

# A Platform for Keyword Search and its Application for COVID-19 Pandemic Data

Yenier T. Izquierdo<sup>2</sup>, Grettel M. García<sup>2</sup>, Melissa Lemos<sup>2</sup>, Alexandre Novello<sup>1</sup>, Bruno Novelli<sup>2</sup>, Cleber Damasceno<sup>2</sup>, Luiz André P.P. Leme<sup>3</sup>, Marco A. Casanova<sup>1</sup>

<sup>1</sup> Department of Informatics, PUC-Rio, Rio de Janeiro, RJ – Brazil  
{anovello, casanova}@inf.puc-rio.br

<sup>2</sup> Tecgraf Institute, PUC-Rio, Rio de Janeiro, RJ – Brazil  
{ytorres, ggarcia, melissa, bnovelli, cleberoli}@tecgraf.puc-rio.br

<sup>3</sup> Institute of Computing, UFF, Niterói, RJ – Brazil  
lapaesleme@ic.uff.br

**Abstract.** Keyword search is typically associated with information retrieval systems. However, recently, keyword search has been expanded to relational databases and RDF datasets, as an attractive alternative to traditional database access. This paper introduces DANKE, a platform for keyword search over databases, and discusses how third-party applications can be equipped with DANKE to take advantage of a data retrieval mechanism that does not require users to have specific technical skills for searching, retrieving and exploring data. The paper ends with the description of an application, called *CovidKeyS*, which uses DANKE to implement keyword search over three COVID-19 data scenarios.

Categories and Subject Descriptors: H.2 [Database Management]: Miscellaneous; H.3 [Information Storage and Retrieval]: Miscellaneous; I.7 [Document and Text Processing]: Miscellaneous

Keywords: COVID-19 data, Platform, Keyword search, SQL

## 1. INTRODUCTION

After the outbreak of the COVID-19 pandemic, a large variety of COVID-19 data became publicly available. The COVID-19 Data Repository maintained by the Center for Systems Science and Engineering (CSSE) at John Hopkins University is perhaps the worldwide reference repository of COVID-19 data [Dong et al. 2020]. In Brazil, the Ministry of Health and other public and private institutions maintain detailed data, including demographic, clinical and lab exams, and hospitalization data<sup>1</sup>. However, some of the datasets are sufficiently large to be cumbersome to process with the usual desktop spreadsheet tools. A perhaps more robust approach would be to: (1) store the data in a standard DBMS; (2) define a query that retrieves the data the user is interested in; (3) export the query results to a data analysis tool.

Typically, there are two alternatives to implement the second step. The first alternative is to implement a traditional filter form that users must fill with their search parameters specifying exact data, such as a “patient disease” or a “country name”. The second alternative is to offer an interface that allows users to write queries based on a query language, such as SQL (for relational databases) or SPARQL (for RDF graphs). Both styles have limitations since users need to be aware of the data structure, in both cases, and they must also know the syntax of queries, in the second case.

---

<sup>1</sup><https://opendatasus.saude.gov.br/dataset/casos-nacionais>

A third alternative would be to offer database keyword search. This is an attractive alternative since users do not need to fill numerous “boxes” (in the first approach) and do not need to know the way data are organized and the syntax of the query language (in the second approach). Users specify a few terms, called *keywords*, and it is up to the system to access the database and retrieve the data that best matches the list of keywords [Izquierdo et al. 2018].

The main contribution of this article is to present DANKE, a platform for database keyword search, and to describe how third-party applications can be equipped with DANKE to take advantage of a data retrieval mechanism that does not require users to have specific technical skills for searching, retrieving and exploring data. The paper first covers the architecture of DANKE, including the core module with the keyword search engine, and the components that support users to execute their searches and to interpret the results obtained. It also discusses how to prepare a dataset to be used with DANKE, dealing with data ingestion and data enrichment issues. Finally, it details how the communication between third-party applications and DANKE is performed, via service requests to the DANKE API.

Then, the article describes a Web application, called *CovidKeyS*, that uses DANKE services to implement keyword queries over COVID-19 pandemic data. The application is deployed over three scenarios, which represent an extension of the results reported in [Izquierdo et al. 2020]. The first scenario refers to data obtained from the Brazilian NSG (*Notificações de Síndrome Gripal*) dataset published by the Ministry of Health of Brazil, the second scenario to the open data published by the COVID-19 Data Sharing/BR initiative, and the third scenario to datasets downloaded from the John Hopkins University (JHU) dataset, the Google COVID-19 Public Datasets program, and the World Bank Global data.

The paper is organized as follows. Section 2 briefly reviews related work. Section 3 describes the DANKE platform. Section 4 describes *CovidKeyS* and illustrates its use in three COVID-19 data scenarios. Finally, Section 5 contains the conclusions and suggestions for future research.

## 2. BRIEF REVIEW OF RELATED WORK

This section first briefly reviews work related to database keyword search and then lists some efforts to collect COVID-19 Pandemic data.

A survey of keyword query processing tools over relational databases and RDF datasets is given by [Bast et al. 2016]. Tools, such as those described in [Bergamaschi et al. 2016; Oliveira et al. 2015; Vinay and Haritsa 2020], explore the foreign/primary keys declared in the relational schema to compile a keyword query into an SQL query with a minimal set of join clauses. Ramada et al. (2020) also explored relational schema information to compile keywords into SQL queries over databases exposed on the Web. They proposed, in [Ramada et al. 2020], a system called SQUIRREL that selects and ranks relational databases on the Web, based on the metadata the databases expose, pre-processes the keywords, which includes identifying aggregation functions the keywords might express, and compiles the pre-processed keywords into SQL queries.

RDF keyword query processing tools can be schema-based, when they exploit the RDF schema to compile a keyword query into a SPARQL query, or graph-based, when they directly explore the RDF dataset or summaries thereof. Examples of graph-based tools can be found in [Han et al. 2017; Tran et al. 2009]. Dosso and Silvello (2020) proposed the TSA+BM25 and the TSA+VDP keyword search systems over RDF datasets, based on a different approach, that they called “virtual documents”. These systems move most of the computational complexity off-line and then exploit highly efficient text retrieval techniques and data structures to carry out the online phase. The authors, in [Dosso and Silvello 2020], show that these approaches are more efficient and effective, when compared with state-of-the-art systems.

QUIOW [Izquierdo et al. 2018; García et al. 2017] is a fully automatic, schema-based tool that supports keyword-based query processing for both the relational and RDF environments. The tool also constructs a Steiner tree that covers a set of nodes (relation schemes or RDF classes) whose instances match the largest set of keywords and incorporates a backtracking step to further expand the keyword-query results by generating alternative (SQL or SPARQL) queries. In particular, these references describe experiments with QUIOW which show that the techniques used to process keyword-based queries scale to large RDF datasets with 200-300 million triples.

SPARK [Zhou et al. 2007] uses techniques, such as synonyms from Wordnet and string metrics, to map keywords to knowledge base elements. Rihany et al. (2018) also explores Wordnet and proposes a ranking method to implement keyword search over RDF graphs.

Finally, an approach for keyword search for temporal RDF graphs that automatically compiles keyword queries into a set of candidate SPARQL queries was presented in [Gkirtzou et al. 2015]. To support temporal exploration, the method is enriched with temporal operators allowing the user to explore data within predefined time ranges. The novelty of the approach lies exactly in this enrichment.

To name a few repositories, the COVID-19 Data Repository maintained by the Center for Systems Science and Engineering (CSSE) at John Hopkins University is perhaps the worldwide reference repository of COVID-19 data [Dong et al. 2020]. The Academic Data Science Alliance also collects data and data science resources related to the COVID-19 pandemic<sup>2</sup> and the Copyright Clearance Center maintains a list of links to COVID-19 data resources<sup>3</sup>. The Google Cloud services created the COVID-19 Public Datasets program<sup>4</sup> to make COVID-19 data more accessible to researchers. This effort includes the BigQuery sandbox, which allows free queries over certain COVID-related datasets. In Brazil, the Ministry of Health maintains a specific dataset, which we will refer to as NSG (*Notificações de Síndrome Gripal*), with notifications of suspected COVID-19 cases<sup>5</sup>, and a more comprehensive database, called SRAG 2020<sup>6</sup>, with Severe Acute Respiratory Syndrome (SARS) and COVID-19 data. The COVID-19 Data Sharing/BR initiative published open data with demographic, clinical, and laboratory data about patients tested for COVID-19 in the State of São Paulo [Mello et al. 2020]. Finally, with the start of vaccination against COVID-19 at the end of 2020, the online scientific publication *Our World in Data* (OWID) began to collect vaccination data from official country reports [Roser et al. 2020].

### 3. AN OVERVIEW OF THE DANKE PLATFORM

This section introduces the DANKE platform<sup>7</sup>, developed at the Tecgraf/PUC-Rio Institute. Section 3.1 briefly describes the architecture of DANKE. Section 3.2 outlines the dataset preparation process. Section 3.3 focuses on the *Keyword Search* module. Section 3.4 describes the *Assistant* module that helps users perform keyword queries. Section 3.5 explains the functionalities of the *Analysis* module. Finally, Section 3.6 summarizes the API REST that exposes the DANKE functionalities to third-party applications.

#### 3.1 Architecture

Figure 1 shows an overview of the DANKE architecture. It has three main components: 1) *Dataset Preparation*; 2) the *DANKE Dataset*; and 3) *Data and Knowledge Extraction*.

<sup>2</sup><https://academicdatascience.org/covid>

<sup>3</sup><http://www.copyright.com/coronavirus-covid-19-data/>

<sup>4</sup><https://console.cloud.google.com/marketplace/product/bigquery-public-datasets/covid19-public-data-program?pli=1>

<sup>5</sup><https://opendatasus.saude.gov.br/dataset/casos-nacionais>

<sup>6</sup><https://opendatasus.saude.gov.br/dataset/bd-srag-2020>

<sup>7</sup><https://danke.tecgraf.puc-rio.br/>

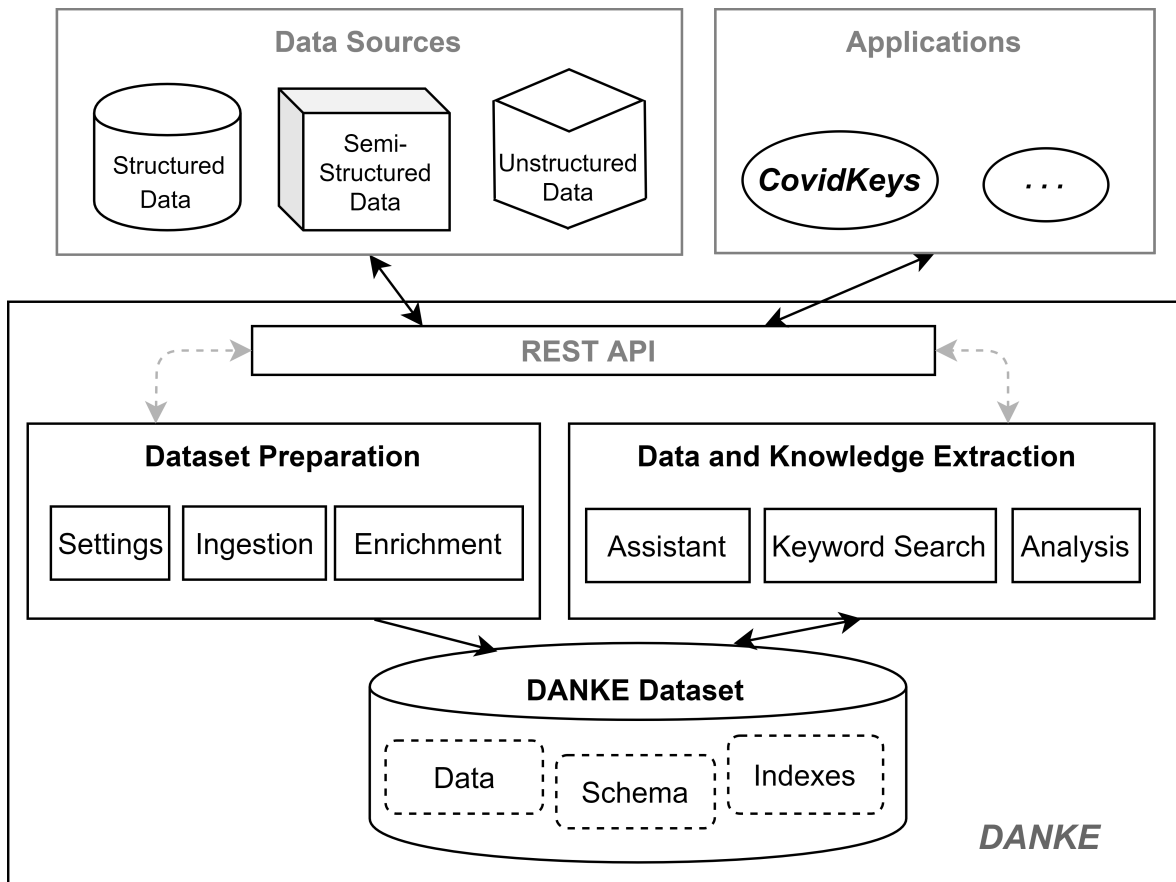


Fig. 1: Architectural Overview of the DANKE Platform

The *DANKE Dataset* component consists of a relational database or an RDF dataset, both in a centralized environment. It defines the data to be accessed and queried, the data indexes, and the schema metadata. The *Dataset Preparation* component is responsible for preparing the *DANKE Dataset* allowing the *Data and Knowledge Extraction* component to work.

Then, the *Data and Knowledge Extraction* component currently provides mechanisms to help users, that do not have specific technical skills, search, retrieve, explore and summarize data from datasets. This component consists of three main modules: 1) the *Keyword Search* module (see Section 3.3); 2) the *Assistant* module (see Section 3.4); and 3) the *Analysis* module (see Section 3.5).

Finally, interested third-party applications can equip their systems with DANKE through service requests for the DANKE REST API (Application Programming Interface), described in Section 3.6.

### 3.2 The DANKE Dataset and Dataset Preparation Components

The DANKE Dataset is a centralized repository that stores the data indexes and additional metadata required to support keyword search and other services. The original datasets may be replicated and stored in the DANKE dataset, or they may remain in their original location. In the second case, the schema metadata includes a reference to where the dataset is physically located. The data indexes are created mostly on string data type attributes, which enable the matching process of the Keyword Search module. Furthermore, the database schema is compiled into an abstract schema, which is independent of the model (relational or RDF) of the underlying database, and stored in the DANKE dataset.

In more detail, the Dataset Preparation component has three main modules: 1) *Data Settings*; 2) *Data Ingestion*; and 3) *Data Enrichment*. The *Data Settings* module aims at setting up and maintaining the DANKE dataset component. This process includes, among other tasks, defining the DANKE dataset schema, adding metadata to this schema, indicating which attributes will have their values indexed, and keeping the DANKE dataset updated incrementally after schema or data changes. Also, it keeps the correspondences between the abstract schema and the data. In turn, the attribute values indexation allows specifying the terms against which the Keyword Search module will match the keywords. This module needs to be executed at least once.

The *Data Ingestion* module aims at populating the data in the DANKE dataset from the data sources. The supported data source formats are:

- (1) *structured*: data that are highly normalized with common schema and stored in relational databases (these data are easily accessible through SQL or data extraction tools);
- (2) *semistructured*: data that contain identifiers without conforming to a predefined schema, often stored in No-SQL databases, such as JSON and XML (these data are easily accessible but require some preparation in order to make them structured);
- (3) *unstructured*: data that do not conform to a data model and are typically stored as individual files (the most common uses are text documents –word, PDF, email–, pictures, audios, and videos).

The *Data Enrichment* module is responsible for making the original data more useful and insightful in order to extract information and knowledge from them. This process enriches data through transformation, categorization, standardization, aggregation, and annotation tasks to facilitate users' search and analysis. If the data is structured, the *Data Enrichment* module is responsible for computing new data and metadata, such as the abstract schema, that DANKE requires to match the metadata in the physical schema and the data in the database with the user's vocabulary. For example, the original values "F" and "M" for "gender" must be considered as "Female" and "Male", since users often perform keyword queries using full words and not abbreviations or acronyms. More complex cases, involving the definition of functions over attributes/values, are also possible. If the data is semi-structured, such as text files, the *Data Enrichment* module can extract metadata from text, applying document interpretation techniques, such as named entity recognition using the closed-world assumption. In other words, using entities already defined in the database, this module can recognize them in text documents. For example, a list of city names already defined in the database can be used to recognize them in a group of text documents. Furthermore, table union and table denormalization are examples of data standardization tasks that are performed by this module to deal with data that are spread over different structured data sources or to increase search engine efficiency.

### 3.3 The Keyword Search Module

The Keyword Search module uses the technology described in [García et al. 2017; Izquierdo et al. 2018; Garcia 2020], operates over both relational databases and RDF datasets, and explores the database schema to compile a keyword query into an SQL or SPARQL query that returns data that best match the keywords. It is capable of synthesizing database queries with projections and selections, as well as joins involving several tables, without user intervention.

This section outlines how the Keyword Search module operates, assuming that the underlying database is relational; the processing of RDF datasets is entirely similar.

A *keyword query* is just a list of terms, or *keywords*, that the user wants to search the database for, and may include reserved terms, such as "<", ">", "between". Section 4 provides examples of keyword queries. An *answer* to a keyword query is formatted as a table whose columns (or column names) contain the keyword matches. The answer may be the result of joining several database tables, that is, an answer to a keyword query does not need to be constructed out of a single table.

The Keyword Search module features an algorithm [García et al. 2017] that accepts a keyword query  $K$  over a relational database  $D$ , together with its relational schema  $S$ , and: (1) finds matches with the keywords in  $K$ ; (2) creates a conceptual query by exploring the keyword matches found and the schema  $S$ ; (3) compiles the conceptual query into a structured SQL query, which is then executed. In more detail, with the help of special indexes, the algorithm computes a set of relation schemes and attributes in  $S$  whose metadata (names and descriptions) match keywords in  $K$  and a set of attribute/value pairs whose values match keywords in  $K$ .

The algorithm synthesizes a *conceptual query* that captures the keyword query, using information from the abstract schema and the matching process output. To synthesize a conceptual query, the algorithm implements two heuristics, called the scoring and the minimization heuristics. Briefly, the scoring heuristic: (1) considers how good a match is; (2) assigns a higher score to metadata matches, on the grounds that, if the user specifies a keyword that matches both a relation scheme name (or description) and an attribute value of a tuple, then the user is probably more interested in the scheme than the specific instance; (3) assigns a higher score to relation schemes, called nucleuses, whose attributes cover a larger number of keywords. The minimization heuristic connects the nucleuses using a small number of equijoins. This is equivalent to generating a Steiner tree  $ST$  of the abstract schema graph that covers the nucleuses that were prioritized.

The algorithm creates the `WHERE` clause of the conceptual query using: the filters included in the keyword query to generate selection clauses; the edges of  $ST$  to generate join clauses; the nucleuses to generate additional selection clauses. The final stage of the algorithm is to compile the conceptual query into a concrete SQL query for the underlying relational DBMS. The exact syntax of the concrete SQL query naturally depends on the target DBMS and may take advantage of the specific features the DBMS offers. For instance, the Oracle SQL syntax permits fuzzy text search via the `CONTAINS` operator, which facilitates the implementation of partial matches between keywords and attribute text values. The SQL query is then executed to generate an answer to the keyword query, which is passed to the user.

### 3.4 The Assistant Module

Although it is apparently easier for the users to enter keywords to perform their searches (compared to other approaches such as writing their own queries in the database or depending on the existence of a variety of forms to answer the diverse queries they need), users still have obstacles to choose the most appropriate keywords to get the results they really want. To make this task simpler, users need to know what data is stored in the database and what queries it is ready to answer. For example, the user will not be able to find countries with COVID cases affecting women, if gender information is not stored in the database. It is very useful to make it clear to users what are the main concepts and relationships defined in the database, for example, using simple diagrams and mindmaps.

For this purpose, DANKE has services to automatically support users in choosing their keywords, such as *autocomplete* and *domain hint*. *Autocomplete* recommends keywords based on text already typed by the users. The domain hint returns an ordered list of pairs, consisting of a keyword and its relevance. This can be used to generate, for example, a word cloud, where the size of each word indicates its importance. However, domain hint can be used even more dynamically, allowing specific keywords to be selected, and new related keywords can be presented.

Both *autocomplete* and *domain hint* are based on keyword relevance. This relevance could be computed by combining the importance of the schema elements that the keyword matches and how many times that keyword was already used in searches. The importance of the schema elements may be manually defined by data-domain experts or using automatic approaches, such as [Menendez et al. 2019]. The choice can be configured according to user requirements.

DANKE also offers *user authentication* and *log* services, which allow users to access the history

of executed queries and to set queries as favorites. These features make it easy to re-run previous queries, and adjust the ranking and, consequently, customize the output of *autocomplete* and *domain hint* closer to the user profile.

### 3.5 The Analysis Module

Since a query answer may be a long result set, with many lines, users are not always able to understand it immediately. DANKE offers data analysis approaches that produce an overview and visualizations of the result, which helps users interpret the result, considering attributes in the result individually or in combination.

For each attribute in the result, the overview approach computes the total number of values, the number of distinct values, the most frequent value, the rarest value, the lowest value and the highest value. If the attribute datatype is numerical, average and sum can also be calculated. Moreover, according to each attribute data type, the algorithm also computes a summary, indicating quantities and percentages. For date or numerical datatype attributes, the algorithm distributes the attribute values found over a fixed number of ranges, and calculates the total and the percentage of values in each range. For string datatype attributes, the algorithm creates a range for each value found. If the number of attribute values is greater than the fixed number of ranges, the algorithm creates a range “*other*”, allocating all other values in it. These ranges and their respective values and percentages can be displayed as pie charts, where each interval can be represented as a slice on the chart.

For combinations between attributes, DANKE can offer visualization approaches that automatically generate a set of visualizations (e.g., bar or line charts) from the correlation, transformation, grouping, and ordering of the result attributes. The generation is based on the expert rules proposed by [Luo et al. 2018], where operations over the data that can (possibly) generate meaningful visualizations by pruning bad visualizations (visualizations that humans will never generate or consider). The rules are separated from three perspectives: (1) transformation rules - whether a grouping or binning operation is useful; (2) sorting rules - whether a column should be sorted; and (3) visualization rules - whether a certain type of visualization (pie, line bar) is the right choice. They are based on the attribute types in the result, which are classified as *categorical*, *numerical* or *temporal*. The overall idea is to progressively generate the candidate visualizations by combining the attributes in the query result with the largest possibility to be in the top-k visualizations, based on a partial order of the rules applied to generate the visualizations.

For example, consider a keyword query about the total number of COVID-19 cases in Brazil. Suppose that the query result consists of a table with columns `date` and `total_cases` on that date. Then, the visualization approach may generate a line chart that illustrates the behavior of the total number of cases over time.

### 3.6 The REST API

The communication between an application and DANKE is via service requests to the DANKE REST API (Application Programming Interface), summarized in Table I. A typical sequence of services requests goes as follows.

An application starts by sending a request to DANKE using the method `/search/queries` (line 1) with a list of keywords as input and receives a conceptual query (in SQL or SPARQL) as output, compiled as outlined in Section 3.3. Then, the application calls `/search/results` (line 2) obtaining the query results as output and `/search/query/metadata` (line 3) obtaining the total of lines in the result. Both methods have as input the query, `/search/results` also have as input the paging data (offset and limit). After accessing the results, DANKE allows users to access a specific element and its linked data, stored in the database, using the method `/explorer/resource/id` (line 4), where *id* identifies the desired element.

Table I: DANKE API Services and their Methods Definition

Line	Service	Method
1	Search	GET /search/queries
2		POST /search/results
3		POST /search/query/metadata
4	Explorer	GET /explorer/resource/id
5		GET /explorer/entities
6		GET /explorer/entities/id/properties
7		GET /explorer/properties
8	Analysis	GET /analysis/query/overview
9		GET /analysis/query/top-visualizations
10	Assistant	GET /assistant/autocomplete
11		GET /assistant/domain-hint
12	Exporter	POST /exporter/query/results
13	Logger	POST /login

The user can add other properties to the result. For that, DANKE first recovers the properties of the entities in the conceptual query using the method */explorer/entities/id/properties* (line 6), where *id* is the entity id. Then, it modifies the conceptual query. Finally, it calls the method */search/results* (line 2) again.

Since a query answer may have a result set with many lines, DANKE has built-in data analysis algorithms to help the user understand the result. The method */analysis/query/overview* (line 8) computes a summary of each attribute or set of attributes and the method */analysis/query/top-visualizations* (line 9) automatically computes visualizations about the property values in the result, as outlined in Section 3.5.

DANKE helps users formulate keyword queries through assistant service. The method */assistant/autocomplete* (line 10) receives a text as input and returns an ordered list of keywords to autocomplete the inputted text. The method */assistant/domain-hint* (line 11) recommends an ordered list of queries based on their relevance for the domain that can be used to create, for instance, a word cloud as in Figure 6a, where the size of each word indicates the relevance of the query. This method can also receive a text as parameter, then it returns only the relevance queries that contain that text.

An application may also call the exporter service to obtain the query results in different formats using the method */exporter/query/results* (line 12), passing the conceptual query and the desired format (such as csv or xls) as input parameters.

Authentication mechanisms are not mandatory for applications to work. In these cases, the method */login* (line 13) does not need to be implemented. In contrast, when called, the method returns a token whose validity will always be checked when any of the DANKE services were instantiated for applications that need it.

DANKE also has services to set up and maintain the DANKE dataset, which includes data settings, data ingestion and data enrichment, as it was presented in the previous section.

#### 4. COVIDKEYS – A KEYWORD SEARCH APPLICATION OVER COVID-19 DATA

*CovidKeyS* is a Web Application that uses the DANKE services, described in Section 3.6, to query COVID-19 data and is accessible at the endpoints indicated in what follows. To illustrate *CovidKeyS*, this section describes three COVID-19 data scenarios, with a few sample keyword queries and the compiled SQL queries. *CovidKeyS* runs on top of an Oracle Database server, and the compiled SQL queries included in the next sections follow the Oracle SQL syntax.



## 4.1 Scenario 1: The Brazilian NSG Dataset

As mentioned in Section 2, NSG (*Notificações de Síndrome Gripal*) stores notifications of suspected COVID-19 cases. The original data are openly available in csv format files at <https://opendatasus.saude.gov.br/dataset/casos-nacionais>. Each data file contains the information offered from a Brazilian state to the Ministry of Health. Such data is sufficiently rich to allow relating demographic, clinical, and laboratory data. NSG data was organized in DANKE as a relational database with three tables, *Paciente* (*Patient*), *Teste* (*Exam Lab*), and *Desfecho* (*Evolution*), respectively storing patient data, COVID-19 test data, and how the case evolved.

The Data Enrichment module (described in Section 3.2) was used to transform the original values of the attribute *sexo* (*gender*) of table *Paciente* from “F” and “M” to “Feminino” (*Female*) and “Masculino” (*Male*), respectively. Note that, since the data in this dataset are in Portuguese, users must search using keywords in Portuguese. Figure 2 shows, through a mind-map, some terms presented in the dataset that users can use in their keyword queries. For more details about the dataset or to try keyword queries, we invite the reader to access the application available at <https://danke.tecgraf.puc-rio.br/covid-sus/>.

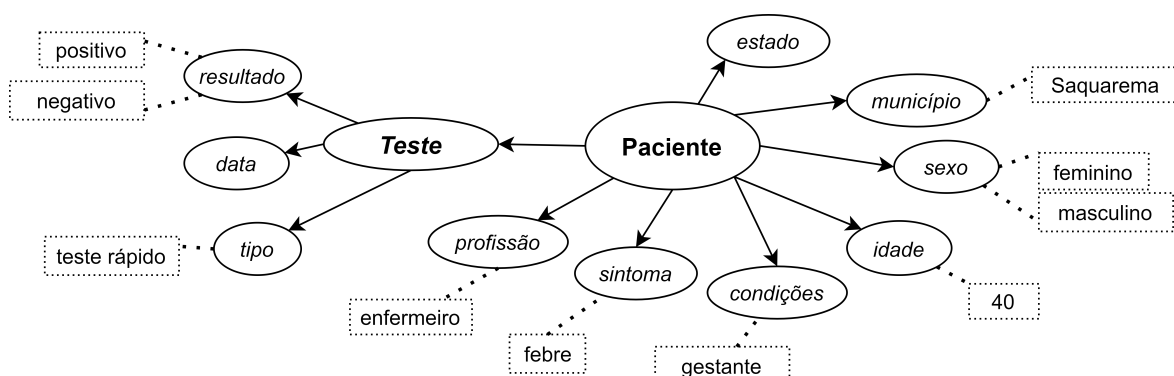


Fig. 2: Mindmap with some NSG dataset terms

Two examples of keyword queries over this dataset are:

- NL (Natural Language) Query: “*Quais foram os casos em Saquarema de gestantes, com sintoma de febre, e idade menor do que 40 anos?*” (“*Which were the cases in Saquarema involving pregnant women, with fever symptoms, and age less than 40 years?*”)
- Keyword query: *saquarema gestante febre idade < 40* (*saquarema pregnant fever age < 40*)
- NL Query: “*Quais enfermeiros, e em que municípios, fizeram teste rápido e tiveram resultado positivo?*” (“*Which nurses in which counties had a rapid test with positive result?*”)
- Keyword query: *enfermeiro município “teste rápido” resultado positivo* (*nurse counties “rapid test” result positive*)

The first keyword query is evaluated over a single table, *Paciente*. The keyword “*saquarema*” exactly matches values of attribute *Paciente.município*, generating the restriction “*Paciente.município = 'Saquarema'*”; the keyword “*gestante*” partially matches values of attribute *Paciente.condicoes*, generating the restriction “*CONTAINS(Paciente.condicoes, 'FUZZY({gestante},60,1,WEIGHT)', 2) > 0*”, and likewise for the keyword “*febre*”, which partially matches values of attribute *Paciente.sintomas*; the last part of the keyword query, “*idade < 40*” generates the restriction “*Paciente.idade < 40*”. The final SQL query that DANKE generates is:

```

SELECT Paciente.id_paciente, Paciente.municipio, Paciente.condicoes,
       Paciente.sintomas, Paciente.idade
FROM Paciente
WHERE Paciente.municipio = 'Saquarema'
      AND CONTAINS(Paciente.condicoes, 'FUZZY({gestante}, 60, 1, WEIGHT)', 2) > 0
      AND CONTAINS(Paciente.sintomas, 'FUZZY({febre}, 60, 1, WEIGHT)', 2) > 0
      AND Paciente.idade < 40

```

However, the second keyword query requires joining two tables, and was compiled into an SQL query as follows. The keyword “*enfermeiro*” matches values of attribute `Paciente.profissao`, generating the restriction “`CONTAINS(Paciente.profissao, 'FUZZY({enfermeiro}, 60, 1, WEIGHT)', 2) > 0`” — note that the keyword “*profissão*” is not required, since attribute `profissao` is inferred from the match; the keyword “*município*” matches the name of attribute `Paciente.municipio`; the keyword “*teste rápido*” (treated as a single term due to the use of double quotes in the keyword query) partially matches values of attribute `Teste.tipo`, generating the restriction: “`CONTAINS(Teste.tipo, 'FUZZY({teste rápido}, 60, 1, WEIGHT)', 2) > 0`”; the keywords “*resultado*” and “*positivo*” were likewise processed. The SQL query is:

```

SELECT Paciente.id_paciente, Paciente.profissao, Paciente.municipio,
       Teste.tipo, Teste.Resultado
FROM Paciente, Teste
WHERE CONTAINS(Paciente.profissao, 'FUZZY({enfermeiro}, 60, 1, WEIGHT)', 2) > 0
      AND CONTAINS(Teste.tipo, 'FUZZY({teste rápido}, 60, 1, WEIGHT)', 2) > 0
      AND CONTAINS(Teste.resultado, 'FUZZY({positivo}, 60, 1, WEIGHT)', 2) > 0
      AND Paciente.id_paciente = Teste.id_paciente

```

Observe that the `SELECT` clause also contains attributes `Paciente.profissao`, `Teste.tipo`, and `Teste.Resultado` to reinforce that each row in the result table indeed indicates a match with the keywords in the keyword query.

#### 4.2 Scenario 2: COVID-19 Data Sharing/BR Dataset

COVID-19 Data Sharing/BR is an initiative of the São Paulo Research Foundation (FAPESP) in collaboration with the University of São Paulo and participation of Fleury Institute, Sírio-Libanês Hospital (SLH), and Israelita Albert Einstein Hospital (IAEH) to provide data related to the pandemic of COVID-19 in the São Paulo state. This initiative makes available, in delimited files format, three types of anonymized data: demographics, clinical and lab exams, and hospitalization. However, we disregard the hospitalization information because (at the time of the writing) only the Israelita Albert Einstein Hospital made such data available.

To use DANKE, we imported these data into a database with two tables and a 1-to-many relationship. The database table `paciente` (*patient*) has 8 attributes: 7 imported from the sources and a new attribute, `age`, computed as `age = 2020 - aa_birthyear`. We decoded the `ic_sex` attribute values: “*M*” for “*Masculino*” (*Male*) and “*F*” for “*Feminino*” (*Female*). The database table `exame` (*lab\_exam*) has nine attributes: eight imported from the sources and a new attribute, `hospital`, whose values indicate the data source. A data cleaning task was performed on the original data to eliminate existing duplicates. Like the case presented in the previous section, the data are in Portuguese. The reader is invited to try keyword queries over these data at <https://danke.tecgraf.puc-rio.br/covid-fapesp/>.

An example of a keyword query is:

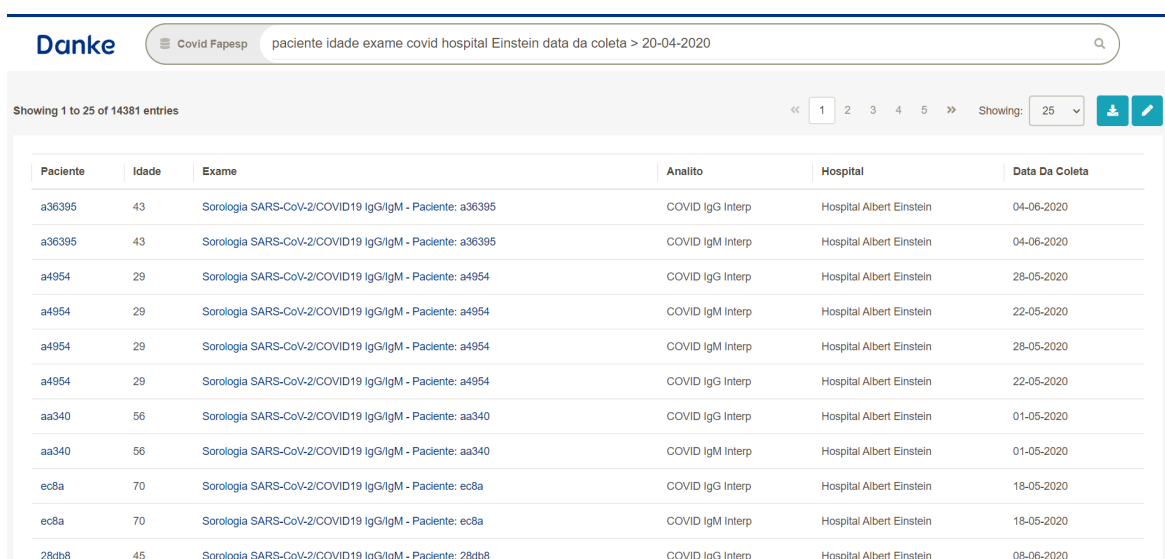
—NL Query: “*Qual é a idade dos pacientes cujo que fizeram exame para COVID no hospital Albert Einstein com data de coleta posterior a 20 de abril de 2020?*” (“*Which was the age of patients whose COVID exams were performed at Albert Einstein Hospital with the collection date after April 20, 2020?*”)

- Keyword query: “*paciente idade exame covid hospital Einstein data da coleta > 20-04-2020*”

The compilation of this keyword query into an SQL query is entirely similar to the second example in Section 3.3, with an equijoin between the two tables, `paciente` and `exame`, over the `id_paciente` attribute. The final SQL query that DANKE generates is<sup>8</sup>:

```
SELECT paciente.label, paciente.idade, exame.label, exame.analito,
       exame.hospital, exame.data_coleta
FROM paciente, exame
WHERE paciente.id_paciente = exame.id_paciente
      AND CONTAINS(exame.analito, 'FUZZY({covid}, 60, 1, WEIGHT)', 2) > 0
      AND CONTAINS(exame.hospital, 'FUZZY({einstein}, 60, 1, WEIGHT)', 2) > 0
      AND exame.data_coleta > DATE('20-04-2020', 'dd-MM-yyyy')
```

When the synthesized SQL query is executed over the DANKE data, the results are shown by the application in tabular form, as illustrated in Figure 3. Note that the result table columns are user-friendly labeled. Also, the columns labeled as `Paciente` and `Exame` represent human-readable identifiers for the resources stored in the respective tables, but recall that the identities of the patients were anonymized. These values are clickable, allowing users to retrieve the attributes and relationships of a target resource. For example, Figure 4 shows the result of clicking on the link that represents the patient identified by “a36395” (the first one in the column `Paciente` in Figure 3). It details, on the right, the attribute values of the element and, below, the element relationships.



Paciente	Idade	Exame	Analito	Hospital	Data Da Coleta
a36395	43	Sorologia SARS-CoV-2/COVID19 IgG/IgM - Paciente: a36395	COVID IgG Interp	Hospital Albert Einstein	04-06-2020
a36395	43	Sorologia SARS-CoV-2/COVID19 IgG/IgM - Paciente: a36395	COVID IgM Interp	Hospital Albert Einstein	04-06-2020
a4954	29	Sorologia SARS-CoV-2/COVID19 IgG/IgM - Paciente: a4954	COVID IgG Interp	Hospital Albert Einstein	28-05-2020
a4954	29	Sorologia SARS-CoV-2/COVID19 IgG/IgM - Paciente: a4954	COVID IgM Interp	Hospital Albert Einstein	22-05-2020
a4954	29	Sorologia SARS-CoV-2/COVID19 IgG/IgM - Paciente: a4954	COVID IgG Interp	Hospital Albert Einstein	28-05-2020
a4954	29	Sorologia SARS-CoV-2/COVID19 IgG/IgM - Paciente: a4954	COVID IgM Interp	Hospital Albert Einstein	22-05-2020
aa340	56	Sorologia SARS-CoV-2/COVID19 IgG/IgM - Paciente: aa340	COVID IgG Interp	Hospital Albert Einstein	01-05-2020
aa340	56	Sorologia SARS-CoV-2/COVID19 IgG/IgM - Paciente: aa340	COVID IgM Interp	Hospital Albert Einstein	01-05-2020
ec8a	70	Sorologia SARS-CoV-2/COVID19 IgG/IgM - Paciente: ec8a	COVID IgG Interp	Hospital Albert Einstein	18-05-2020
ec8a	70	Sorologia SARS-CoV-2/COVID19 IgG/IgM - Paciente: ec8a	COVID IgM Interp	Hospital Albert Einstein	18-05-2020
28db8	45	Sorologia SARS-CoV-2/COVID19 IgG/IgM - Paciente: 28db8	COVID IgG Interp	Hospital Albert Einstein	08-06-2020

Fig. 3: Result of the keyword query “*paciente idade exame covid hospital Einstein data da coleta > 20-04-2020*”

The result of this query (see Figure 3) is relatively large and contains 14,381 entries, which means that a manual analysis would require considerable effort. Thus, the Data Analysis module (described in Section 3.5) is a useful method to create an overview of the query result. Figure 5 shows the result of executing the Data Analysis module for attribute `idade` of the query result. Figure 5a, at the top, shows statistical measures for attribute `idade`: quantity of distinct values, most and least frequent values, min, max and average, recalling that `idade` is a numerical attribute; however, it is naturally not appropriate to sum the values of `idade`. At the down left, Figure 5a depicts a pie chart with the values of the attribute `idade`, where each slice size corresponds to the number of values in the slice range; on the right, the table indicates the age range values and the corresponding percentage of the total number of age values. For example, most patients are between 30 and 41 years old, with a total of 2,038 people, or 31.5%. Figure 5b lists the most common values of attribute `idade` in the result and their respective frequency, in descending order. The most frequent value of age is 38

<sup>8</sup>In this case, *analito*, in Portuguese (*analyte*, in English), refers to the target of the analysis of a lab exam.

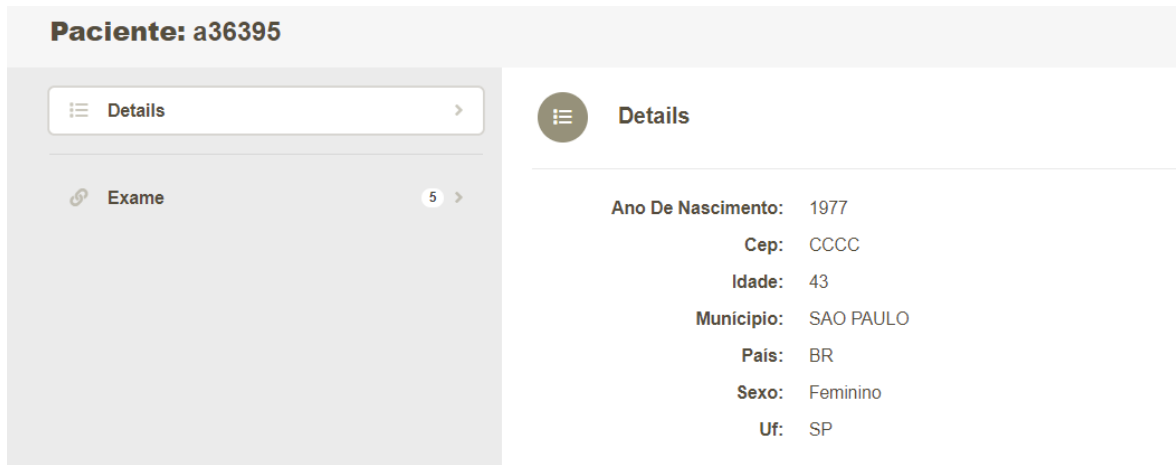
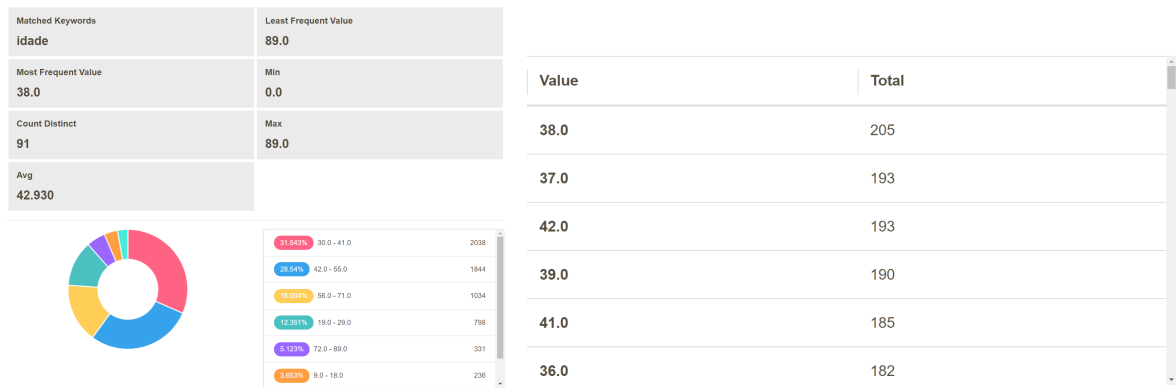


Fig. 4: Exploration panel for the patient identified by “a36395”



(a) Data Summarization and Chart

(b) The most common values and their total appearances

Fig. 5: Result Analysis for attribute *idade* from the keyword query “*paciente idade exame covid hospital Einstein data da coleta > 20-04-2020*”

years old, with 205 appearances. Note that this analysis is always performed over the values of the attributes in the query result and not the attribute values stored in the dataset. Moreover, the user can export the result table as a CSV file and submit it to a different tool for further analysis.

### 4.3 Scenario 3: JHU, Google, and World Bank Global Dataset

To further illustrate the use of *CovidKeyS*, we chose the following country-level aggregated datasets: (1) the John Hopkins University (JHU) dataset, which includes the location and number of confirmed COVID-19 cases, deaths, and recoveries for all affected countries; (2) the Google COVID-19 Public Mobility Report, which reports movement trends over time by geography across different categories of places (park, residential, workplaces); and (3) the World Bank Global data about population, education, and external debt by country.

Contrasting with the two scenarios presented before, the data, in this case, are in English. Figure 6a shows a word cloud with terms present in the dataset to help users run queries. For example, the keyword “*country*” appears in the word cloud, which indicates to the user that searches using country names as keywords can be performed. The autocomplete feature also assists in the execution of the keyword queries. Figure 6b illustrates this functionality. When a user types “*country m*” with the intention of performing a search, the system displays a list of data and metadata values that start with “*m*”. The reader is invited to run keyword queries at <https://danke.tecgraf.puc-rio.br/covid-global/>.

An example keyword query over these data would be:



Fig. 6: Examples of DANKE features

—NL Query: “List the total number of deaths and the recreation mobility index on the same date for Belarus.”

- Keyword query: *Belarus deaths recreation date*

The keyword query was compiled into an SQL query as follows. The keyword “*Belarus*” matches a value of attribute “*Country\_Name*” of table *Country* (from the World Bank global data about population); the keyword “*deaths*” partially matches the name of attribute “*Total\_Deaths*” and the keyword “*date*” matches the name of attribute *Date* of table *Contamination* (from the global synthesis of COVID-19 cases by country); and the keyword “*recreation*” partially matches the name of attribute “*Retail\_and\_Recreation\_Mobility\_Percentage*” of table *Mobility* (from the Google Mobility dataset). From these matches, equijoins between the keys of these tables are inferred (the keys of all three tables are composed of attributes *Country\_Name* and *Date*) to create a coherent answer.

The compiled SQL query would be:

```
SELECT Country.Country_Name, Contamination.Total_Deaths, Contamination.Date,
       Mobility.Retail_and_Recreation_Mobility_Percentage
FROM Country, Mobility, Contamination
WHERE Country.Country_Name = 'Belarus'
      AND Country.Country_Id = Contamination.Country_Id
      AND Contamination.Country_Id = Mobility.Country_Id
      AND Contamination.Date = Mobility.Date
```

A second example would be:

—NL Query: “What are the names, number of deaths, and population of countries with GDP per capita less than 15,000 and number of deaths greater than 5,000?”

- Keyword query: “*country deaths population 'GDP per capita' < 15000 deaths > 5000*”

The compiled SQL query would be (table *distribution* refers to the JHU deaths data and table *wb\_education* refers to the World Bank global education data):

```
SELECT distribution.countries_and_territories, distribution.deaths,
       distribution.pop_data_2019, wb_education.indicator_name, wb_education.value
FROM wb_education, distribution
WHERE wb_education.indicator_name = 'GDP per capita (constant 2005 US$)'
      AND wb_education.value < 15000
      AND distribution.deaths > 5000
      AND wb_education.country_name = distribution.countries_and_territories
```

## 5. CONCLUSIONS

The main contribution of this paper is a platform for keyword search, called DANKE. Unlike the keyword search tools reported in the literature, DANKE does not follow a black box architecture, but it exposes an API

that offers keyword search functionality to third-party database applications, regardless of the data domain. To illustrate the advantages of such architecture, the article describes an application, called *CovidKeyS*, that uses DANKE to search data over three COVID-19 data scenarios.

As for future work, we may point out several directions, based on the fact that the development of an application for COVID-19 cases must solve some classic, non-trivial data integration problems, such as: (1) processing large data volumes; (2) handling a large number of data sources; (3) treating semi- and non-structured data; (4) instance alignment; (5) schema mapping; (6) data cleaning; (7) maintaining several versions of the same data; (8) accounting for the evolution of the conceptual organization of the data; and (9) simplifying access to the data. The implementation of the application *CovidKeyS* addressed just the last problem.

In all three scenarios described in Section 4, manual intervention was required to download, clean, integrate, and organize the data into a standard relational database, tasks that must be repeated on a regular basis to maintain the data up-to-date, in a quite dynamic scenario. Therefore, we first plan to extend DANKE to help automate these steps and to move to a federated databases environment, as in [Izquierdo et al. 2017].

A less obvious extension of the DANKE platform, but better aligned with the main thrust of this article, would be to offer a pseudo-natural language interface that accounted for temporal data, since COVID-19 cases data are time series, and to support temporal data summarization and aggregate operators, backed up by new graphical output features. This extension also involves grouping operators and aggregation functions in the query compilation process, which DANKE currently does not support. We are also currently working on improving the visualization approach included the Analysis module.

#### ACKNOWLEDGMENT

This work was partly funded by grants CAPES 88881.134081/2016-01, CNPq 302303/2017-0, FAPERJ E-26-202.818/2017, and FAPERJ E-26/200.770/2019. This work used data obtained from the COVID-19 Data Sharing/BR, available at <https://repositoriodatasharingfapesp.uspdigital.usp.br/>.

#### REFERENCES

- BAST, H., BJÖRN, B., AND HAUSSMANN, E. Semantic search on text and knowledge bases. *Foundations and Trends in Information Retrieval* 10 (2-3): 119–271, 2016.
- BERGAMASCHI, S., GUERRA, F., INTERLANDI, M., TRILLO-LADO, R., AND VELEGRAKIS, Y. Combining user and database perspective for solving keyword queries over relational databases. *Information Systems* vol. 55, pp. 1–19, 2016.
- DONG, E., DU, H., AND GARDNER, L. An interactive web-based dashboard to track covid-19 in real time. *The Lancet infectious diseases* 20 (5): 533–534, 2020.
- DOSSO, D. AND SILVELLO, G. Search text to retrieve graphs: A scalable rdf keyword-based search system. *IEEE Access* vol. 8, pp. 14089–14111, 2020.
- GARCIA, G. M. *A Keyword-based Query Processing Method for Datasets with Schemas*. Ph.D. thesis, Thesis presented to the Graduate Program in Informatics, PUC-Rio, 2020.
- GARCÍA, G. M., IZQUIERDO, Y. T., MENENDEZ, E., DARTAYRE, F., AND CASANOVA, M. A. Rdf keyword-based query technology meets a real-world dataset. In *Proceedings of the International Conference on Extending Database Technology*. OpenProceedings.org, pp. 656–667, 2017.
- GKIRTZOU, K., KAROZOS, K., VASSALOS, V., AND DALAMAGAS, T. Keywords-to-sparql translation for rdf data search and exploration. In *International Conference on Theory and Practice of Digital Libraries*. Springer, pp. 111–123, 2015.
- HAN, S., ZOU, L., YU, J. X., AND ZHAO, D. Keyword search on rdf graphs—a query graph assembly approach. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (CIKM)*. ACM, pp. 227–236, 2017.
- IZQUIERDO, Y. T., CASANOVA, M. A., GARCÍA, G., DARTAYRE, F., AND LEVY, C. H. Keyword search over federated rdf datasets. In *ER Forum 2017 and ER Demo track co-located with the 36th Int. Conf. on Conceptual Modeling*. CEUR-WS.org, pp. 86–99, 2017.
- IZQUIERDO, Y. T., GARCÍA, G. M., LEMOS, M., NOVELLO, A., NOVELLI, B., DAMASCENO, C., LEME, L. A. P., AND CASANOVA, M. A. Keyword search over covid-19 data. In *35th Edition of the Brazilian Symposium on Databases*. Journal of the Brazilian Computer Society (JBCS), pp. 205–210, 2020.

- IZQUIERDO, Y. T., GARCÍA, G. M., MENENDEZ, E. S., CASANOVA, M. A., DARTAYRE, F., AND LEVY, C. H. Quiow: a keyword-based query processing tool for rdf datasets and relational databases. In *International Conference on Database and Expert Systems Applications (DEXA)*. Springer, pp. 259–269, 2018.
- LUO, Y., QIN, X., TANG, N., AND LI, G. Deepeye: Towards automatic data visualization. *2018 IEEE 34th International Conference on Data Engineering (ICDE)* vol. DOI: 10.1109/ICDE.2018.00019, pp. 101–112, 2018.
- MELLO, L. E., SUMAN, A., MEDEIROS, C. B., PRADO, C. A., RIZZATTI, E. G., NUNES, F. L. S., BARNABÉ, G. F., FERREIRA, J. E., SÁ, J., REIS, L. F. L., RIZZO, L. V., SARNO, L., DE LAMONICA, R., MACIEL, R. M. B., CESAR-JR, R. M., AND CARVALHO, R. Opening Brazilian COVID-19 patient data to support world research on pandemics. DOI: 10.5281/zenodo.3966427, 2020.
- MENENDEZ, E. S., CASANOVA, M. A., LEME, L. A. P., AND BOUGHANEM, M. Novel node importance measures to improve keyword search over rdf graphs. In *International Conference on Database and Expert Systems Applications*. Springer, pp. 143–158, 2019.
- OLIVEIRA, P., SILVA, A., AND MOURA, E. Ranking candidate networks of relations to improve keyword search over relational databases. In *31st International Conference on Data Engineering*. IEEE, DOI: 10.1109/ICDE.2015.7113301, pp. 399–410, 2015.
- RAMADA, M. S., SILVA, J. C., AND LEITÃO-JÚNIOR, P. S. From keywords to relational database content: A semantic mapping method. *Information Systems* vol. 88, pp. 101460, 2020.
- ROSER, M., RITCHIE, H., ORTIZ-OSPINA, E., AND HASELL, J. Coronavirus pandemic (covid-19). *Our World in Data*, 2020.
- TRAN, T., WANG, H., RUDOLPH, S., AND CIMIANO, P. Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In *25th International Conference on Data Engineering (ICDE)*. IEEE, pp. 405–416, 2009.
- VINAY, M. S. AND HARITSA, J. R. Operator implementation of result set dependent kws scoring functions. *Information Systems* vol. 88, pp. 11, 2020.
- ZHOU, Q., WANG, C., XIONG, M., WANG, H., AND YU, Y. Spark: adapting keyword query to semantic search. In *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC'07/ASWC'07)*. Springer, pp. 694–707, 2007.