time per function. The concept of benchmarking software performance provides

```
JVM: OpenJDK 64-Bit Server VM (25.152-b8, mixed mode)

Java: version 1.8.0_152-release, vendor JetBrains s.r.o
Java Home: C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2017.3.1\jre64
JVM Flags: <none>
-Xms24m
-Xmx256m
-Dsun.jvmstat.perdata.syncWaitMs=10000
-Dsun.java2d.noddraw=true
-Dsun.java2d.d3d=false
-Dnetbeans.keyring.no.master=true
-Djdk.home=C:\Program Files\Java\jdk1.8.0_25
-Dnetbeans.home=C:\Program Files\Java\jdk1.8.0_25\lib\visualvm\platform
-Dnetbeans.user=C:\Users\rskeggs\AppData\Roaming\VisualVM\8u20
-Dnetbeans.default_userdir_root=C:\Users\rskeggs\AppData\Roaming\VisualVM
-XX:+HeapDumpOnOutOfMemoryError
-XX:HeapDumpPath=C:\Users\rskeggs\AppData\Roaming\VisualVM\8u20\var\log\heapdump.hprof
-Dsun.awt.keepWorkingSetOnMinimize=true
-Dnetbeans.dirs=C:\Program Files\Java\jdk1.8.0_25\lib\visualvm\visualvm;C:\Program
Files\Java\jdk1.8.0_25\lib\visualvm\profiler
```

Figure 6. The shows the setting for the Java Virtual Machine on the test server. The output was taken from the Java Visual VM application version 1.8.0_25 (build 140407).

Table 4. Server specifications.

Variable	Value
Operating System	Windows 7 Enterprise
Service Pack	SP1
Processor	Intel® Core™ i5-4570 CPU @3.2GHz
Installed Memory	8.00 GB
System Type	64-bit Operating System

a tangible metric to evidence the performance of a software solution as supported by Sims *et al.* [12]. The benchmarking work carried out by Siewiorek *et al.* [13] highlights the fact that monitoring memory is key to understanding the performance of a software solution. These techniques update the work by Gama *et al.* [14] and Whaley [15].

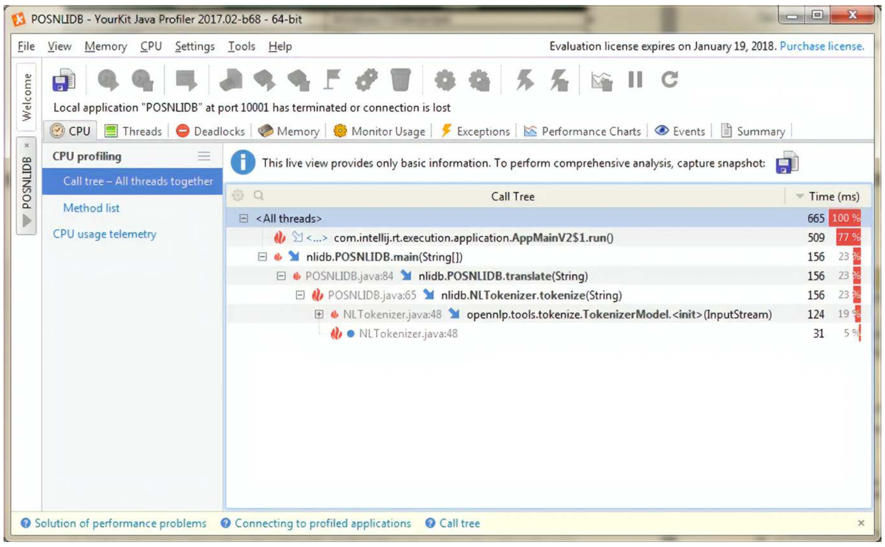
The YourKit Java profiler was used to measure the CPU of a conversion from a natural query to SQL. The profile modelled the application through the required classes as part of the execution cycle. **Figure 6** shows the performance in milliseconds that each class takes to complete a task. **Figure 7** shows just how much of the code gets executed when converting a simple natural language query to an SQL statement. For the simple example used as part of the test the execution time to convert the natural language query to SQL took a total of 665 milliseconds.

The Java Visual VM tool provides detailed information about Java applications while being executed on a Java Virtual Machine. The performance figures highlight the fact that no specialist hardware is required to run the process, which could be hosted on commodity hardware. To substantiate this **Figure 7** shows the screenshot from of the Visual Machine usage, that the largest resource allocation during testing was 42 Mb which accounted for 51% of all memory allocations by the virtual machine. Running tests against larger data will use more resources but the need to move to specialist hardware may not be a requirement, though further testing will need to be conducted to determine more accurately resource requirements. Tuning for performance in high throughput environments can also be managed by distributing resources across a platform when bottlenecks are identified. More in depth testing will need to be carried out to understand where and when these limits are reached (**Figure 8**).

Having completed a conversion and extraction of data from the dataset the next step was to compare performance the system discussed in this paper with other comparable systems. For this, the paper by Joshi, Akerkar [7] which proposed a similar approach using a Part of Speech based algorithm for converting natural language into an extraction-based query. The researchers compared the performance for two systems and the results are summarised in **Table 5**.

Table 5. Shows the performance figures from Joshi, Akerkar [7]

Type of Data No. of Words	Time Required by QTAG (Used in Enlight)	Time Required by Minipar (Used in Sapere)
News Extract Times of India (202 Words)	1.71 secs	2.88 secs
Reply START QA System (251 Words)	1.89 secs	3.11 secs
University Information NMU Broacher (226 Words)	1.55 secs	2.86 secs
Brazil Information Source: Wikipedia (226 Words)	1.67 secs	3.13 secs
Average	1.705 secs	2.995 secs

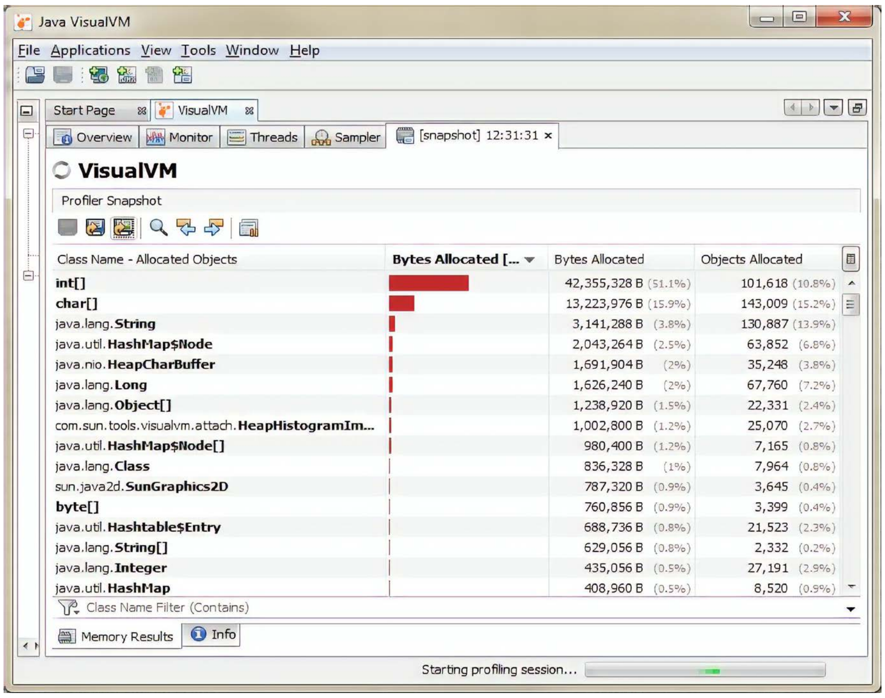

POSNLIDB - YourKit Java Profiler 2017.02-b68 - 64-bit
File View Memory CPU Settings Tools Help
Evaluation license expires on January 19, 2018. Purchase license.
Welcome
POSNLIDB
CPU
Threads
Deadlocks
Memory
Monitor Usage
Exceptions
Performance Charts
Events
Summary
CPU profiling
Call tree – All threads together
Method list
CPU usage telemetry
Solution of performance problems
Connecting to profiled applications
Call tree

Local application "POSNLIDB" at port 10001 has terminated or connection is lost

This live view provides only basic information. To perform comprehensive analysis, capture snapshot:

```
Call Tree                                                                                      Time (ms)
<All threads>                                                                                  665  100 %
    <...> com.intellij.rt.execution.application.AppMainV2$1.run()                              509   77 %
  nlidb.POSNLIDB.main(String[])                                                                156   23 %
    POSNLIDB.java:84 nlidb.POSNLIDB.translate(String)                                          156   23 %
      POSNLIDB.java:65 nlidb.NLTokenizer.tokenize(String)                                      156   23 %
        NLTokenizer.java:48 opennlp.tools.tokenize.TokenizerModel.<init>(InputStream)          124   19 %
        NLTokenizer.java:48                                                                     31    5 %
```

Figure 7. Shows the execution time the conversion process took courtesy of the YourKit Java Profiler.



Class Name - Allocated Objects	Bytes Allocated [...	Bytes Allocated	Objects Allocated
int[]		42,355,328 B (51.1%)	101,618 (10.8%)
char[]		13,223,976 B (15.9%)	143,009 (15.2%)
java.lang.String		3,141,288 B (3.8%)	130,887 (13.9%)
java.util.HashMap$Node		2,043,264 B (2.5%)	63,852 (6.8%)
java.nio.HeapCharBuffer		1,691,904 B (2%)	35,248 (3.8%)
java.lang.Long		1,626,240 B (2%)	67,760 (7.2%)
java.lang.Object[]		1,238,920 B (1.5%)	22,331 (2.4%)
com.sun.tools.visualvm.attach.HeapHistogramIm...		1,002,800 B (1.2%)	25,070 (2.7%)
java.util.HashMap$Node[]		980,400 B (1.2%)	7,165 (0.8%)
java.lang.Class		836,328 B (1%)	7,964 (0.8%)
sun.java2d.SunGraphics2D		787,320 B (0.9%)	3,645 (0.4%)
byte[]		760,856 B (0.9%)	3,399 (0.4%)
java.util.Hashtable$Entry		688,736 B (0.8%)	21,523 (2.3%)
java.lang.String[]		629,056 B (0.8%)	2,332 (0.2%)
java.lang.Integer		435,056 B (0.5%)	27,191 (2.9%)
java.util.HashMap		408,960 B (0.5%)	8,520 (0.9%)

Figure 8. This shows the memory allocation of the NLIDB process. Courtesy of Oracle's Java VisualVM.

The paper [7] did not specify the specification of the computer used to carry out the benchmark. The questions used by the paper [7] were taken from the TREC-2005 Question Database but there was some ambiguity in identifying the actual datasets used for the benchmarking. In comparison, this paper has taken a much larger dataset and has added the additional complexity of creating a join between two tables. The natural language questions used by this paper were of a similar complexity to the questions used in testing carried out by Joshi, Akerkar

[7] and are listed in Table 6. The average conversion time using the solution proposed by Joshi *et al.* was 1.7 seconds with the fastest being 1.5 seconds.

Testing the solution proposed by this paper the conversion time from natural language to structured query language took consistently under 700 milliseconds. The datasets from this paper consists of two files one containing over 36,000 events and the other over 11,000 (see Figure 5). Where also larger than the datasets used by Joshi *et al.* as these datasets contained approximately 220 records (see Table 5). Table 5 also shows the completion of time for the solution proposed by Joshi *et al.* and Table 7 also contains the times of each process to complete by the solution discussed in this paper. In summary, the tables highlight the improvements in performance the approach being taken by the paper as over existing solutions.

7. Conclusions

There are a number of limitations to the system being proposed in this paper. The storage space required for the index file and index file might make this solution unworkable. More testing against larger datasets is also required to understand the limitations and performance of the proposed solution. This paper has suggested a solution for joining tables together. Further testing would also be required to validate the performance of joining more than two tables.

The biggest issue that has not been addressed by this paper is that around the selection of data points being retrieved from the underlying database. Currently, the solution relies on the statement SELECT * which retrieves all data points from the tables being searched. Retrieving data from all columns in the target database could prove to be costly in terms of memory and processing resources. Refining the SELECT statement could possibly be achieved through the use of deep learning techniques. It may be possible to identify columns in tables that have a higher probability of being selected.

Table 6. Sample questions used for performance comparison by Joshi, AkerKer [7].

Who killed militants?
Who did Forman defeat for his first heavyweight championship?
What do frogs eat?
Who visited Bill Clinton?
Who did France beat for the World Cup?
What Shiite leaders were killed in Pakistan?
What is the largest volcano in the Solar System?
What is the longest river in the world?

Table 7. Performance from the proposed system which includes the conversion from natural language to SQL.

College	SQL Conversion	Data Extraction
ginf.csv (19531 Words) events.csv (13697026 Words)	0.665 secs	0.9 secs

Regardless of the identifiable short comings from the proposed system, the paper has reinforced the benefits of using part of speech within a framework that translates natural language into a query language for searching a database. Performance of NLIDB solutions has been an issue that researchers are continually trying to improve upon [1] [3] [5] [7] [16]. As can be seen from this paper the performance of the proposed system is an improvement on the performance recorded by Enlight and Sapere (**Table 7**).

The shallow nature of the parsing through the use of the natural language part of speech also reduces the need to understand the complexity underpinning language nuance. Further work will be carried out to improve the performance of the proposed system as well as reduce the number of identifiable shortcomings. The proposed work will look at the use of deep learning to refine the select statement.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Soumya, M.D. and Patil, B.A. (2017) An Interactive Interface for Natural Language Query Processing to Database Using Semantic Grammar. *International Journal of Advance Research, Ideas and Innovations in Technology*, **3**, 193-198.

[2] Kim, M. and Kim, H. (2018) Dialogue Act Classification Model Based on Deep Neural Networks for a Natural Language Interface to Databases in Korean. *IEEE International Conference on Big Data and Smart Computing*, Shanghai, 15-17 January 2018, 537-540. https://doi.org/10.1109/BigComp.2018.00090

[3] Bais, H., Machkour, M. and Koutti, L. (2018) An Arabic Natural Language Interface for Querying Relational Databases Based on Natural Language Processing and Graph Theory Methods. *International Journal of Reasoning-Based Intelligent Systems*, **10**, 155-165. https://doi.org/10.1504/IJRIS.2018.092221

[4] Li, Y.Y. and Rafiei, D. (2017) Natural Language Data Management and Interfaces: Recent Development and Open Challenges. *Proceedings of the* 2017 *ACM International Conference on Management of Data*, Chicago, 14-19 May 2017, 1765-1770. https://doi.org/10.1145/3035918.3054783

[5] Jwalapuram, P. and Mamidi, R. (2017) Domain Independent Keyword Identification for Question Answering. *International Conference on Asian Language Processing*, Singapore, 5-7 December 2017, 95-98. https://doi.org/10.1109/IALP.2017.8300554

[6] Voorhees, E.M. (2001) The TREC Question Answering Track. *Natural Language Engineering*, **7**, 361-378. https://doi.org/10.1017/S1351324901002789

[7] Joshi, M.R. and Akerkar, R.A. (2008) Algorithms to Improve Performance of Natural Language Interface. *International Journal of Computer Science and Applications*, **5**, 52-68.

[8] Ratnaparkhi, A. (1996) A Maximum Entropy Model for Part-of-Speech Tagging. In: *Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, Stroudsburg, 133-142.

http://aclweb.org/anthology/W/W96/W96-0213

[9] Baldridge, J. (2005) The OpenNLP Project. http://opennlp.apache.org/

[10] Taghipour, K. and HweeTou, N. (2015) One Million Sense-Tagged Instances for Word Sense Disambiguation and Induction. In: *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, Association for Computational Linguistics, Beijing, 338-344. https://doi.org/10.18653/v1/K15-1037 http://www.aclweb.org/anthology/K15-1037

[11] Deuter, A. and Hans-Jürgen, K. (2015) Applying Manufacturing Performance Figures to Measure Software Development Excellence. In: *Software Measurement*, Lecture Notes in Business Information Processing, Springer, Cham, 62-77. https://doi.org/10.1007/978-3-319-24285-9_5

[12] Sim, S.E., Easterbrook, S. and Holt, R.C. (2003) Using Benchmarking to Advance Research: A Challenge to Software Engineering. 25*th International Conference on Software Engineering*, Portland, 3-10 May 2003, 74-83. https://doi.org/10.1109/ICSE.2003.1201189

[13] Siewiorek, D.P., Hudak, J.J., Suh, B.H. and Segal, Z. (1993) Development of a Benchmark to Measure System Robustness. *The* 23*rd International Symposium on Fault-Tolerant Computing*, Toulouse, 22-24 June 1993, 88-97. https://doi.org/10.1109/FTCS.1993.627311

[14] Gama, K., Pedraza, G., Lévêque, T. and Donsez, D. (2011) Application Management Plug-Ins through Dynamically Pluggable Probes. *Proceedings of the* 1*st Workshop on Developing Tools as Plug-Ins*, Waikiki, 28 May 2011, 32-35. https://doi.org/10.1145/1984708.1984718

[15] Whaley, J. (2000) A Portable Sampling-Based Profiler for Java Virtual Machines. *Proceedings of the ACM Conference on Java Grande*, San Francisco, 3-4 June 2000, 78-87. https://doi.org/10.1145/337449.337483

[16] Brad, F., Iacob, R., Hosu, I. and Rebedea, T. (2017) Dataset for a Neural Natural Language Interface for Databases (NNLIDB). *Proceedings of the* 8*th International Joint Conference on Natural Language Processing*, Vol. 1, 13 p. http://arxiv.org/abs/1707.03172

Appendix A

The standard list of tags and definitions used by Apache Open NLP (These tags are not a definitive list and are used by convention).

CC Coordinating conjunction
CD Cardinal number
DT Determiner
EX Existential there
FW Foreign word
IN Preposition or subordinating conjunction
JJ Adjective
JJR Adjective, comparative
JJS Adjective, superlative
LS List item marker
MD Modal
NN Noun, singular or mass
NNS Noun, plural
NNP Proper noun, singular
NNPS Propernoun, plural
PDT Predeterminer
POS Possessive ending
PRP Personal pronoun
PRP$ Possessive pronoun
RB Adverb
RBR Adverb, comparative
RBS Adverb, superlative
RP Particle
SYM Symbol
TO to
UH Interjection
VB Verb, base form
VBD Verb, past tense
VBG Verb, gerund or present participle
VBN Verb, past participle
VBP Verb, non 3rd person singular present
VBZ Verb, 3rd person singular present
WDT Whdeterminer
WP Whpronoun
WP$ Possessive whpronoun
WRB Whadverb

Appendix B

The data is from the dictionary text file. The data contains a dictionary with the textual description of each categorical variable coded with integers event_type.

1) Announcement
2) Attempt
3) Corner
4) Foul
5) Yellow card
6) Second yellow card
7) Red card
8) Substitution
9) Free kick won
10) Offside
11) Hand ball
12) Penalty conceded

EVENT_TYPE 2

13) Key Pass
14) Failed through ball
15) Sending off
16) Own goal

SIDE

1) Home
2) Away

SHOT_PLACE

1) Bit too high
2) Blocked
3) Bottom left corner
4) Bottom right corner
5) Centre of the goal
6) High and wide
7) Hits the bar
8) Misses to the left
9) Misses to the right
10) Too high
11) Top centre of the goal
12) Top left corner
13) Top right corner

SHOT_OUTCOME

1) On target
2) Off target
3) Blocked
4) Hit the bar

LOCATION

1) Attacking half
2) Defensive half
3) Centre of the box
4) Left wing

5) Right wing
6) Difficult angle and long range
7) Difficult angle on the left
8) Difficult angle on the right
9) Left side of the box
10) Left side of the six yard box
11) Right side of the box
12) Right side of the six yard box
13) Very close range
14) Penalty spot
15) Outside the box
16) Long range
17) More than 35 yards
18) More than 40 yards
19) Not recorded

BODYPART

1) right foot
2) left foot
3) head

ASSIST_METHOD

1) None
2) Pass
3) Cross
4) Headed pass
5) Through ball

SITUATION

1) Open play
2) Set piece
3) Corner
4) Free kick