Empowering Natural Language Interfaces to Databases with Aggregations

Alexandre F. Novello and Marco A. Casanova

Department of Informatics, PUC-Rio, Rio de Janeiro, RJ – Brazil
`{anovello, casanova}@inf.puc-rio.br`

Abstract. A Natural Language Interface to Database (NLIDB) refers to a database interface that translates a question asked in natural language into a structured query. Aggregation questions express aggregation functions, such as count, sum, average, minimum and maximum, and optionally a group by clause and a having clause. NLIDBs deliver good results for standard questions but usually do not deal with aggregation questions. The main contribution of this article is a generic module, called GLAMORISE (GeneraL Aggregation MOdule using a RelatIonal databaSE), that extends NLIDBs to cope with aggregation questions. GLAMORISE covers aggregations with ambiguities, timescale differences, aggregations in multiple attributes, the use of superlative adjectives, basic recognition of measurement units, and aggregations in attributes with compound names.

Categories and Subject Descriptors: H.3 [**Information Storage and Retrieval**]: Miscellaneous; I.2.7 [**Natural Language Processing**] I.7 [**Document and Text Processing**]: Miscellaneous.

Keywords: Natural Language Interface to Database (NLIDB), Question Answering (QA), Databases, Natural Language Processing (NLP), Aggregation, SQL.

1. INTRODUCTION

Question Answering (QA) is a field of study dedicated to constructing systems that automatically answer questions asked in natural language. Database interfaces that translate questions asked in natural language into structured queries are known as Natural Language Interfaces to Database (NLIDBs). The typical data flow of an NLIDB is: (1) the user submits a question *N* in natural language through the user interface; (2) the NLIDB translates *N* into a structured query *Q* – SQL in the case of an RDBMS or SPARQL in the case of a triplestore; (3) the database system returns a result set *R* for *Q* to the NLIDB; (4) the NLIDB transfers *R* to the user interface. The translation process (Step 2) goes through several phases, such as question analysis, phrase mapping, disambiguation, and query construction.

One of the less well-solved tasks in NLIDBs is the treatment of aggregation questions, that is, questions that express aggregation functions, such as count, sum, average, minimum and maximum, and optionally a group by clause and a having clause. Indeed, a recent survey compared 26 different NLIDBs [Affolter et al. 2019], using ten questions in natural language that cover different aspects of structured queries (join, filter, aggregation, ordering, union, subquery, and concept). The survey found that most of the NLIDBs analyzed correctly answered simple questions, but only 2 out of the 26 NLIDBs (7.69%) managed to correctly answer the only aggregation question in the benchmark. In particular, aggregation questions that use a timescale distinct from that of the stored data pose a difficult challenge since they require an extra timescale conversion step. For example, consider the question: *"What was the average yearly compensation for employees in 2020?"* If the compensation data is on a weekly scale, it is necessary to understand that the question is on an annual scale and perform a conversion operation. Note that, in this case, it is not enough to multiply the average weekly remuneration by 52, as the salary may have changed over the year or changed due to sporadic events, such as vacations, or events with specific periodicity such as bonuses. It is necessary to filter the sum of all 2020 compensation data per employee and then compute the average.

The main contribution of this article is a generic module, called GLAMORISE (GeneraL Aggregation MOdule using a RelatIonal databaSE), that extends NLIDBs to cope with aggregation questions, provided that the result of the NLIDB is, or can be transformed into, a result set in the form of a table. Hence, GLAMORISE can also be used with triplestore (RDF store) NLIDBs, with the proviso that the result is presented in a tabular format (SELECT SPARQL queries). GLAMORISE supports aggregations with certain specificities, including ambiguities, timescale differences, aggregations in multiple attributes, the use of superlative adjectives, basic recognition of measurement units, and aggregations in attributes with compound names. In particular, to the best of our knowledge, GLAMORISE is the first attempt to provide a generic solution for aggregations with timescale differences.

The article also describes experiments with GLAMORISE, integrated with an NLIDB. The first experiment uses a mock NLIDB and 22 static questions as a proof-of-concept. The second experiment integrates GLAMORISE with a real NLIDB, NaLIR [Li and Jagadish, H. V. 2014]. NaLIR is the NLIDB that performed best in the survey described in [Affolter et al. 2019], including answering an aggregation question. The aggregation layer was removed from NaLIR, leaving GLAMORISE to address this issue, while NaLIR addressed the remaining issues involved in an NLIDB. To validate the results of the GLAMORISE/NaLIR integration, the second experiment used the same 22 questions developed for the proof-of-concept and the 17 benchmark questions using the Microsoft Academic Search (MAS) database originally used by the NaLIR research group. All code, natural language questions, and links to the databases used can be found at https://github.com/novello/GLAMORISE (accessed on Jun/2021) for reuse or reproducibility of the experiments and results. An online version of the ready-to-use experimental system can be found at https://glamorise.gruposantaisabel.com.br/ (accessed on Jun/2021)

The initial research involving GLAMORISE, with a prototype and the proof-of-concept with the mock NLIDB, was published in [Novello and Casanova 2020]. This article is an extended version of this paper, expanding the description of GLAMORISE and presenting the results of the integration of GLAMORISE with NaLIR.

The rest of this article is structured as follows. Section 2 summarizes related work. Section 3 describes the architecture of GLAMORISE. Section 4 discusses the types of aggregation GLAMORISE supports. Section 5 covers the experiments. Section 6 contains a brief discussion about the limitations of GLAMORISE. Finally, Section 7 contains the conclusions and lists directions for future research.

2. RELATED WORK

SQAK (SQL Aggregates using Keywords) [Tata and Lohman 2008] is a framework that allows users to perform queries with aggregations using only keywords, with no knowledge of the database schema or SQL. The concept of Simple Query Network (SQN), which is similar to the Steiner Tree, but with better results for this purpose, was created. A greed algorithm was developed to find the minimal SQN, since this is an NP-Complete problem, and used to build the SQL. Our work and SQAK deal with similar issues, aiming at using keywords for the final translation into SQL. The difference is that their work and others that we will refer to in this section are complete and monolithic NLIDBs, while ours aims at enabling existing NLIDBs to handle aggregations, which they do not perform well.

NaLIR [Li and Jagadish, H. V. 2014a; Li and Jagadish, H. V. 2014b; Li and Jagadish 2016] is a generic NLIDB capable of handling aggregations, nesting, and various types of joins. The Stanford NLP Parser is used to convert from natural language into a parser tree. The approach followed is to get feedback from the user and return the adjusted parse trees to the user in natural language to select the natural language question that makes the most sense or revise accordingly. In our view, this exchange of information jeopardizes the user experience, as they have the impression that they are carrying out work that should be done by the NLIDB, regardless of the extent to which it ensures that the resulting query is correct after the adjustments. However, the NaLIR version we integrated into our work does not have the layer called interactive communicator. We preferred to generate a structured query from the natural language query without user intervention. Still, NaLIR has good results and is considered one of the best academic NLIDBs.

A novel approach to building NLIDB is presented in [Gupta et al. 2012]. The approach is based on dependency trees using Computational Paninian Grammar (CPG) [Bharati et al. 2014] in which the relationships are syntactic-semantic. CPG was originally developed for Indian languages and afterward extended to support English. The authors argued that using this technique makes the trees more semantic than other kinds of dependency trees, thus making them easier to map to SQL. In the following article [Gupta and Sangal 2013], the framework was extended to handle aggregation processing with different types of aggregation operations in natural language, including quantitative and qualitative aggregations, and those combining quantifiers or relational operators with aggregation. A separate layer in the querying process was devised. First, the SQL query is generated and processed in the RDBMS without the aggregation, and then the aggregation is processed in the returned result set. The whole concept is explained in detail in [Abhijeet Gupta 2013]. This work served as an inspiration regarding the isolation and classification of the parts that compose the aggregation and the processing of the aggregation in the returned result set described in Section 3.

Other approaches also adopted a two-stage process, similar to the previous work. ITCM NLIDB [Pazos R et al. 2016] was first created without the ability to process aggregations (or subqueries), and a second work [Pazos R et al. 2018] added a module capable of carrying out these activities. The NLIDB kernel is composed of three main modules: a lexical analyzer (tags the words in the lexicon with their syntactic categories); a syntactic module (leaves only one syntactic category for each lexical component and disregards irrelevant ones); and a semantic module (maps the result of the previous steps in tables and columns in the database). Although simpler than previous works, the main contribution of this work was to identify recurring problems in how aggregations (and subqueries) are stated in natural language and propose solutions to these problems. The problem is that, as in NaLIR, they did not seek an automatic solution but returned the ambiguities for the user to resolve, which, in our opinion, makes the process less user-friendly. Nevertheless, this work was helpful as it confirmed various recurring problems when dealing with queries in natural language with aggregations, which we also identified.

A recent survey [Affolter et al. 2019] compared 26 different NLIDBs, using ten natural language questions covering various aspects of structured queries (join, filter, aggregation, ordering, union, subquery, and concept). Only one of ten questions (Q7 – What was the best movie of each genre?) is an aggregation question, and only 2 (NaLIR/NaLIX, NLQ/A), out of the 26 NLIBDs (7.69%), managed to answer Q7 correctly, which supports the claim that NLIDBs do not handle aggregation questions well.

Three other papers that are tangential to this article are worth mentioning. TiQi is a solution for Software Traceability, allowing users to build natural language trace queries that are converted into SQL [Pruski et al. 2015]. SpeakQL facilitates the construction of SQL queries using spoken questions in mobile devices [Shah et al. 2019]. Rather than focusing on questions with aggregation, in [Pinheiro et al. 2020], aggregation is employed to improve the user experience when browsing query results.

As mentioned in the Introduction, GLAMORISE can be integrated with any NLIDB to support aggregation questions, as shown in Fig. 1. In particular, GLAMORISE can be integrated with RDF NLIDBs, with the proviso that the query results are presented in a tabular format (SELECT SPARQL queries). That is, contrasting with the approaches described above, GLAMORISE is not a monolithic

NLIDB. Furthermore, GLAMORISE can be extended to cover other types of aggregations, including domain-specific aggregations, as discussed in Sections 4 and 6. Also, to the best of our knowledge, GLAMORISE is the first attempt to provide a generic solution for aggregations with timescale differences. Finally, as addressed in Section 5, GLAMORISE exhibited good performance on a carefully selected set of questions extracted from [Li and Jagadish, H. V. 2014a].

3. GLAMORISE – A PROPOSED SOLUTION

3.1 Architecture

As discussed in the introduction, NLIDBs usually do not deal with aggregations, but they return good results for typical queries. GLAMORISE extends NLIDBs to support queries with aggregations, as long as the result is, or can be transformed into, a result set in the form of a table. Fig. 1 shows the architecture of GLAMORISE.

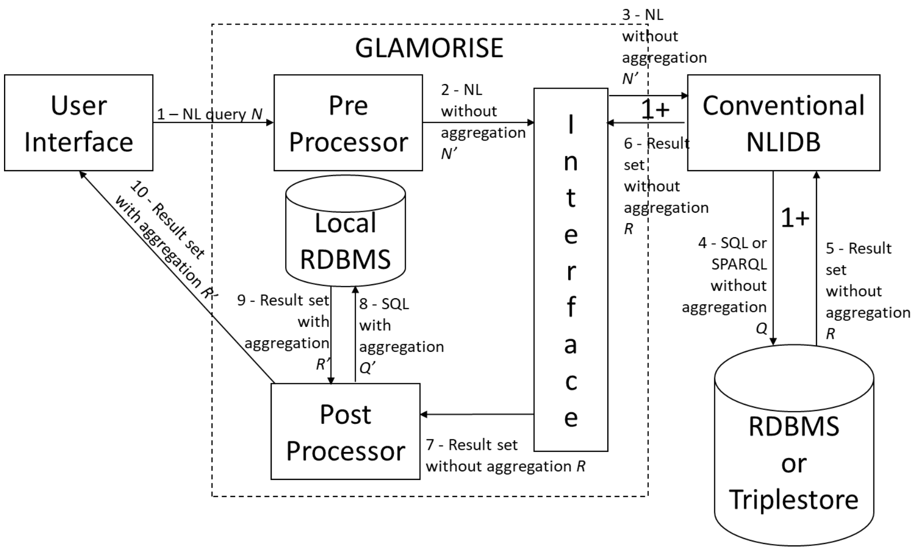

GLAMORISE
User Interface
1 – NL query N
Pre Processor
2 - NL without aggregation N'
Interface
3 - NL without aggregation N'
1+
Conventional NLIDB
6 - Result set without aggregation R
Local RDBMS
1+
4 - SQL or SPARQL without aggregation Q
5 - Result set without aggregation R
10 - Result set with aggregation R'
9 - Result set with aggregation R'
8 - SQL with aggregation Q'
Post Processor
7 - Result set without aggregation R
RDBMS or Triplestore

Fig. 1 – GLAMORISE Architecture.

The user starts by typing a natural language question *N* through the user interface (1). The Preprocessor Module removes the aggregation elements and transforms *N* into a natural language question *N'* without aggregation and registers all the elements related to the aggregation, to be subsequently used by the Postprocessor Module. Then, *N'* is sent to the Interface Module (2), responsible for the integration with the conventional NLIDB (3). Next, the question is processed by the conventional NLIDB and converted into a structured query *Q* (4) – SQL in the case of an RDBMS or SPARQL in the case of a triplestore – and a result set *R* without any aggregations is returned in a tabular format (5, 6 and 7). Additionally, the metadata of the result set can be retrieved to process more complex questions. To improve the results of a conventional NLIDB and depending on the implementation of the integration at the Interface Module, Steps (3) to (6) could be iterated until the result achieved is satisfactory. The Postprocessor Module stores *R* in a local SQLite[1] RDBMS and processes the

[1] https://www.sqlite.org/, accessed on Jun/2021.

aggregation over *R* by creating an SQL query *Q'* (8), which returns the final result set with aggregations *R'* (9) and presents it to the user interface (10).

In more detail, the Preprocessor Module maps natural language terms to the respective aggregation functions and identifies whether the natural language question also expresses a group by clause or a having clause. Some of these terms could be removed from the question to guarantee that the conventional NLIDB will not be confused by their existence, leading to an incorrect mapping. Examples will be described later on in this section.

The Interface Module is responsible for the integration with the conventional NLIDB. Its implementation is dependent on the conventional NLIDB.

The Postprocessor Module is responsible for constructing the aggregation query in SQL. It first stores the result set received from the Interface in a table in a local RDBMS (SQLite) and then uses this table to create an SQL query with aggregation. This is achieved by analyzing the metadata saved by the Preprocessor Module, including the aggregation functions (sum, max, min, avg, and count), and recognizing the fields in which these functions should be applied in the result set received by the Interface Module. Then, an identical process is executed for the group by and having clauses, reading the metadata to determine if grouping is required, which fields are involved, and mapping them in the result set. All fields identified as belonging to a group by clause are also inserted in the SELECT clause and in the ORDER BY, the latter for the user's convenience only. This module is also responsible for analyzing any timescales that should be converted into a subquery.

4. TYPES OF AGGREGATION

This section discussed some of the types of aggregations that GLAMORISE supports using a simple sample database. A description of the other types of aggregation that GLAMORISE handles can be found in [Novello 2021]. They are multiple attribute aggregations, aggregations of attributes with compound names, basic recognition of measurement units, nested aggregation functions, ellipsis, and the use of the having clause.

4.1 A Running Example

Consider a database with a single table storing the production of oil fields in Brazil, released by the National Petroleum Agency (ANP)[2].

Table I. ANP database table.

Name	**Type**
FIELD	TEXT
BASIN	TEXT
STATE	TEXT
OPERATOR	TEXT
CONTRACT_NUMBER	TEXT
OIL_PRODUCTION	REAL
GAS_PRODUCTION	REAL
MONTH	INTEGER

[2] http://www.anp.gov.br/, accessed on Jun/2021.

YEAR	INTEGER

Recall that aggregation questions express aggregation functions, such as count, sum, average, minimum and maximum, optionally a group by clause, and a having clause. For example, the natural language question *"How many fields are there in Paraná?"* would be translated to the following SQL query:

```
SELECT COUNT(DISTINCT field)
FROM anp
WHERE lower(state) like '%paraná%'
```

In this example, the aggregation function is count.

As a second example, consider the natural language question *"What was the maximum production of oil in the state of Ceará per field?"*, which would be translated to the following SQL query:

```
SELECT field, MAX(oil_production) AS max_oil_production
FROM anp
WHERE lower(state) like '%ceará%'
GROUP BY field
ORDER BY field
```

Note that this query has the additional group by clause. The order by clause is introduced just to simplify browsing the result set.

Consider now a question with a having clause *"What was the mean gas production per field with production greater than 100 cubic meters?"*, which is translated to the following SQL query:

```
SELECT field, AVG(gas_production) AS avg_gas_production
FROM anp
GROUP BY field
HAVING AVG(gas_production) > 100
ORDER BY field
```

Note that the way the data is stored in the database also influences the answer given to the natural language questions. Hence, the database should be designed (or at least database views should be included) to meet the responses expected by users. On the other hand, the lay user must have some notion of what granularity and how the information is stored in the database.

4.2 Aggregation Ambiguities

We separate ambiguities into two types. First, some ambiguities can be resolved directly by the NLIDB, such as a keyword that is mapped to more than one element of the database (table, attribute, data). Second, there are ambiguities that involve aggregation, i.e., a word that should be understood as part of the aggregation, but that can also be considered of another syntactic-semantic nature. Any ambiguity to be addressed by the NLIDB will be resolved without GLAMORISE even being aware of it. For ambiguities directly related to aggregations, GLAMORISE tries to recognize the true nature of the word according to the pattern of the utterance.

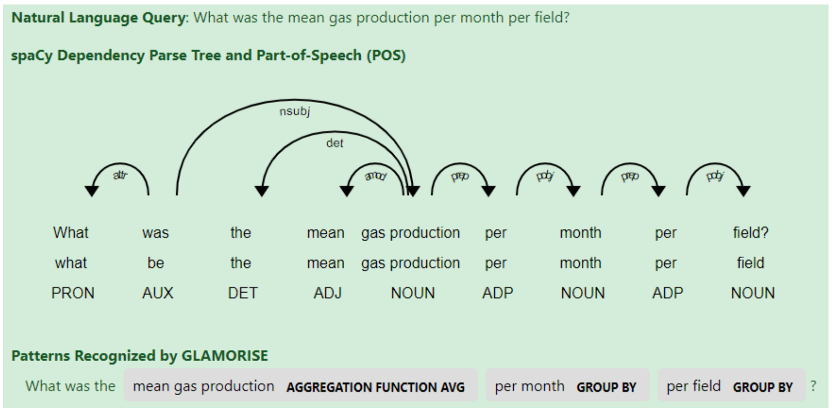
For example, the two sentences below have the same intent and should be translated to the same query. The only difference between them is the word order:

"What was the mean gas production per month per field?"

"What was the per month mean gas production per field?"

GLAMORISE uses the spaCy POS-tagging and parse tree functionalities. spaCy [Honnibal and Montani 2017] is an open-source software library for advanced natural language processing that features convolutional neural network models for Part-Of-Speech (POS)-tagging, parse tree [Honnibal and Johnson 2015], text categorization, and named-entity recognition (NER).

The problem with these two sentences is that the parse tree that spaCy generates is not perfect, despite being very good and outperforming other libraries, such as NLTK and the Stanford NLP Parser. Fig. 2 and Fig. 3 show the spaCy parse trees of these sentences and the interpretation GLAMORISE gives to them.


Natural Language Query: What was the mean gas production per month per field?

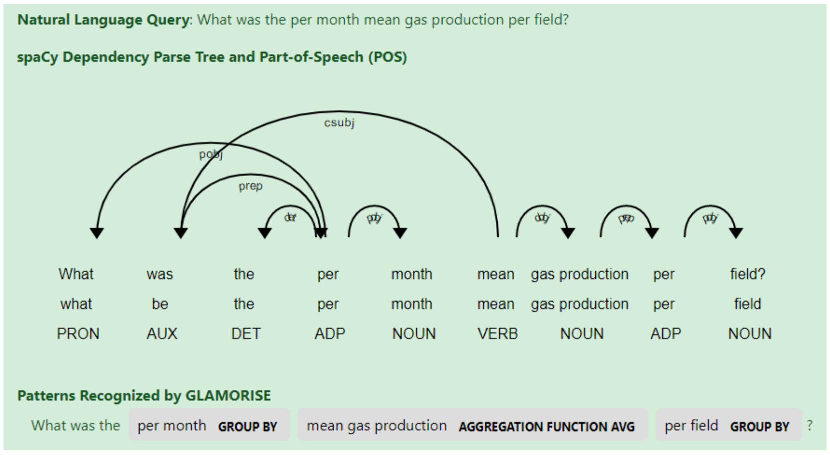
spaCy Dependency Parse Tree and Part-of-Speech (POS)

What	was	the	mean	gas production	per	month	per	field?
what	be	the	mean	gas production	per	month	per	field
PRON	AUX	DET	ADJ	NOUN	ADP	NOUN	ADP	NOUN

Patterns Recognized by GLAMORISE

What was the | mean gas production **AGGREGATION FUNCTION AVG** | per month **GROUP BY** | per field **GROUP BY** | ?

Fig. 2 – Aggregation ambiguity - first example.


Natural Language Query: What was the per month mean gas production per field?

spaCy Dependency Parse Tree and Part-of-Speech (POS)

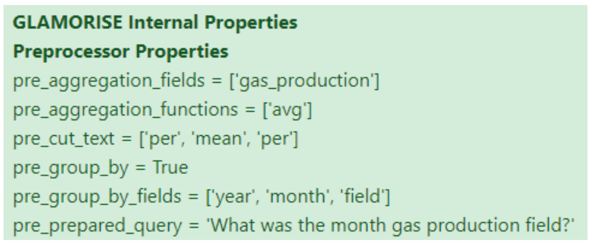
What	was	the	per	month	mean	gas production	per	field?
what	be	the	per	month	mean	gas production	per	field
PRON	AUX	DET	ADP	NOUN	VERB	NOUN	ADP	NOUN

Patterns Recognized by GLAMORISE

What was the | per month **GROUP BY** | mean gas production **AGGREGATION FUNCTION AVG** | per field **GROUP BY** | ?

Fig. 3 – Aggregation ambiguity – second example.

In the second question, the word “mean” was recognized in the first parse tree as ADJ, which means an adjective, while in the second question, it was wrongly recognized by the parse tree as VERB, which indicates a verb, despite the Preprocessor Module recognizing the true intent of “mean” and converting it to an avg (average) aggregation function, as shown in Fig. 4.



GLAMORISE Internal Properties
Preprocessor Properties
```
pre_aggregation_fields = ['gas_production']
pre_aggregation_functions = ['avg']
pre_cut_text = ['per', 'mean', 'per']
pre_group_by = True
pre_group_by_fields = ['year', 'month', 'field']
pre_prepared_query = 'What was the month gas production field?'
```

Fig. 4 – Aggregation ambiguity – Preprocessor recognizes correctly.

This behavior is possible because, in our experiments, we realized that the best results were not in the analysis of the Part-Of-Speech (POS) of the keyword, in the above case "mean", but the POS of the words that come after the keyword. The configuration of keywords and patterns is explained in detail in [Novello 2021].

We will use this case to demonstrate how GLAMORISE settings in JSON work. To recognize "mean", we have a rule with the following linguistic pattern.

```
"mean example": {
  "reserved_words": ["mean"],
  "pre_aggregation_functions": "avg",
  "pre_cut_text": true
}
```

The first declaration is the name of the rule, in this case, "mean example". Then, we define the reserved words of this rule, using the "reserved_words" parameter; in the example, only "mean" is set as a reserved word. Next, the "pre_aggregation_functions" parameter indicates which aggregation functions to use, in the case "avg". Finally, since the "pre_cut_text" parameter is set to true, GLAMORISE cuts the reserved word "mean" to avoid confusing the integrated NLIDB that is not prepared to deal with aggregations.

Within each rule, a set of words is expected after each reserved word, and they must have certain Part-Of-Speech (POS) and relations in the dependency tree. This is indicated through the "specific_pattern" parameter or the parameter "default_pattern", which is valid for all rules that do not have a "specific_pattern" defined, as is the case with this rule “mean example” above. The "default_pattern" is described below.

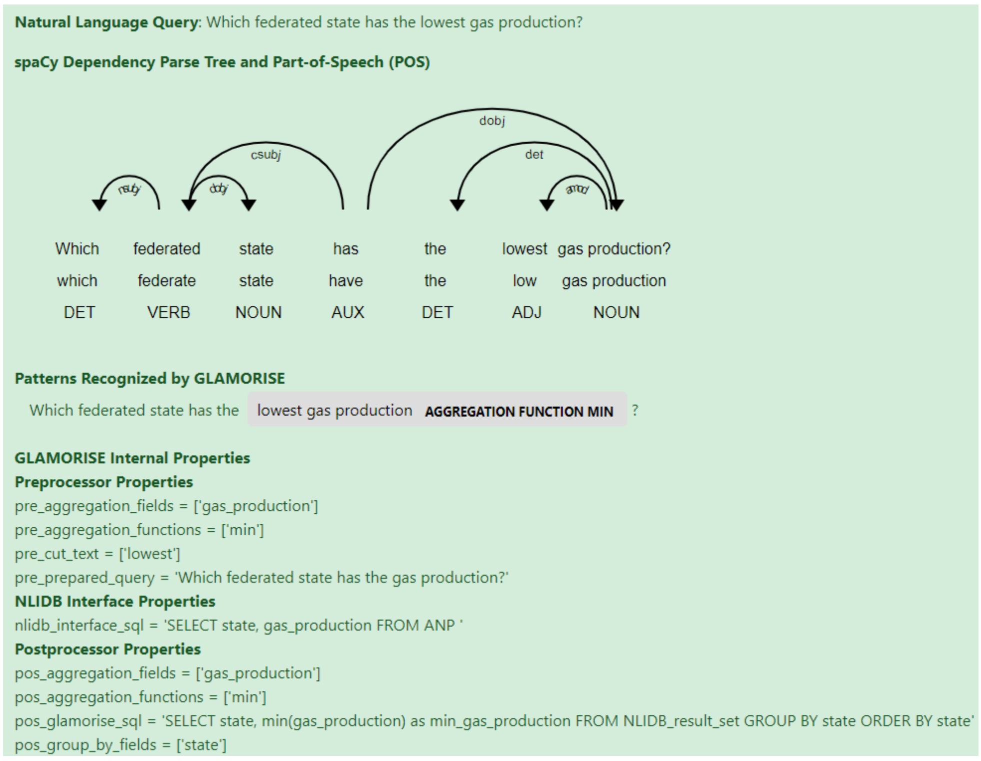
```
"default_pattern": [{"POS": "ADV", "OP": "*"},
              {"POS": "ADJ", "OP": "*"},
              {"POS": "NOUN", "LOWER": {"NOT_IN": ["number"]}}
       ]
```

This parameter defines the standard way in which a field is found. Used in conjunction with the rule “mean example” above, this use of the default pattern says that the reserved word (or keyword) “mean” must be followed by an optional adverb ({"POS": "ADV", "OP": "*"}), an optional adjective ({"POS": "ADJ", "OP": "*"}) or a noun ({"POS": "NOUN", "LOWER": {"NOT_IN": ["number"]}}), which is the field to be identified. This noun can be simple or compound. The declaration ("LOWER": {"NOT_IN": ["number"]}) indicates that "number" and "number of" are reserved words that are often used to mean a counting function, and the word "number" is a noun, so it should not be considered for field match purposes. For example, in the part of the sentence "mean gas production", the "mean" is taken from the "reserved_words" of the rule “mean example”. This rule uses "default_pattern" because it does not have "specific_pattern" defined, and "gas production" matches the “default_pattern” because it is a noun. Remembering that "default_pattern" waits for an adverb (optional), an adjective (optional), and a noun, which "gas production" matches. Therefore, the rule as a whole works, as "mean" matches "reserved_words" and "gas production" matches the “default_pattern”.

The second kind of ambiguity that we deal with uses terms such as “greater than”, “more than”, “less than”. This kind of expression could be translated into two different clauses in SQL depending on the database schema. If the condition is directly related to the value of a field in a table, the condition is translated into a WHERE clause. If it is related to any grouping action, the condition is translated into a HAVING clause. As we do not have access to the database schema that NLIDB will query, we do not know in advance which of the two cases we are dealing with. Bearing that in mind, we save the expression to be used later by the Postprocessor in the HAVING clause. We also do not remove the expression from the query communicated to the NLIDB since if the NLIDB can interpret it, then the expression impacts the WHERE clause, otherwise the NLIDB will ignore the expression.

4.3 Superlative Adjectives

Superlative adjectives are suppressed and, depending on the type of superlative, a *min* or *max* function is added to the aggregation functions metadata, together with the respective aggregate field. The superlative adjective is then removed from the query not to confuse the NLIDB with a term that it cannot handle. Fig. 5 presents an example.



Natural Language Query: Which federated state has the lowest gas production?

spaCy Dependency Parse Tree and Part-of-Speech (POS)

Which	federated	state	has	the	lowest	gas production?
which	federate	state	have	the	low	gas production
DET	VERB	NOUN	AUX	DET	ADJ	NOUN

Patterns Recognized by GLAMORISE

Which federated state has the lowest gas production **AGGREGATION FUNCTION MIN** ?

GLAMORISE Internal Properties
Preprocessor Properties
pre_aggregation_fields = ['gas_production']
pre_aggregation_functions = ['min']
pre_cut_text = ['lowest']
pre_prepared_query = 'Which federated state has the gas production?'
NLIDB Interface Properties
nlidb_interface_sql = 'SELECT state, gas_production FROM ANP '
Postprocessor Properties
pos_aggregation_fields = ['gas_production']
pos_aggregation_functions = ['min']
pos_glamorise_sql = 'SELECT state, min(gas_production) as min_gas_production FROM NLIDB_result_set GROUP BY state ORDER BY state'
pos_group_by_fields = ['state']

Fig. 5 – Example of a superlative adjective.

The *min* aggregation function is identified due to the superlative adjective “lowest”; also, “gas production” is identified as the aggregate field related to the pattern configuration.

4.4 Subquery - Aggregations with timescale differences

As a motivating example, consider the question *"What was the average yearly production of oil in the state of Alagoas?"*. From Table I, note that the ANP table has two attributes, one for the year and another for the month, in addition to the production (oil or gas). That is, each tuple stores the monthly production, whereas the question asks for the yearly production. The SQL translation for this question would be:

```
SELECT AVG(SUM(oil_production)) as avg_sum_oil_production
FROM nlidb_result_set
WHERE state = 'Alagoas'
GROUP BY year
```

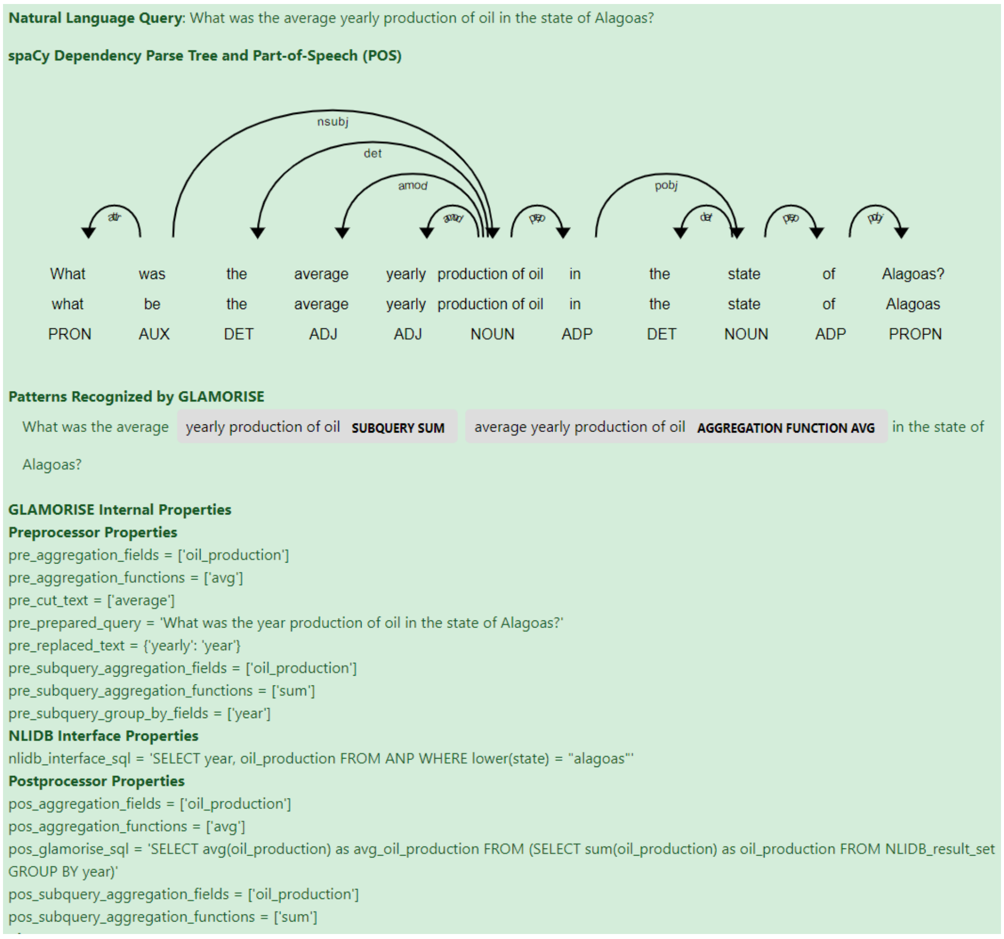
Note that this query is different from the previous examples since it has two aggregation functions: the first performs the sum of oil production grouped by the attribute year; and the second computes the average of all years.

Recall that GLAMORISE uses SQLite to store the metadata and process the aggregation over the result set. At present, SQLite does not support nested aggregation functions, such as "AVG(SUM(*field*))", so example question had to be translated into a nested query:

```
SELECT AVG(sum_oil_production) as avg_sum_oil_production
FROM(SELECT SUM(oil_production) as sum_oil_production
FROM nlidb_result_set
WHERE state = 'Alagoas'
GROUP BY year)
```

The Preprocessor Module converts the adjective, in the example "yearly", to its corresponding noun, in this case, "year". When the Postprocessor Module receives the NLIDB result set for this type of question from the Interface Module, it also receives information regarding the timescale in which the data is stored (daily, monthly, yearly, etc.). If the question were asked on a different scale, the Postprocessor Module would synthesize the aggregation accordingly (SUM(*field*) and GROUP BY).

Fig. 6 shows how the Preprocessor Module recognizes the sentences and separates the "average" interpretation, which is the normal aggregation that will be made, from the "yearly" interpretation, which is the timescale aggregation that will be made, in the form of a subquery, depending on the timescale that is stored in the database.



Natural Language Query: What was the average yearly production of oil in the state of Alagoas?

spaCy Dependency Parse Tree and Part-of-Speech (POS)

What	was	the	average	yearly	production of oil	in	the	state	of	Alagoas?
what	be	the	average	yearly	production of oil	in	the	state	of	Alagoas
PRON	AUX	DET	ADJ	ADJ	NOUN	ADP	DET	NOUN	ADP	PROPN

Patterns Recognized by GLAMORISE

What was the average yearly production of oil **SUBQUERY SUM** average yearly production of oil **AGGREGATION FUNCTION AVG** in the state of Alagoas?

GLAMORISE Internal Properties
Preprocessor Properties
pre_aggregation_fields = ['oil_production']
pre_aggregation_functions = ['avg']
pre_cut_text = ['average']
pre_prepared_query = 'What was the year production of oil in the state of Alagoas?'
pre_replaced_text = {'yearly': 'year'}
pre_subquery_aggregation_fields = ['oil_production']
pre_subquery_aggregation_functions = ['sum']
pre_subquery_group_by_fields = ['year']
NLIDB Interface Properties
nlidb_interface_sql = 'SELECT year, oil_production FROM ANP WHERE lower(state) = "alagoas"'
Postprocessor Properties
pos_aggregation_fields = ['oil_production']
pos_aggregation_functions = ['avg']
pos_glamorise_sql = 'SELECT avg(oil_production) as avg_oil_production FROM (SELECT sum(oil_production) as oil_production FROM NLIDB_result_set GROUP BY year)'
pos_subquery_aggregation_fields = ['oil_production']
pos_subquery_aggregation_functions = ['sum']

Fig. 6 – Aggregations with timescale differences example

5. EXPERIMENTS

This section describes the experiments carried out to validate GLAMORISE. Both experiments used an Intel(R) Core$^{(TM)}$ i7-4600U CPU @ 2.10GHz / 2.70 GHz with 16 GB of RAM machine, with Ubuntu 20.04.1 LTS over WSL2 with Windows 10 Pro 64 bits operating system. Following [Affolter et al. 2019], which we recall compared 26 NLIDBs, the experiments adopted as evaluation metric the percentage of the test queries that were correctly translated, that is, whose translation to SQL corresponded to the expected SQL queries, as indicated in the benchmarks.

We also tested GLAMORISE with DANKE [García 2020; Izquierdo et al. 2018, 2020; Torres Izquierdo et al. 2020], which is an NLIDB of the Keyword Search (KwS) type and can work with both relational database and triplestore, to prove that GLAMORISE works independently of the structure of the NLIDB database chosen, even though GLAMORISE uses a relational database (SQLite). The experiments are described in [Novello 2021].

5.1 A Proof-of-Concept Experiment

The first experiment used a benchmark with: a simple SQLite (version 3.27.2) database, with just one table and 2.46 MB of data; a set of 22 test questions, with different phrasings; and a mock NLIDB. Table II shows the questions but omits the expected translations to SQL for brevity. The experiment confirmed that 100% of the test questions were correctly preprocessed, including the removal or replacement of aggregation elements, and that the SQL queries with aggregation generated were correct.

Table II. ANP Natural Language Queries (NLQs)

ID	NLQ
Q1	What was the production of oil in the State of Rio de Janeiro?
Q2	What was the average monthly production of oil in the State of Rio de Janeiro?
Q3	What was the average yearly production of oil in the State of Alagoas?
Q4	How many fields are there in the State of Paraná?
Q5	What was the maximum production of oil in the State of Ceará per field?
Q6	What was the minimum gas production in the State of São Paulo per basin?
Q7	What was the average monthly oil production by the operator Petrobras?
Q8	What was the mean yearly gas production per field?
Q9	What was the mean gas production per month per field?
Q10	What was the per month mean gas production per field?
Q11	What was the per field mean gas production per month?
Q12	What was the mean monthly petroleum production by field in the State of Rio de Janeiro?
Q13	What was the mean yearly petroleum production by field by Rio de Janeiro?
Q14	What was the mean gas production per field with production greater than 100 cubic meters?
Q15	What was the mean gas production per basin with production less than 1000 cubic meters?
Q16	Which field has the highest oil production per month?
Q17	Which basin has the highest yearly oil production?
Q18	Which federated state has the lowest gas production?
Q19	Which state of the federation has the lowest gas production?
Q20	What was the average yearly production of oil per field and state in the year 2015?
Q21	What was the average monthly production of oil per field in the State of Rio de Janeiro and the year 2015?
Q22	Give me the operator with the highest number of fields.

5.2 Experiment with the Integration of GLAMORISE with NaLIR

The second experiment used GLAMORISE integrated with NaLIR [Li and Jagadish, H. V. 2014]. NaLIR is the NLIDB that performed best in comparison with 26 NLIDBs [Affolter et al. 2019]. Since NaLIR can answer aggregation questions, the aggregation layer of NaLIR was removed, leaving GLAMORISE to address this issue, while NaLIR addressed the remainder of the issues involved in an NLIDB. To facilitate the integration step, the original Java implementation of NaLIR[3] was not adopted, but a Python version[4,5], the same language in which GLAMORISE was developed. Another important consideration is that originally NaLIR had a layer called Interactive Communicator, responsible for the interaction with the user to refine the answer. The implementation used suppressed this layer, which is in line with the strategy implemented with GLAMORISE.

[3] https://github.com/umich-dbgroup/NaLIR, accessed on Jun/2021.

[4] https://sbbd.org.br/2020/tutorial-1/, accessed on Jun/2021.

[5] https://github.com/pr3martins/nalir-sbbd, accessed on Jun/2021.

As a first experiment with NaLIR, the same 22 proof-of-concept questions over the ANP database were adopted as a baseline. This time, we used a MySQL (version 8.0.21-0ubuntu0.20.04.4) port of the database with 3.5 MB, instead of SQLite, as NaLIR is prepared to work with MySQL.

Table III shows that GLAMORISE correctly answered 22 questions (100%) regarding the part that was under its responsibility, and NaLIR correctly answered 18 queries (~82%) regarding the part that was under its responsibility, leading to a final result of 18 correctly answered queries (~82%).

Table III. Results of GLAMORISE/NaLIR Integration – ANP database

status/NLIDB	GLAMORISE	%	NaLIR	%	Final Result	%
success	22	100%	18	82%	18	82%
failure	0	0%	4	18%	4	18%
Total	**22**	**100%**	**22**	**100%**	**22**	**100%**

Most of the NaLIR errors were due to its inability to perform a particular match or due to an improper match. In Q5, NaLIR incorrectly matched “Ceará” with the “basin” field instead of the “state” field; this occurred because “Ceará” is also the name of a basin, which caused the improper match. In Q17, NaLIR was unable to identify the “basin” field. In Q18 and Q19, NaLIR failed to identify the “state” field.

As a second experiment with NaLIR, we used the Microsoft Academic Search (MAS) database, a MySQL database with 18 tables and 5.0 GB of data. MAS was used to test NaLIR with 193 unique NLQs[6], of which 99 referred to questions with aggregation, but within these, the linguistic and structural patterns recurred repeatedly. Therefore, we chose 17 NLQs, shown in Table IV, representing the universe of questions in the article to evaluate GLAMORISE’s performance.

Of these queries, GLAMORISE correctly answered 17 NLQs (100%) regarding the part under its responsibility (aggregation processing), and NaLIR correctly answered 11 NLQs (~65%) queries, regarding the part that was its responsibility, leading to a final result of 11 NLQs (~65%) correctly answered.

Again, most of the NaLIR errors were due to its inability to perform a particular match or due to an improper match. In Q1**,** it mistakenly exchanged citations for references. In Q7, Q9, and Q10**,** it mistakenly exchanged publications for references. In Q6 and Q14**,** it was unable to join two instances of the author table. Finally, in Q7 and Q9**,** it mistakenly concluded that "Relational Database" should be in the title of the publication and not in the keywords.

Table IV. MAS Natural Language Queries (NLQs)

ID	NLQ
Q1	return me the author in the "University of Michigan" in Databases area whose papers have more than 5000 total citations.
Q2	return me the author in the "University of Michigan" whose papers have the largest number of citations.
Q3	return me the author who has the largest number of papers containing keyword "Relational Database".
Q4	return me the conference that has the largest number of papers containing keyword "Relational Database".
Q5	return me the keyword, which have been used by the largest number of papers in PVLDB.
Q6	return me the number of authors who have cited the papers by "H. V. Jagadish".

[6] https://raw.githubusercontent.com/umich-dbgroup/NaLIR/master/mas_all.nlqs, accessed on Jun/2021.

Q7	return me the number of authors who have more than 10 papers containing the keyword "Relational Database".
Q8	return me the number of citations of "Making database systems usable" in each year.
Q9	return me the number of conferences, which have more than 60 papers containing the keyword "Relational Database".
Q10	return me the number of keywords, which have been used by more than 10 papers of "H. V. Jagadish".
Q11	return me the number of keywords.
Q12	return me the number of papers after 2000 in "University of Michigan".
Q13	return me the number of papers published in PVLDB in each year.
Q14	return me the papers written by "H. V. Jagadish" and "Divesh Srivastava" with the largest number of citations.
Q15	return me the total citations of all the papers in PVLDB.
Q16	return me the total citations of the papers containing the keyword "Natural Language"
Q17	return me the total number of citations of papers in PVLDB before 2005.

Table V. Results of GLAMORISE/NaLIR Integration – MAS database

status/NLIDB	GLAMORISE	%	NaLIR	%	Final Result	%
success	17	100%	11	65%	11	65%
failure	0	0%	6	35%	6	35%
Total	**17**	**100%**	**17**	**100%**	**17**	**100%**

6. DISCUSSION

As summarized in Section 4 and discussed in detail in [Novello 2021], GLAMORISE covers a broad spectrum of *quantitative aggregation types*, including count, sum, average, minimum and maximum, and optionally a group by clause, and a having clause; aggregations with timescale differences; multiple attribute aggregations; aggregations of attributes with compound names; basic recognition of measurement units; and nested aggregations.

GLAMORISE currently does not support *qualitative aggregation functions*, such as *good*, *bad*, *high*, *low*, etc., as discussed in [Abhijeet Gupta 2013; Gupta and Sangal 2013]. In general, database systems do not natively handle qualitative aggregations as there are no standard mappings to SQL aggregation functions; for example, what is good or near for one person is not good or near for another.

As stated earlier, it is outside the scope of this work an advanced treatment of measurement units, such as unit conversions, all the nuances involving ellipsis, all possible cases involving subqueries, in addition to those treated in Section 4. We only dealt with a few cases involving subqueries and only on two levels. For example, one type of subquery not covered but common in natural language is when the outer query has no aggregation, and the nested query has aggregation, such as "Give me the fields that produce more oil than the average production of all fields ".

7. CONCLUSIONS AND FUTURE DIRECTIONS

The main contribution of this article is a generic module, called GLAMORISE, that extends NLIDBs to cope with aggregation questions, on the condition that the result of the NLIDB is, or can be transformed into, a result set in the form of a table. In addition, the article addressed aggregations with some specificities, such as ambiguities, timescale differences, aggregations in multiple attributes, the use of superlative adjectives, basic recognition of measurement units, and aggregations in attributes with compound names.

The article also described a first experiment with GLAMORISE integrated with a mock NLIDB and 22 static questions as a proof-of-concept, and a second experiment with GLAMORISE integrated with NaLIR, the best performing NLIDB in a recent survey. In both experiments, GLAMORISE performed quite satisfactorily.

As future directions, we contemplate dealing with other types of subqueries, such as nested aggregation functions. We also plan to integrate GLAMORISE with others NLIDBs, as well as with database keyword search systems.

REFERENCES

ABHIJEET GUPTA. Complex Aggregates In Natural Language Interface To Databases. *International Institute of Information Technology*, Hyderabad, 2013.

AFFOLTER, K., STOCKINGER, K. AND BERNSTEIN, A. A comparative survey of recent natural language interfaces for databases. *VLDB Journal*, v. 28, n. 5, p. 793–819, 2019.

BHARATI, A., BHATIA, M., CHAITANYA, V. AND SANGAL, R.. Paninian Grammar Framework Applied to English. *South Asian Language Review*, *Creative Books*, New Delhi, 1998.

GARCÍA, G. M. A Keyword-based Query Processing Method for Datasets with Schemas. *D.Sc. Thesis, Department of Informatics, PUC-Rio*, 2020.

GUPTA, A., AKULA, A., MALLADI, D., ET AL. A novel approach towards building a portable NLIDB system using the computational Paninian grammar framework. *Proceedings of the 2012 International Conference on Asian Language Processing - IALP 2012*, p. 93–96, 2012.

GUPTA, A. AND SANGAL, R.. A Novel Approach to Aggregation Processing in Natural Language Interfaces to Databases. *Proceedings of the 10th International Conference on Natural Language Processing - ICON-2013*, 2013.

HONNIBAL, M. AND JOHNSON, M. An improved non-monotonic transition system for dependency parsing. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, p. 1373–1378, 2015.

HONNIBAL, M. AND MONTANI, I. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. *Unpublished software application*. https://spacy.io, 2017.

IZQUIERDO, Y. T., GARCÍA, G. M., MENENDEZ, E. S., ET AL. QUIOW: A keyword-based query processing tool for RDF datasets and relational databases. *Lecture Notes in Computer Science* - LNCS, v. 11030, p. 259–269, 2018.

IZQUIERDO, Y. T., GARCÍA, G. M., NOVELLI, B. A., ET AL. Integrating a geomechanical collaborative research portal with a data & knowledge retrieval platform. *Rio Oil and Gas Expo and Conference*, v. 20, p. 421–422, 2020.

LI, F. AND JAGADISH, H. V. NaLIR: An interactive natural language interface for querying relational databases. *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, Association for Computing Machinery, p. 709–712, 2014.

LI, F. AND JAGADISH, H. V. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, v. 8, n. 1, p. 73–84, 2014.

LI, F. AND JAGADISH, H. V. Understanding Natural Language Queries over Relational Databases. *ACM SIGMOD Record*, v. 45, n. 1, p. 6–13, 2016.

NOVELLO, A. F. A Novel Solution to Empower Natural Language Interfaces to Databases (NLIDB) to Handle Aggregations. *M.Sc. Dissertation, Department of Informatics, PUC-Rio*, 2021.

NOVELLO, A. F. AND CASANOVA, M. A. A Novel Solution for the Aggregation Problem in Natural Language Interface to Databases (NLIDB). *Proceedings of the XXXV Brazilian Symposium on Databases - SBBD 2020*, Sociedade Brasileira de Computação - SBC. 2020.

PAZOS R, R. A., AGUIRRE L, M. A., GONZÁLEZ B, J. J., ET AL. Comparative study on the customization of natural language interfaces to databases. *SpringerPlus*, v. 5, n. 1, p. 553, 2016.

PAZOS R, R. A., VERASTEGUI, A. A., MARTÍNEZ F, J. A., CARPIO, M. AND GASPAR H, J. Translation of natural language queries to SQL that involve aggregate functions, grouping and subqueries for a natural language interface to databases. *Studies in Computational Intelligence*, v. 749, p. 431–448, 2018.

PINHEIRO, J. P. V, CASANOVA, M. A. AND MENENDEZ, E. S. Improving the Quality of the User Experience by Query Answer Modification. *Proceedings of the XXXV Brazilian Symposium on Databases - SBBD* 2020. Sociedade Brasileira de Computação - SBC.

PRUSKI, P., LOHAR, S., GOSS, W., RASIN, A. AND CLELAND-HUANG, J. TiQi: Answering unstructured natural language trace queries. *Requirements Engineering*, v. 20, n. 3, p. 215–232, 2015.

SHAH, V., LI, S., KUMAR, A. AND SAUL, L. SpeakQL: towards speech-driven multimodal querying of structured data. *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, Association for Computing Machinery, p. 2363–2374, 2020.

TATA, S. AND LOHMAN, G. M. SQAK: Doing more with keywords. *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, Association for Computing Machinery, p. 889–902, 2008.

IZQUIERDO, Y.T., GARCIA, G.M., LEMOS, M., ET AL. Keyword Search over the COVID-19 Data. *Proceedings of the XXXV Brazilian Symposium on Databases - SBBD* 2020, Sociedade Brasileira de Computação - SBC, 2020.