# A Novel Solution for the Aggregation Problem in Natural Language Interface to Databases (NLIDB)

**Alexandre F. Novello, Marco A. Casanova**

Department of Informatics, PUC-Rio, Rio de Janeiro, RJ – Brazil

{anovello, casanova}@inf.puc-rio.br

***Abstract.*** *Natural Language Interface to Databases (NLIDB) systems usually do not deal with aggregations, which can be of two types: aggregation functions (such as count, sum, average, minimum, and maximum) and grouping functions (GROUP BY). This paper addresses the creation of a generic module, to be used in NLIDB systems, that allows such systems to perform queries with aggregations, on the condition that the query results the NLIDB returns are or can be transformed into tables. The paper covers aggregations with specificities, such as ambiguities, timescale differences, aggregations in multiple attributes, the use of superlative adjectives, basic unit measure recognition, and aggregations in attributes with compound names.*

## 1. Introduction

Question Answering (QA) is a field of study dedicated to building systems that automatically answer questions asked in natural language. The translation of a question asked in natural language into a structured query in a database is also known as Natural Language Interface to Databases (NLIDB). To obtain the structured query (SQL or SPARQL) used to query the database, the question in natural language goes through several processing steps such as: (1) question analysis; (2) phrase mapping; (3) disambiguation; and (4) query construction.

One of the less well-solved tasks in step 4 of this process is the treatment of questions with aggregation involving grouping (GROUP BY clause) and aggregation functions such as count, sum, average, minimum and maximum, especially when the question is on another timescale with respect to the stored data, which call for a scale conversion. For example, consider the question: *"What was the average annual compensation for employees in 2019?"* If the compensation data is stored on a weekly scale, it is necessary to understand that the question is on an annual scale and perform the conversion. Note that it is not enough to multiply the average weekly compensation by 52, as the salary may have changed over the year, as well as due to other events, such as vacations, or bonuses. It is necessary to filter the sum of all 2019 compensation data per employee and only then perform the average.

NLIDB systems usually do not deal with aggregations, but they produce good results for normal queries. The contribution of this paper is the creation of a generic module to be used with NLIDB systems, called GLAMORISE (GeneraL Aggregation MOdule for RelatIonal databaSEs). This module allows NLIDB systems to perform queries with aggregations, on the condition that the result of the NLIDB is or can be transformed into a table. Hence, GLAMORISE can even be used with Triplestore (RDF Store) NLIDBs, with the proviso that the result is presented in a tabular format.

GLAMORISE also deals with some aggregations with specificities, including ambiguities, time-scale differences, aggregations in multiple attributes, and aggregations in attributes with compound names.

The rest of this paper is organized as follows. Section 2 contains a brief literature review. Section 3 describes GLAMORISE, its features and limitations. Section 4 evaluates the performance of GLAMORISE against benchmark questions. Finally, Section 5 contains the conclusions and suggestions for future work.

## 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. It uses the concept of Simple Query Network (SQN), which is similar to the Steiner Tree, but gives better results. Our work and SQAK deal with similar problems, aiming at the use of keywords for the final translation to SQL. The difference is that SQAK is a complete and monolithic NLIDB, while ours extends an NLIDB to handle aggregations.

NaLIR [Li and Jagadish, H. V. 2014; Li and Jagadish 2016; Li and Jagadish, H. V 2014] is a generic NLIDB that handles aggregations, nesting, and various types of joins. The approach followed is to take feedback from the user, returning the adjusted parse trees in the form of natural language, so that they can 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 it gives the impression of 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. In our work, we prefer the approach of generating a structured query automatically from the natural language query.

Gupta et al. (2012) presented a novel approach to build NLIDB based on the use of Computational Paninian Grammar (CPG) [Bharati et al. 2014] in which the relationships are syntactic-semantic. CPG was originally developed for Indian languages and afterward received an English version. They argue that using this technique makes the trees more semantic than other kinds of dependency trees, making them easier to map to a SQL. In the following article [Gupta and Sangal 2013], the framework is 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 aggregations. 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 Gupta's master's dissertation [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.

The following contributions were also developed in two stages. First, an NLIDB was created without the ability to process aggregations (or subqueries) [Pazos R et al. 2016], called ITCM NLIDB, and a second work [Pazos R et al. 2018] added a module capable of carrying out these activities. 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 them.

## 3. GLAMORISE – A Proposed Solution

NLIDB systems usually do not deal with aggregations, but they return good results for normal queries. This paper introduces GLAMORISE, a generic module that allows NLIDB systems to perform queries with aggregations, as long as the result is a table. Figure 1 shows the architecture of GLAMORISE.

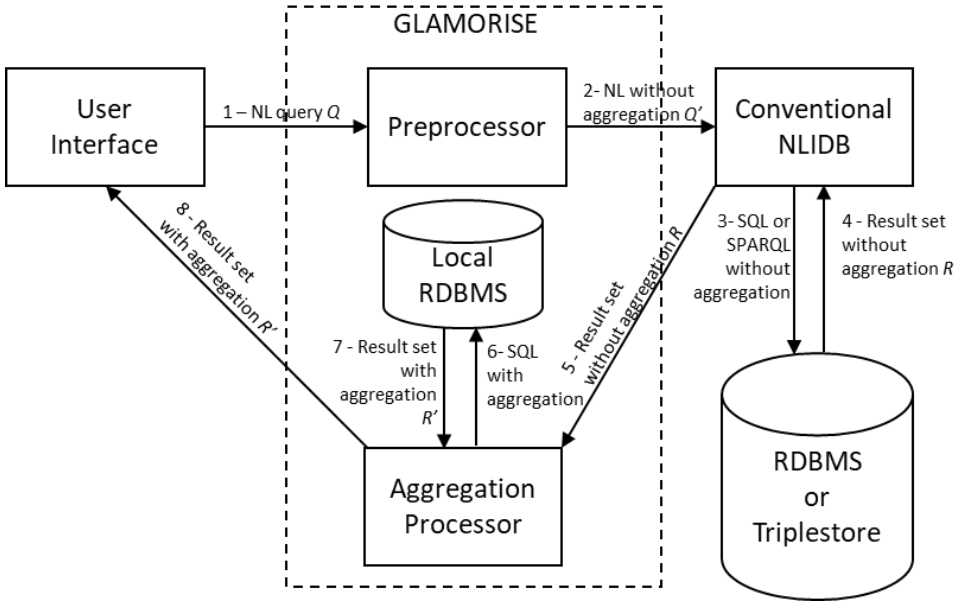


**Fig. 1. GLAMORISE Architecture**

Initially, the user types a query *Q* in natural language in the user interface. The GLAMORISE **Preprocessor** removes the aggregation elements and transforms *Q* into a query without aggregation *Q'* in natural language and registers all the elements related to the aggregation, to be subsequently used by the **Aggregation Processor**. Then, *Q'* is sent to the conventional NLIDB, where the query is processed by the NLIDB and converted into a structured query, being 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. Additionally, the metadata of the data result set can be retrieved to process more complex questions. Following this, the GLAMORISE **Aggregation Processor** stores *R* as a table in a local RDBMS (SQLite). Then, it processes the aggregation over the stored result set *R* by creating a SQL query, resulting in the final result set with aggregations *R'*, and returns it to the user interface.

The purpose of the **Preprocessor** is to map the keywords in natural language to the respective aggregation functions. The other primary mission is to identify whether the query in natural language also has grouping (GROUP by clause) conditions. These keywords are removed or substituted from the query to guarantee that the conventional NLIDB will not be confused by their existence, leading to incorrect mappings.

The **Aggregation Processor** is responsible for building the query itself in SQL, being responsible for analyzing the metadata saved in the Preprocessor stage, and including the aggregation functions (sum, max, min, avg and count), as well as recognizing the fields in which these functions should be applied in the received result set. Then, an identical process is undertaken for grouping (GROUP BY clause), reading the metadata to determine if there is grouping, which fields are involved, mapping them

in the result set, and also if there is a HAVING clause with conditions. Another step is conducted to analyze any timescales that should be converted with the use of an aggregation function (sum) and grouping.

Lastly, this work focuses on treating aggregations with some types of ambiguities, time-scale differences, aggregations in multiple attributes, aggregations with the use of superlative adjectives, basic unit measure recognition, and aggregations in attributes with compound names. It is beyond the scope of this work to deal with elliptical aggregations and queries with subqueries using aggregations (other than the time-scale problem, which was properly addressed). Also, it does not deal with qualitative functions, such as *good*, *bad*, *high*, *low*, which have a subjective component.

The GLAMORISE prototype was implemented in Python[1] with the help of the library spaCy[2], which is an open-source software library for advanced natural language processing. In our work, the spaCy functionalities of POS tagging and parse tree are used.

## 4. Proof-of-concept

To test the performance of GLAMORISE, we implemented a mock NLIDB prepared to receive the set of testing questions with completely different phrasings. First, we confirmed that the questions were preprocessed correctly, removing or substituting the expected words. Second, we confirmed that the generated SQL queries with aggregation were correct. GLAMORISE answered all 21 questions correctly. Two, among all types of aggregations addressed in this work, are discussed in what follows.

**Superlative Adjectives**. The superlative adjectives are suppressed and, depending on their type, a *min* or *max* function is added to the aggregation functions metadata, as well as the respective aggregation field. The superlative adjective is then removed from the query since the NLIDB cannot handle such terms. Figure 2 shows an example. The *min* aggregation function is identified because of the presence of the superlative adjective "lowest"; also, "gas production" is identified as the aggregation field that is related. The relation is obtained by the parse tree.

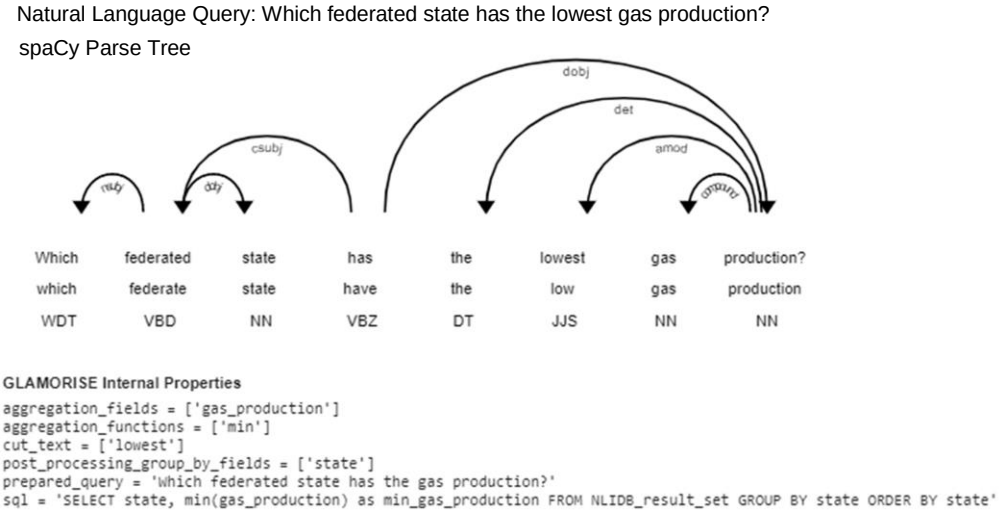


**Fig. 2. Superlative adjective example**

[1] https://www.python.org/
[2] https://spacy.io/

**Aggregations with time-scale differences**. To the best of our knowledge, this is a special condition in aggregation for which we did not find a solution in other works. The problem becomes apparent when the data is stored in one time-scale, and the question is asked in another. An example could be: *"What was the average yearly production of oil in the state of Alagoas?"*. The problem will arise if the data stored in the table is on a monthly basis. Assume that the database has two attributes, one for the year and another for the month, in addition to production (oil or gas). That is each tuple associates production to one year and one month. The SQL of this query 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 example since it involves two aggregation functions: the first performs the sum of the grouped attribute, "year", and then the average of all years is calculated. For convenience, in this first implementation, we are using the SQLite to store the result set returned by the NLIDB and process aggregations. At the moment, the SQLite does not support nested aggregation functions like "AVG(SUM(*field*))", so our query has to be translated to:

```
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)
```

In the case of the example "yearly", the **Preprocessor** converts the adjective to its corresponding noun, in this case, "year". When we receive the NLIDB result set for this type of question, we also receive a metadata result set with information regarding the timescale in which the data is stored (daily, monthly, yearly, etc.). If the question was asked on a different scale, the **Processor** does the aggregation accordingly (SUM(*field*) and GROUP BY). Figure 3 shows how the **Preprocessor** recognizes the sentences and separates the "average" interpretation, which is the normal aggregation that will be made, from the "yearly" interpretation, which is the time-scale aggregation that will be made, depending on the time-scale that is stored in the database.

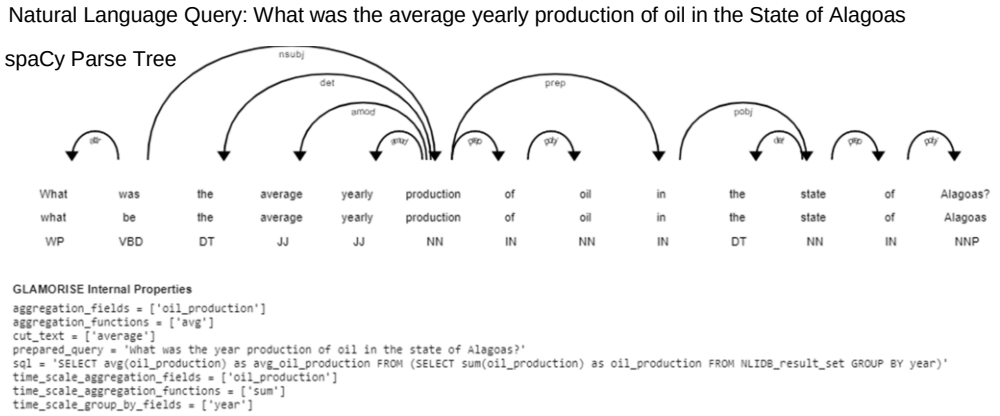


**Fig. 3. Aggregations with time-scale differences example**

## 5. Conclusions and Future Work

The main contribution of this work was the creation of a generic module, called GLAMORISE, to be used in NLIDB systems, which allows the processing of queries with aggregations, on the condition that the result of the NLIDB is, or can be transformed into, a result set in the form of a table. The work addressed aggregations with some specificities such as ambiguities, time-scale differences, aggregations in multiple attributes, the use of superlative adjectives, basic unit measure recognition, and aggregations in attributes with compound names.

As future directions, we contemplate dealing with more complex unit measure recognition, followed by tackling subqueries and elliptical aggregations. Ellipsis is the suppression of a term that can be easily understood by the linguistic context or situation. We also consider using questions from previous QALD challenges as a benchmark.

## 6. References


Abhijeet Gupta (2013). Complex Aggregates In Natural Language Interface To Databases. International Institute of Information Technology, Hyderabad.

Bharati, A., Bhatia, M., Chaitanya, V. and Sangal, R. (2014). Paninian Grammar Framework Applied to English South Asian Language Review, Creative Books, New Delhi, 1998.

Gupta, A., Akula, A., Malladi, D., et al. (2012). A novel approach towards building a portable NLIDB system using the computational Paninian grammar framework. Proc. 2012 Int'l. Conf. on Asian Language Processing, IALP 2012, p. 93–96.

Gupta, A. and Sangal, R. (2013). A Novel Approach to Aggregation Processing in Natural Language Interfaces to Databases. Proc. 10th International Conference on Natural Language Processing - ICON-2013.

Li, F. and Jagadish, H. V. (2014). NaLIR: An interactive natural language interface for querying relational databases. Proc. 2014 ACM SIGMOD International Conference on Management of Data, Snowbird, Utah, USA (June 2014), p. 709–712.

Li, F. and Jagadish, H. V (2014). Constructing an interactive natural language interface for relational databases. Proc. of the VLDB Endowment, v. 8, n. 1, p. 73–84.

Li, F. and Jagadish, H. V (2016). Understanding Natural Language Queries over Relational Databases. ACM SIGMOD Record, v. 45, n. 1, p. 6–13.

Pazos R, R. A., Aguirre L, M. A., González B, J. J., et al. (2016). Comparative study on the customization of natural language interfaces to databases. SpringerPlus 5, 553.

Pazos R, R. A., Verastegui, A. A., Martínez F, J. A., Carpio, M. and Gaspar H, J. (2018). Translation of natural language queries to SQL that involve aggregate functions, grouping and subqueries for a natural language interface to databases. In: Fuzzy Logic Augmentation of Neural and Optimization Algorithms: Theoretical Aspects and Real Applications. Studies in Computational Intell., vol 749. Springer, Cham, p. 431–448.

Tata, S. and Lohman, G. M. (2008). SQAK: Doing more with keywords. Proc. of the 2008 ACM SIGMOD International Conference on Management of Data, Vancouver Canada (June 2008), p. 889–901.