Web Semantics: Science, Services and Agents on the World Wide Web 71 (2021) 100662

Contents lists available at ScienceDirect

Web Semantics: Science, Services and Agents on the World Wide Web

journal homepage: www.elsevier.com/locate/websem

# Towards the next generation of the LinkedGeoData project using virtual knowledge graphs

Linfang Ding [a], Guohui Xiao [a,b,*], Albulen Pano [a,b], Claus Stadler [c], Diego Calvanese [a,b,d]

[a] KRDB Research Centre, Free-University of Bozen-Bolzano, Piazza Domenicani 3, 39100 Bolzano, Italy
[b] Ontopic s.r.l, via A. Volta 13/A, 39100 Bolzano, Italy
[c] Institute for Applied Informatics, University of Leipzig, 04109 Leipzig, Germany
[d] Department of Computing Science, Umeå University, 901 87 Umeå, Sweden

ARTICLE INFO

*Article history:*
Received 18 January 2021
Received in revised form 7 August 2021
Accepted 16 September 2021
Available online 6 October 2021

*Keywords:*
LinkedGeoData
Virtual knowledge graph
GeoSPARQL
Ontop
OpenStreetMap

ABSTRACT

With the advancement of Semantic Technologies, large geospatial data sources have been increasingly published as Linked data on the Web. The *LinkedGeoData* project is one of the most prominent such projects to create a large knowledge graph from *OpenStreetMap* (OSM) with global coverage and interlinking of other data sources. In this paper, we report on the ongoing effort of exposing the relational database in LinkedGeoData as a SPARQL endpoint using *Virtual Knowledge Graph* (VKG) technology. Specifically, we present two realizations of VKGs, using the two systems Sparqlify and Ontop. In order to improve compliance with the OGC GeoSPARQL standard, we have implemented GeoSPARQL support in Ontop v4. Moreover, we have evaluated the VKG-powered LinkedGeoData in the test areas of Italy and Germany. Our experiments demonstrate that such system supports complex GeoSPARQL queries, which confirms that query answering in the VKG approach is efficient.



## 1. Introduction

With the advancement of Semantic Technologies and emerging standards such as GeoSPARQL, large amounts of geospatial data have been increasingly published as Linked Data on the Web. Such projects normally convert existing data sources to RDF graphs using a materialization-based approach [1]. One of the most notable efforts is the *LinkedGeoData* project,[1] which derives a large and rich spatial semantic data source from *OpenStreetMap* (OSM). The OSM project[2] is creating a free editable map of the world. Since its inception in 2004, it has grown to cover both points- and regions-of-interests comprising 6.5 billion points on the earth[3] and has become the most prominent example of a Volunteered Geographic Information (VGI) ecosystem with contributions from around seven million users.

The LinkedGeoData project complements OSM by converting it to an RDF knowledge graph and linking it with other data sources, e.g., DBPedia. LinkedGeoData began in 2009 [2] and further broadened in 2012 [3]. Nowadays, LinkedGeoData has become a valuable resource for spatial Semantic Web research. To name a few, it has been used in the research fields of string-based entity linking [4,5], spatial entity linking [6], entity alignment [7], and topological relation discovery [8]. Moreover, the SPARQL query logs over LinkedGeoData, as a separate resource, have been used for various analysis tasks [9].

In the very beginning, the LinkedGeoData project was based on an extract-transform-load (ETL) process paradigm using a set of mapping rules, as described in [3]. However, the most notable drawback of ETL over ∼7.5B OSM entities is the time it takes to reprocess all data after any change in the mapping. As an alternative approach, the *Virtual Knowledge Graph* (VKG) paradigm [10], also called *Ontology-based Data Access* (OBDA) [11] in the literature, can virtualize underlying data sources, typically relational tables, through ontology and mapping, as a knowledge graph. This virtual knowledge graph can be queried using the SPARQL language.

Indeed, shortly after the publication of the main reference of LinkedGeoData [3], the project shifted behind the scenes to a VKG approach using *Sparqlify*,[4] as first mentioned in [12]. Sparqlify makes it possible to create RDF dumps, generate the ontology, and power the Linked Data interface using SPARQL queries against the VKG. However, due to the functional and performance

* Corresponding author at: KRDB Research Centre, Free-University of Bozen-Bolzano, Piazza Domenicani 3, 39100 Bolzano, Italy.
*E-mail addresses:* ding@inf.unibz.it (L. Ding), xiao@inf.unibz.it (G. Xiao), albulen.pano@ontopic.biz (A. Pano), cstadler@informatik.uni-leipzig.de (C. Stadler), calvanese@inf.unibz.it (D. Calvanese).

[1] http://linkedgeodata.org/

[2] https://www.openstreetmap.org

[3] https://wiki.openstreetmap.org/wiki/Stats as of 23 November 2020.

[4] https://github.com/SmartDataAnalytics/Sparqlify

https://doi.org/10.1016/j.websem.2021.100662

limitations of Sparqlify, the VKG is also materialized and served via a Virtuoso Open Source[5] triple store. A full dump generated in 2013[6] amounted to ∼27B triples in 121GB of bz2 compressed data. As this is unwieldy even today, a materialization of the complete VKG is typically not feasible such that exporting only the data related to certain classes of interest is more useful.

The use of Sparqlify in the LinkedGeoData project confirmed that geospatial functions could be robustly defined and executed in seconds using the VKG approach. However, the LinkedGeoData project requires an *open source* VKG engine that is *(1)* compliant with all the relevant standards (namely OWL, R2RML, SPARQL, and GeoSPARQL), and *(2)* whose query execution approach scales to the number of involved mappings and the size of OSM data, which contains billions of spatial entities. Sparqlify partially supports the R2RML and SPARQL standards, but not OWL or GeoSPARQL, and the performance of query answering is suboptimal. Other open source VKG engines, e.g., D2RQ [13] and Morph [14], are in a similar situation. The most promising one is Ontop-spatial [15,16], which is derived from Ontop v1.18 [17] and supports a large fragment of GeoSPARQL. However, since Ontop-spatial is based on an old version of Ontop, it cannot take advantage of the features provided by the latest versions (v4.x) [18], notably better compliance with relevant standards (e.g., aggregation functions in SPARQL), improved performance, and new tooling (e.g., built-in SPARQL endpoint and Docker infrastructure).

In order to build a suitable VKG engine for LinkedGeoData, we have reimplemented GeoSPARQL support in Ontop v4.1. This has significantly improved compliance with the OGC GeoSPARQL standard. All of the geospatial functions defined in GeoSPARQL are implemented. In particular, it features improved handling of units (such as degrees and metres) and different spatial reference systems (SRIDs). We have tested it over PostgreSQL/PostGIS and H2/H2GIS, and it should work with all relational database systems that are compliant with the OGC Implementation Standard *Simple Feature Access* [19].

Using Sparqlify and Ontop, we can expose LinkedGeoData as VKGs through a suitable ontology and mappings over the OSM tables (and some derived views). This new VKG-based architecture is using the container technology Docker, which encapsulates all the required software. Now users can easily deploy an instance of LinkedGeoData and customize the area and the mappings according to their needs. We have deployed an instance (currently with only data in the country of Monaco for demonstration) of the new version of LinkedGeoData online[7] using both Ontop and Sparqlify.

We have conducted an evaluation of LinkedGeoData VKGs with both Sparqlify and Ontop using the three areas of North-East Italy, Italy, and Germany. The experiments demonstrate that these systems support complex GeoSPARQL queries and confirm that the VKG approach is efficient in query answering. Finally, it shows a clear advantage of Ontop over Sparqlify in both supported GeoSPARQL query features and performance of query answering.

The remainder of this work is organized as follows: Section 2 reports the limitations and recent developments of the LinkedGeoData project. Section 3 discusses the implementation of Sparqlify and Ontop-spatial for basic geospatial SPARQL functions. Section 4 presents the new implementation of GeoSPARQL functions in Ontop v4. Section 5 details the setup of LinkedGeoData VKGs. Section 6 presents evaluations over LinkedGeoData. Finally, Section 7 discusses future work and concludes the paper.

5 http://vos.openlinksw.com/

6 http://downloads.linkedgeodata.org/full-dumps/

7 https://linkedgeodata.org/monaco/

## 2. The LinkedGeoData project

In this section, we present the revised architecture of the LinkedGeoData (LGD) project based on VKG technology and the improvements over the prior ETL-based one.

LinkedGeoData is an effort to add a spatial dimension to the Web of Data. LinkedGeoData uses the information collected by the OpenStreetMap project and makes it available as an RDF knowledge base according to the Linked Data principles. As such, standard compliance w.r.t. data format and access (RDF and SPARQL) as well as w.r.t. vocabularies (e.g., spatial vocabularies) are an essential part of the project. All components of the project are available as Open Source software and there now exists a dockerized setup that enables custom deployments of LinkedGeoData's services with the desired subsets of OSM data.

The core of the project is the "RDFization" of OSM data. In the very beginning, this was based on an ETL process. However, the most notable drawback of ETL over roughly 7.5B OSM entities (of which each is further described using a possibly empty set of tags) is the time it takes to reprocess all data after any change in the mapping. Regarding ETL, although Big Data technology brought significant advancements by distributing workloads over clusters together with massive parallelization of operations over large sets of records, it is still rather heavy and not very flexible in handling the changes of data and mapping rules.

VKG technology enables modified mappings to take effect instantly w.r.t. SPARQL query execution. This is particularly useful for checking whether a change in the mapping has the desired effect and for running "unit tests" over the data with frameworks such as RDFUnit [20] in order to, e.g., detect regressions.

The ETL-based version of LinkedGeoData required a custom (Java) software framework to ingest OSM data, process it into RDF, and keep a triple store in sync. With the shift to VKG technology, LinkedGeoData's core is much more lightweight as it is mainly formed only by a set of SQL scripts and RDB2RDF mappings.[8] LinkedGeoData's SQL scripts extend the OSM schema with RDF mapping tables, views and additional database indices without modifying OSM's schema on which other tool chains depend upon. A small set of utility shell scripts, especially the `lgd-createdb` command, take care of the setup.

The components of LinkedGeoData's current architecture are depicted in Fig. 1 and explained as follows:

- **Data Sources.** The downloads and changesets of OSM are the enablers for subsequent data processing and replication. *GeoFabrik*[9] is a well-known service which publishes pre-partitioned OSM dumps and changesets organized into hierarchical regions. For example, one path is Europe » Italy » Nord-Est.
- **Replication.** *Osmosis* is a command line Java application for processing OSM data.[10] It features readers for OSM data from different sources, filters, and converters, as well as writers for different data sinks such as files and databases. It also supports all common OSM serialization formats (xml, pbf, csv). The most relevant feature for LinkedGeoData is its capability to import and replicate a PostgreSQL/PostGIS database system with OSM data.
- **Physical Storage.** The PostGIS database for LinkedGeoData is extended with mapping tables, views, and functional indexes (a limited form of incremental materialized views), as detailed in Section 5.

8 We use "RDB2RDF mapping" as a generic term for the W3C standard R2RML (RDB to RDF Mapping Language), and other dialects supported by systems like Ontop and Sparqlify.

9 https://www.geofabrik.de/

10 https://wiki.openstreetmap.org/wiki/Osmosis

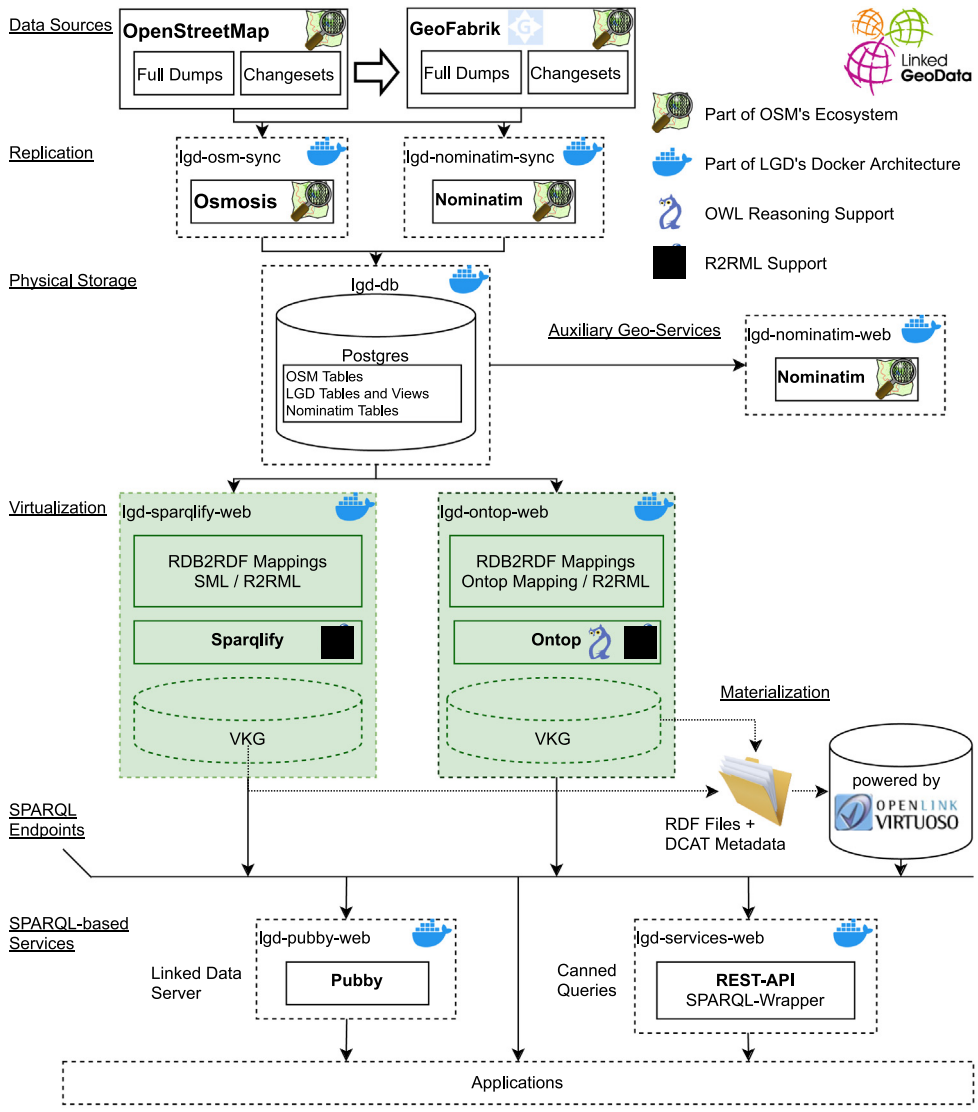


**Fig. 1.** LinkedGeoData's architecture. Boxes with dashed lines indicate dockerized components. Ontop is the novel addition to the project.

- **SPARQL Services via Virtualization.** Given a set of RDB2RDF mappings, both Ontop and Sparqlify can expose the same database as similar VKGs via a SPARQL endpoint. We kept both systems because they originate from different capabilities of the engines, and because hosting a VKG engine does not require much resource thanks to the virtual nature of such systems. Addition of future VKG engines with support for PostGIS and R2RML to LinkedGeoData's architecture only requires to contribute an appropriately configured docker file.
- **Auxiliary Geo-Services.** *Nominatim*[11] is a search tool over OSM data that supports geocoding and reverse-geocoding. Its introduction to LinkedGeoData solves a long standing issue: Although the Osmosis tool replicates "raw" OSM data, it does not compute effective geometries. For example, large line-strings, such as boundaries of countries, are split into multiple segments, which are subsequently related to each other using an OSM relation. Interpretation of whether an OSM relation that forms a closed sequence of ways represents a line-string or a polygon depends on the tags. Nominatim supports computation of the effective geometries for many relevant spatial features and also uses PostGIS as a backend. Effective geometries are stored together with the OSM entity identifier, which allows for joins with the tables of OSM. Additional RDB2RDF mappings are employed to expose these effective polygons. Nominatim also ships with its own replication tool.
- **Materialization.** Downloadable RDF datasets are produced from the VKG by means of materializing portions of it using SPARQL `CONSTRUCT` queries. In accordance with the *FAIR principles* (findability, accessibility, interoperability, and reuse) [21], the `lgd-dumpdb` tool not only exports data partitions using SPARQL `CONSTRUCT` queries, but it also generates a DCAT model that describes the exported RDF files. On this basis, the data publishing process of a collection of data partitions (such as by classes, e.g., Amenity and Peak) can be automated. OpenLink's *Virtuoso Open Source* (VOS) RDF store is a high performance native triple store that is traditionally used to serve the materialized datasets. Virtuoso's commercial version won the MOCHA2018 Triple Store Challenge [22] with VOS also achieving high scores. VOS 7.2.6 promises improved GeoSPARQL support, however

11 https://wiki.openstreetmap.org/wiki/Nominatim

at present only a development release[12] was published that has not yet been evaluated for LinkedGeoData. Note, that the focus of this work is about realizing and evaluating LinkedGeoData's virtual knowledge graph with support for GeoSPARQL queries, rather than evaluating native RDF store performance.

- **SPARQL-based Services.** These services can wrap any of the SPARQL endpoints, especially the VKG-based ones, in order to deliver data access interfaces that some may consider easier to use than SPARQL. Pubby[13] is a very old yet functional wrapper for publishing data according to the Linked Data principles.[14] The LinkedGeoData REST API is essentially a set of REST methods backed by canned SPARQL queries. For example, the `/ontology` method only executes a SPARQL `CONSTRUCT` query that retrieves all triples of resources typed with `owl:Class` or `rdf:Property`.

The LinkedGeoData docker architecture is aimed at making the setup of all involved services as simple as possible. A "`.env`" file contains all relevant options for performing initial data loading and subsequent replication, and for running the services. The main options are shown in Listing 1 and comprise the amount of memory allowed for the database and two URLs that point to the initial OSM dataset and the subsequent updates.

**Listing 1:** Excerpt of config options of a LinkedGeoData docker stack

```
DB_SHARED_BUFFERS=2GB
DB_WORK_MEM=256MB
OSM_DATA_BASE_URL=http://download.geofabrik.de/<dataset>.osm.pbf
OSM_DATA_SYNC_URL=http://download.geofabrik.de/<dataset>-updates/
```

The OSM schema is non-typical for a relational database, and for this reason LinkedGeoData also serves as a test-bed for evaluating corner cases of VKG engines. The conventional baseline approach for mapping relational data to RDF is based on class-per-table and predicate-per-column mappings. The R2RML W3C recommendation includes a specification of how to perform this approach in a standard way under the name *Direct Mapping*.[15] However, in an OSM database *nearly all* data is represented as tags that are stored in a generic schema with key–value columns. As OSM contains billions of spatial entities, a VKG engine's query execution approach needs to scale to both the number of involved RDB2RDF mappings and the size of the data. Furthermore, GeoSPARQL has become the specification for spatial data access in the Semantic Web. LinkedGeoData's VKG approach to exposing OSM data as RDF provides a test-bed for state-of-the-art VKG engines to demonstrate their advances in GeoSPARQL compliance. Conversely, VKG engines supporting this standard are a highly relevant and welcome contribution to the LinkedGeoData project.

## 3. Basic spatial SPARQL support in VKGs

In this section, we first present the SPARQL-SQL rewriter Sparqlify used in LinkedGeoData, and discuss its limitation. Then we present the basic spatial SPARQL support in Ontop-spatial v1.

### 3.1. Sparqlify

*Sparqlify*[16] is a SPARQL-SQL rewriter that enables the definition of RDF views on relational databases and querying them with SPARQL. The LinkedGeoData project has used Sparqlify to provide access to virtual triples from the OpenStreetMap database. Sparqlify supports two mapping languages, i.e., R2RML and its own *Sparqlification Mapping Language* (SML), which was developed as a more human friendly alternative to R2RML [23].

```
Create View people As
  Construct
    { ?s foaf:firstName ?fn }
  With
    ?s = uri(eg:, ?id)
    ?fn = plainLiteral(?"first name")
  From personTable
```

**Fig. 2.** Example SML mapping.

SML is inspired by SQL's `CREATE VIEW` statement and SPARQL's `CONSTRUCT` query form. Both SPARQL and SQL result sets are relational in nature,[17] however SPARQL mandates "column types" to be uniformly RDF terms, whereas in SQL different types cannot be mixed. The construction of RDF terms from column values can be achieved by externally providing suitable meta data to perform a mapping. An example of an SML specification is shown in Fig. 2. The `With` clause is a set of *term constructor expressions*, where the variable on the left hand side of '=' is the one being defined, whereas the ones on the right hand side refer to column names. Special characters in column names can be handled with the use of double quotes.

The recent work [24] reports its findings on using Sparqlify to rewrite SPARQL queries to the SQL dialect of the Apache Spark[18] framework.

*Geospatial functions.* In LinkedGeoData, the Virtuoso DBMS in its Open Source edition (VOS) is traditionally used as the primary triple store. While recent enterprise versions of Virtuoso feature support for GeoSPARQL, the Open Source version features only a limited set of spatial functions that predate GeoSPARQL. Sparqlify's geospatial support is aimed at interoperability with VOS, which comprises the following functions:

- `bif:st_intersects`. This function has two different implementations depending on the respective PostGIS SQL function used: *(1)* `ST_INTERSECT`, which checks whether the geometries intersect; *(2)* `ST_DWITHIN`, which checks whether the geometries are within a specified distance in kilometres (km) from one-another.
- `bif:st_point`. This function is derived from the PostGIS function `ST_POINT`, and it can retrieve geometry `ST_Point(float x, float y)` with the respective longitude and latitude. An additional 3rd argument can be provided as an SRID `ST_SetSRID(ST_Point(float x, float y), int srid)` to set this point into a specific spatial reference system.
- `bif:st_geomFromPoint`. This function is equivalent to `bif:st_point`.
- `bif:st_geomFromText`. This function is derived from the PostGIS function `ST_GeomFromText`. Similar to `ST_POINT`, through `ST_SetSRID` a composite function is created through which an SRID can be set.

12 https://sourceforge.net/projects/virtuoso/files/virtuoso/
13 https://github.com/cygri/pubby
14 https://www.w3.org/DesignIssues/LinkedData
15 https://www.w3.org/TR/rdb-direct-mapping/
16 http://aksw.org/Projects/Sparqlify.html
17 Even though SPARQL result sets are formally defined a sets of partial functions from variables to RDF terms, a relation is obtained by treating each variable mentioned in the domains as a separate column.
18 https://spark.apache.org/

The distance and intersection functions provide useful functionalities and can be written so as to replicate other functions, such as `geof:sfWithin`. However, the lack of OGC GeoSPARQL compliance and even deviation in the case when `bif:st_intersects` acts as `ST_DWITHIN`, limits the scenarios for which LinkedGeoData can be compared with other VKG solutions.

Sparqlify uses an XML format to declare the mappings of SPARQL extension function IRIs to SQL expressions, as shown below:

```
<simpleFunctions>
 <simpleFunction>
  <name>http://www.openlinksw.com/schemas/bif#st_geomFromPoint</name>
  <mappings>
   <mapping>
    <signature>geometry ST_Point(float, float)</signature>
    <pattern>ST_SetSRID($name$($1$, $2$), 4326)</pattern>
   </mapping>
   <mapping>
     <signature>geometry ST_Point(float, float, int)</signature>
     <pattern>ST_SetSRID($name$($1$, $2$), $3$)</pattern>
   </mapping>
  </mappings>
 </simpleFunction>
</simpleFunctions>
```

The `signature` element declares a specific SQL function name together with the argument and return types. Note that as a design choice Java names such as `float` or `double` were used for the declarations, however these names are not in the standard SQL (even though some RDBMSs support them). Internally, these names are then mapped to the appropriate RDF datatype IRIs and SQL datatype names. This allows for converting the RDF terms provided as function arguments to the appropriate SQL types, and for converting back the SQL function's return value to an RDF term. In accordance with the SPARQL 1.0 and 1.1 specifications, incompatible types result in a type error.

*Shortcomings.* Although Sparqlify features a configurable and extensible framework and has proven that VKG technology can be applied to the OSM database, it does not support GeoSPARQL, aggregate functions besides `COUNT(*)`, or ontological reasoning. Also, query answering performance can be suboptimal. The main reason is that when translating a SPARQL query, Sparqlify simply translates each triple pattern in the input SPARQL query to the corresponding SQL queries from mappings as is, but does not perform further optimizations. This often leads to SQL queries with many redundant subqueries, which are expensive to evaluate by current DB engines. As a consequence, the LinkedGeoData project can be significantly improved by introducing a VKG engine that mitigates these issues.

### 3.2. Ontop-spatial v1

Ontop [17,18], initiated at Free University of Bozen-Bolzano, is a state-of-the-art VKG system. It supports almost all the features of the relevant W3C standards (R2RML, OWL 2 QL, SPARQL 1.1), and all major relational databases. It also has its own mapping language. For example, the example SML mapping in Fig. 2 can be equivalently written in Ontop as:

```
mappingId people
target    eg:{id} foaf:firstName {first name} .
source    SELECT * FROM personTable
```

A spatial extension of Ontop, which we call Ontop-spatial v1,[19] has been developed by National and Kapodistrian University of Athens as a fork of Ontop v1.18 for supporting geospatial data [16]. Ontop-spatial v1 implemented a large fragment of the GeoSPARQL standard and has been successfully deployed in a number of use cases in maritime security [25], oil exploration [26], data quality assessment [27], and visual analytics [28]. It has also been used as a main technology in the H2020 projects Copernicus App Lab [29] and DeepCube.[20]

However, since Ontop-spatial v1 is based on Ontop v1.18, it cannot catch up with the latest Ontop version (v4.x) [18], which comes with better compliance with relevant standards (e.g., aggregation functions in SPARQL), improved performance, and new tooling (e.g., built-in SPARQL endpoint and Docker infrastructure). Also, the simpler internal representation used in Ontop v1 (and also in Ontop-spatial v1) does not allow one to properly take into account all the aspects of GeoSPARQL with respect to projection systems, units, and type inference. An adoption of Ontop-spatial v1 in the LinkedGeoData project would suffer from the same limitations.

## 4. Improving GeoSPARQL support in VKGs

In this section, we describe how we improve GeoSPARQL support in VKGs, which is crucial for the LinkedGeoData project. Specifically, we first recall in Section 4.1 some GeoSPARQL features and their challenges, and then explain in Section 4.2 how we implement them in Ontop v4. We observe that in the following we refer to the following three namespace prefixes: `geo`,[21] `geof`,[22] and `uom`.[23]

### 4.1. OGC GeoSPARQL and current implementations in VKGs

The OGC GeoSPARQL standard defines a vocabulary for representing geospatial data in RDF, and an extension to the SPARQL query language for processing geospatial data [30]. This standard defines a rich set of geospatial functions, which can be split into two categories:

- Topological functions, which take two geometries as inputs and return a boolean value with respect to a certain topological relation, e.g., `geof:sfIntersects`. There are three families of topological relations: *Simple Features*, *Egenhofer*, and *Region Connection Calculus (RCC8)*. Simple Features and Egenhofer apply to all geometries, including points, lines, and polygons. In contrast, RCC8 functions only apply to 2-dimensional geometries such as polygons, but not to lines or points. The more general ternary `geof:relate` function takes one more input for a *pattern-matrix*, which represents a *Dimensionally Extended 9-Intersection Model* (DE-9IM) intersection pattern consisting of T (true) and F (false) values, and returns true if the spatial relationship between the two geometries corresponds to the pattern-matrix.
- Non-topological functions, which take a geometry and possibly some other parameters as inputs and compute some values (e.g., `geof:distance`) or geometries (e.g., `geof:buffer`).

A complete list of GeoSPARQL functions is provided in Table 1.

Below, we describe some important aspects of GeoSPARQL, including *measurement units*, *SRIDs*, and *Geometry Literal Serialization*. To the best of our knowledge, no existing open source VKG system is able to support all of them.

19 http://ontop-spatial.di.uoa.gr/

20 https://deepcube-h2020.eu/

21 http://www.opengis.net/ont/geosparql#

22 http://www.opengis.net/def/function/geosparql/

23 http://www.opengis.net/def/uom/OGC/1.0/

**Table 1**
GeoSPARQL functions.

| Topological functions. (∗) indicates no direct SQL counterpart | | | Non-topological function |
|---|---|---|---|
| *Simple features* | *Egenhofer* | *RCC8* | `geof:distance` |
| `geof:sfEquals` | `geof:ehEquals` | `geof:rcc8eq` | `geof:buffer` |
| `geof:sfDisjoint` | `geof:ehDisjoint` | `geof:rcc8dc` (∗) | `geof:convexHull` |
| `geof:sfIntersects` | `geof:ehMeet` | `geof:rcc8ec` (∗) | `geof:intersection` |
| `geof:sfTouches` | `geof:ehOverlap` (∗) | `geof:rcc8po` (∗) | `geof:union` |
| `geof:sfCrosses` | `geof:ehCovers` | `geof:rcc8tppi` (∗) | `geof:difference` |
| `geof:sfWithin` | `geof:ehCoveredBy` (∗) | `geof:rcc8tpp` (∗) | `geof:symDifference` |
| `geof:sfContains` | `geof:ehInside` (∗) | `geof:rcc8ntpp` (∗) | `geof:envelope` |
| `geof:sfOverlaps` | `geof:ehContains` (∗) | `geof:rcc8ntppi` (∗) | `geof:boundary` |
| | | | `geof:getSRID` |
| `geof:relate` | | | |

*Measurement units.* Measurement units are of critical importance to functions that deal with distance. Specifically, for the functions `geof:distance` and `geof:buffer`, the last parameter is the measurement unit. Currently, three widely used metrics are: `uom:metre`, `uom:degree`, and `uom:radian`. Standard distance and buffer functions implemented on popular geospatial RDBMSs, such as PostGIS and H2GIS, default to degrees. Therefore, special attention on units needs to be taken when translating GeoSPARQL functions to SQL functions in VKG systems.

We recall that Ontop-spatial v1 and Sparqlify ignore measurement units, which in practice means that they only support `uom:degree`.

*Spatial reference system identifier and geometry literal serialization.* The *spatial reference system identifier* (SRID) is the identifier for a geographic coordinate system. In GeoSPARQL, an SRID is specified as an IRI, e.g.,

- `<http://www.opengis.net/def/crs/OGC/1.3/CRS84>` denotes the WGS 84 geodetic *longitude–latitude* spatial reference system. This is the default SRID used in GeoSPARQL if not specified explicitly.
- `<http://www.opengis.net/def/crs/EPSG/0/4326>` denotes the WGS 84 geodetic *latitude–longitude* spatial reference system. Note that this spatial reference system defines a different axis order from the former SRID.
- `<http://www.opengis.net/def/crs/EPSG/0/3044>` is a Cartesian 2D coordinate system used in Europe between 6°E and 12°E, with the metre as the unit.

The SRIDs are used for serializing geometry as literals. GeoSPARQL adopts two serialization types of geometries: *Well-Known Text* (WKT) and *Geographic Markup Language* (GML) as two RDF datatypes: `geo:wktLiteral` and `geo:gmlLiteral`. For example, the literal below encodes a point geometry in WKT using the default WGS 84 SRID:

```
"POINT(-83.38 33.95)"^^geo:wktLiteral
```

and the one below encodes the same point using EPSG 4326 (note the order of axes):

```
"<http://www.opengis.net/def/crs/EPSG/0/4326>
POINT(33.95 -83.38)"^^geo:wktLiteral
```

We recall that Ontop-spatial v1 and Sparqlify do not support specifying SRIDs in geometry literals.

### 4.2. Implementation of Ontop-spatial v4

Ontop has been undergoing significant refactoring to be better compliant with relevant standards [18]. The new internal data structures in Ontop v4 provide a solid foundation to reimplement GeoSPARQL support systematically and to address the challenges mentioned above. To distinguish it from Ontop-spatial v1, in the following, we call this new implementation Ontop-spatial v4. Below we discuss in detail how to implement GeoSPARQL functions.

#### 4.2.1. GeoSPARQL functions

Recall that, to support GeoSPARQL in VKGs, the SQL counterpart we primarily rely on is the OGC standard *Simple Feature Access Standard* [19], which in turn depends on the geospatial SQL functionalities that are standardized in ISO/IEC 13249 SQL/MM [31]. It defines geometries as the main data type on which spatial operations are applied. The SQL/MM standard uses the prefix `ST_`, which stands for *spatial* and *temporal*, for all tables, views, types, methods, and function names.

Many GeoSPARQL functions, e.g., all the Simple Feature functions, have direct SQL correspondences. For example, the GeoSPARQL function `geof:sfDisjoint` can be translated to the standard SQL function `ST_DISJOINT`. Such functions have already been implemented in Ontop-spatial v1. Some GeoSPARQL functions do not have direct SQL counterparts (marked with (∗) in Table 1). These include half of the Egenhofer functions and most of the RCC8 functions, and they are not fully supported in Ontop-spatial v1. To support them, we rely on the *DE-9IM pattern-matrices* of these functions (defined in Tables 1–3 in [30]). For example, the matrix of `geo:ehInside` is `TFF*FFT**`. These GeoSPARQL functions can be translated to the generic SQL function `ST_RELATE`,[24] which allows for evaluating the topological relationships between two geometries according to a DE-9IM pattern-matrix. E.g., `geo:ehInside(?x,?y)` can be translated to `ST_RELATE (x,y,'TFF*FFT**')`. For more information about these three families and pattern-matrices, we refer to the GeoSPARQL specification [30] and the relevant references within.

#### 4.2.2. Handling SRIDs and units

SRIDs and units play an important role in the functions `geof:getSRID`, `geof:distance`, and `geof:buffer`. Below we detail the implementations of `geof:getSRID` and `geof:distance`, but ignore `geof:buffer` since it works similarly to `geof:distance`.

*Implementation of `geof:getSRID`.* The function `geof:getSRID`($x$) returns the SRID of the input literal $x$. The challenge is that the SRID can be part of $x$ as shown in the example in Section 4.1. In this case, it requires analysing the string value of the literal, which can be expensive. To guarantee good performance, we limit our implementation to be *database instance independent*, i.e., we compute the SRID during the query translation process, and do not need to delegate it to query evaluation.

For example, we support the following templates for specifying the WKT of a point in the mapping target:

- `"{geom}"^^geo:wktLiteral`
- `"<http://www.opengis.net/def/crs/EPSG/0/3044> {geom}"^^geo:wktLiteral`
- `"<http://www.opengis.net/def/crs/EPSG/0/3044> POINT({x} {y})"^^geo:wktLiteral`

[24] https://postgis.net/docs/ST_Relate.html

When evaluating `geof:getSRID` over these points, we obtain the default WGS 84 for the first one, and `<http://www.opengis.net/def/crs/EPSG/0/3044>` for the last two. However, in the first case, if the column `geom` stores *string values* with an SRID inside, like

```
"<http://www.opengis.net/def/crs/EPSG/0/3044>
 POINT(668682.853 5122639.964)",
```

our implementation is not able to extract the SRID in such columns. Nevertheless, such values do not make sense in practical GIS systems, because the string is not a valid WKT for the database, and spatial indexes cannot be applied.

Note that this complication is a consequence of the "feature" of mixing SRID and WKT in the serialization. There is an ongoing discussion in the community to decouple them in GeoSPARQL 2.0.[25] Such a change would simplify our implementation.

*Implementation of `geof:distance`.* One of the most challenging functions to implement is `geof:distance`, because both measurement units (i.e., metres, degrees, or radians) and SRIDs need to be considered when defining correct translations. To illustrate the translation, we first present two examples and then introduce the general algorithm.

**Example 1** (*Distance on a Cartesian 2D Coordinate*)**.** Consider the following table `cities` with two columns `name` and `geom`, where the geometries are in the EPSG 3044[26] projection:

| Name | Geom |
|---|---|
| Bolzano | POINT(680690.38 5152087.65) |
| Merano | POINT(665178.23 5170708.71) |

We can use the following mapping (in the Ontop mapping syntax) to construct a VKG:

```
target  :{name} geo:asWKT
  "<http://www.opengis.net/def/crs/EPSG/0/3044>
   {geom}"^^geo:wktLiteral .
source   SELECT name, geom FROM cities
```

From the template of the WKT literal, we know that the SRID is EPSG 3044. By consulting a library like proj4j,[27] we know that the input unit is metre.

Now, consider the following SPARQL query to compute the distance between Bolzano and Merano in metres:

```
SELECT ?dist WHERE {
  :bolzano geo:asWKT ?wkt1 .
  :merano  geo:asWKT ?wkt2 .
  BIND(geof:distance(?wkt1, ?wkt2, uom:metre)
       AS ?dist)
}
```

We can directly translate it to the following SQL query using the `ST_DISTANCE` function:

```
SELECT ST_DISTANCE(t1.geom, t2.geom) AS dist
FROM cities t1, cities t2
WHERE t1.name = 'bolzano' AND t2.name = 'merano'
```

If we change the unit to radian in the query, i.e., using the following BIND clause:

```
BIND(geof:distance(?wkt1, ?wkt2, uom:radian)
     AS ?dist)
```

we can convert the length to metre by dividing by the radius of the earth (6,370,986 m), and obtain the following SQL query:

```
SELECT ST_DISTANCE(t1.geom, t2.geom)/6370986
         AS dist
FROM cities t1, cities t2
WHERE t1.name = 'bolzano' AND
      t2.name = 'merano'    ◁
```

**Example 2** (*Distance Between Geometries on a Sphere*)**.** Consider the table with the same structure as above but the geometries are in the WGS 84 longitude–latitude projection:

| Name | Geom |
|---|---|
| Paris | POINT(2.3522 48.8566) |
| London | POINT(-0.1278 51.5074) |

and the mapping

```
target :{name} geo:asWKT
  "{geom}"^^geo:wktLiteral.
source SELECT name, geom FROM cities
```

In this case, the WKT literal uses the default SRID `<http://www.opengis.net/def/crs/OGC/1.3/CRS84>` and the unit is degree.

To compute the distance between Paris and London, we can use the following GeoSPARQL query:

```
SELECT ?dist WHERE {
  :paris  geo:asWKT ?wkt1 .
  :london geo:asWKT ?wkt2 .
  BIND(geof:distance(?wkt1, ?wkt2, uom:metre)
       AS ?dist)
}
```

This query can be translated to the following SQL query using the `ST_DISTANCESPHERE` function:

```
SELECT ST_DISTANCESPHERE(t1.geom, t2.geom)
         AS dist FROM cities t1, cities t2
WHERE t1.name = 'paris' AND t2.name = 'london'    ◁
```

To summarize, for all the different cases we can use the SQL functions `ST_DISTANCE` and `ST_DISTANCESPHERE` in conjunction with some additional arithmetic functions. The more general translation, which supports metre and degree as input units, and also radian as output unit, is provided below:

```
function translate_distance(term1, term2,
                            outputUnit):
  let R_e = radius of the earth in metres
  srid1, geom1 = extractSRIDandGeom(term1)
  srid2, geom2 = extractSRIDandGeom(term2)
  if (srid1 != srid2)
    exit "unsupported: SRIDs do not match"
  // using the proj4j library
  inputUnit = getUnit(srid1)
  if inputUnit == METRE:
    d_m = ST_DISTANCE(geom1, geom2)
  else if inputUnit == DEGREE:
    d_m = ST_DISTANCESPHERE(geom1, geom2)

  if outputUnit == METRE:
    return d_m
  else if outputUnit == RADIAN:
    return  d_m · 180/π
  else if outputUnit == DEGREE:
    return d_m/R_e · 180/π
```

25 https://github.com/opengeospatial/ogc-geosparql/issues/31
26 https://epsg.io/3044
27 https://github.com/locationtech/proj4j

```
<node id="6865960209" version="4"
  timestamp="2021-03-07T19:22:05Z"
  lat="43.7344681" lon="7.4178776">
    <tag k="addr:country" v="MC"/>
    <tag k="amenity" v="university"/>
    <tag k="email" v="admissions@monaco.edu"/>
    <tag k="name"
         v="International University of Monaco"/>
    <tag k="name:it"
         v="Università Internazionale di Monaco"/>
    <tag k="phone" v="+377 97 986 996"/>
    <tag k="website" v="https://www.monaco.edu/"/>
    <tag k="wikidata" v="Q2504327"/>
    <tag k="wikipedia"
         v="fr:Université internationale de Monaco"/>
</node>
```

**Fig. 3.** Example OSM XML data item.

#### 4.2.3. Summary

To summarize, Ontop-spatial v4 is a significant improvement upon v1. Among the new features and functionalities, we highlight the following ones: *(i)* support for SRIDs beyond CRS 84, *(ii)* support for units of metre and radian, in addition to degree, *(iii)* support for all topological functions, and *(iv)* support for the `geof:relate` function. The current version of Ontop-spatial v4 has been tested extensively on PostgreSQL/PostGIS and H2GIS. Other DB systems should also work almost out of the box if they are compliant with the standards.

We note that Ontop-spatial v4 has still not ported all the features from Ontop-spatial v1. Among these, we mention in particular the *query rewrite extent* of GeoSPARQL, and a raster data extension implemented in v1 [16,32].

## 5. Exposing LinkedGeoData as a VKG

In this section, we describe how to expose LinkedGeoData as a VKG. We show the two realizations using Sparqlify and Ontop. Specifically, we first introduce in Section 5.1 the database tables and views derived from OpenStreetMap for LinkedGeoData, and then the ontology in Section 5.2, and the mapping for VKGs in Section 5.3. Finally, we illustrate in Section 5.4 the SPARQL endpoint with an example query.

### 5.1. Database schema

Recall that OSM data consists of three fundamental geographic entities:

- *Nodes* are the most primitive entities and represent geographic points.
- *Ways* are entities that have a list of at least two node references associated with them.
- *Relations* relate points, ways, and potentially other relations to each other, thereby forming complex objects.

Each of these entities has a numeric identifier *id*, the geographic location represented using *lat* and *lon*, and a set of generic attributes described using a set of key–value pairs, known as *tags*.[28] An example data item of the node type in OSM XML is shown in Fig. 3.

**Table 2**
Example tables and views.

| node_id | k | v |
|---|---|---|
| 6865960209 | addr:country | MC |
| 6865960209 | amenity | university |
| 6865960209 | email | admissions@monaco.edu |
| 6865960209 | name | International University of Monaco |
| 6865960209 | name:it | Università Internazionale di Monaco |
| 6865960209 | phone | +377 97 986 996 |
| 6865960209 | website | https://www.monaco.edu/ |
| 6865960209 | wikidata | Q2504327 |
| 6865960209 | wikipedia | fr:Université internationale de Monaco |

(a) `node_tags`

| id | version | user_id | tstamp | geom |
|---|---|---|---|---|
| 6865960209 | 4 | 0 | 2021-03-07 19:22:05 | 01...DE4540 |

(b) `nodes`

| node_id | property | object |
|---|---|---|
| 6865960209 | rdf:type | lgdo:Amenity |

(c) `lgd_node_tags_resource_k`

| node_id | property | object |
|---|---|---|
| 6865960209 | rdf:type | lgdo:University |

(d) `lgd_node_tags_resource_kv`

| node_id | property | v |
|---|---|---|
| 6865960209 | foaf:homepage | https://www.monaco.edu/ |

(e) `lgd_node_tags_url`

| node_id | property | v | language |
|---|---|---|---|
| 6865960209 | rdfs:label | International University of Monaco | |
| 6865960209 | rdfs:label | Università Internazionale di Monaco | it |
| 6865960209 | foaf:phone | +377 97 986 996 | |

(f) `lgd_node_tags_text`

| k | object |
|---|---|
| highway | lgdo:HighwayThing |
| amenity | lgdo:Amenity |
| tourism | lgdo:TourismThing |
| historic | lgdo:HistoricThing |
| landuse | lgdo:Landuse |

(g) `lgd_map_resource_k`

| k | v | object |
|---|---|---|
| building | university | lgdo:BuildingUniversity |
| amenity | university | lgdo:University |
| amenity | airplane | lgdo:Airplane |
| amenity | drinking_water | lgdo:DrinkingWater |
| amenity | school | lgdo:School |

(h) `lgd_map_resource_kv`

We first import the OSM files using osm2pgsql[29] into the tables `node_tags`, `way_tags`, and `relation_tags` in a PostGIS database. However, these tables are not suitable for creating mappings because their structure reflecting directly the key–value pairs is too generic, and it even stores values of all different types, e.g., integer and string, into the same columns. Hence we create additional tables and views for LinkedGeoData, whose names start with ‘‘`lgd_`’’. For example, the views `lgd_node_tags_resource_k` and `lgd_node_tags_resource_kv` store the top-level and second-level classes derived from the `node_tags` table, respectively. In a similar manner, data properties are respectively loaded into views depending on their datatypes. These operations apply to ways and relations as well. In Table 2, we provide some tables and views with sample data, which we will also use in the next two subsections while explaining the ontology and mappings.

[28] A list of all the tags can be found in https://wiki.openstreetmap.org/wiki/Map_Features.

[29] https://osm2pgsql.org/

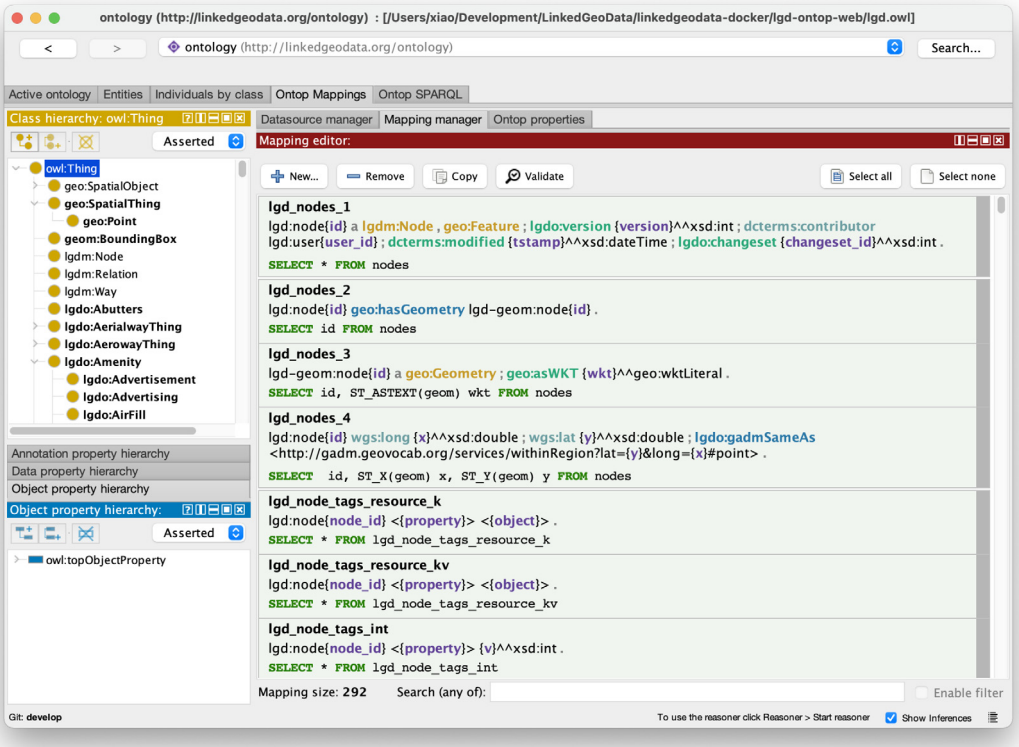


**Fig. 4.** Screenshot of ontology and mapping management in Protégé with the Ontop plugin.

**Table 3**
Statistics of the OSM datasets and DB size used for the evaluation.

| Area | #nodes | #ways | #relations | DB size |
|---|---|---|---|---|
| North-East Italy | 59.6M | 6.7M | 0.1M | 28 GB |
| Italy | 193.3M | 21.1M | 0.3M | 85 GB |
| Germany | 319.4M | 52M | 0.6M | 179 GB |

### 5.2. Ontology

The ontology is derived from the OSM tags as described in [3]. In total, the ontology includes around 1,200 classes, 250 data properties, and 80 object properties.

For Sparqlify, the ontology information is stored in the following 6 tables:

- `lgd_map_resource_k` stores top-level classes;
- `lgd_map_resource_kv` stores second-level classes;
- `lgd_map_property` stores data properties;
- `lgd_map_literal` stores data properties that also include language tags (e.g., Italian, German);
- `lgd_map_datatype` stores the data properties and their associated types (e.g., boolean, integer, float);
- `lgd_map_resource_prefix` stores some additional object properties, in particular the wikipedia language website corresponding to a class.

Sparqlify dynamically populates the ontology using the mappings. For example, the SML mapping below (slightly simplified for readability) populates the `rdfs:subClassOf` object property from the tables `lgd_map_resource_k` and `lgd_map_resource_kv`. Note that the SML mapping includes a "`Constrain`" section, which can declare, e.g., additional prefix constraints. Such information can be used as a hint for query optimization.

```
Create View sub_classes As
  Construct {
    ?child rdfs:subClassOf ?parent .
  }
  With
    ?child  = uri(?child)
    ?parent = uri(?parent)
  Constrain
    ?child  prefix "http://linkedgeodata.org/ontology/"
    ?parent prefix "http://linkedgeodata.org/ontology/"
  From
    [[SELECT a.object AS parent, b.object AS child
      FROM   lgd_map_resource_k a JOIN
             lgd_map_resource_kv b ON (b.k = a.k)
]];
```

As Ontop does not support dynamically populating the ontology from the mapping, for Ontop, we first extract the ontology using the mappings and store it as a standard OWL file. To be compliant with GeoSPARQL, we also import the GeoSPARQL ontology.[30] A screenshot of the ontology in the Protégé ontology editor[31] is provided in Fig. 4.

### 5.3. Mapping

Next we show how to develop mappings for Sparqlify and Ontop to populate the instances of the RDF graphs in LinkedGeoData. A screenshot of the Ontop mappings is provided also in Fig. 4.

The following tables and views are used for nodes. Those for ways and relations are similar:

- `nodes` stores the OSM id and geometry of the nodes;

30 http://www.opengis.net/ont/geosparql
31 https://protege.stanford.edu/

- `lgd_node_tags_resource_k` stores object properties of the nodes (including the property `rdf:type`) and top-level classes;
- `lgd_node_tags_resource_kv` stores object properties of the nodes (including the property `rdf:type`) and second-level classes;
- `lgd_node_tags_string`, `lgd_node_tags_int`, `lgd_node_tags_float`, `lgd_node_tags_boolean`, and `lgd_node_tags_url` store data properties of the nodes of different types (string, integer, float, boolean, and url, respectively);
- `lgd_node_tags_text` stores data properties of the nodes of type text as well as the language tag (e.g., en, fr, it);
- `lgd_node_interlinks` stores the DBpedia and GeoKnow links for nodes (used for creating `owl:sameAs` relations between OSM and other sources).

We show some example mappings for nodes. Consider the mapping for the table `lgd_node_tags_resource_kv` storing information on object properties for nodes. First, we show the mapping in R2RML, which is supported by both Sparqlify and Ontop:

```
<lgd_node_tags_resource_kv> a rr:TriplesMap;
  rr:logicalTable [
    rr:sqlQuery "SELECT * FROM lgd_node_tags_resource_kv"
  ];
  rr:subjectMap [
    rr:template
      "http://linkedgeodata.org/triplify/node{node_id}";
    rr:termType rr:IRI
  ];
  rr:predicateObjectMap [
    rr:predicateMap [ rr:column "property";
                      rr:termType rr:IRI ];
    rr:objectMap [ rr:column "object";
                   rr:termType rr:IRI ];
  ].
```

This mapping can be written in the Ontop syntax as follows:

```
mappingId lgd_node_tags_resource_kv
target    lgd:node{node_id} <{property}> <{object}> .
source    SELECT * FROM lgd_node_tags_resource_kv
```

Below is the same mapping in the SML syntax:

```
Create View lgd_node_tags_resource_kv As
  Construct {
    ?s ?p ?o .
  }
  With
    ?s = uri(concat(lgd:node, ?node_id))
    ?p = uri(?property)
    ?o = uri(?object)
  Constrain
    ?p prefix "http://linkedgeodata.org/ontology/" "http:...
    ?o prefix "http://linkedgeodata.org/ontology/"
  From
    lgd_node_tags_resource_kv
```

Next we discuss how to map text to `xsd:langString`, i.e., strings with language tags. The example SML mapping below shows that it is possible to construct a literal with language from the database using `plainLiteral(?v,?language)`:

```
Create View lgd_node_tags_text As
  Construct {
    ?s ?p ?o .
  }
  With
    ?s = uri(concat(lgd:node, ?node_id))
    ?p = uri(?property)
    ?o = plainLiteral(?v, ?language)
  From
    lgd_node_tags_text
```

This is, however, not supported by R2RML and Ontop. In an Ontop (and R2RML) mapping, one has to explicitly enumerate all possible languages in LinkedGeoData. For instance, below is the mapping for the Italian language:

```
mappingId lgd_node_tags_text_lang_it
target    lgd:node{node_id} <{property}> {v}@it .
source    SELECT * FROM lgd_node_tags_text
          WHERE language = 'it'
```

This means that 3 such mappings (for nodes, ways, and relations) in SML need to be translated to a large number of mappings in Ontop and R2RML. This blowup makes the management of the mappings more difficult and can also cause performance issues.

We list in Fig. 5 the triples of the (Virtual) Knowledge Graph generated by the example mappings and ontology from the example OSM XML in Fig. 3.

### 5.4. SPARQL endpoint

Using the infrastructure described in Section 2, one can deploy easily an instance of LinkedGeoData with the developed ontology and mappings, and expose the SPARQL endpoints powered by Ontop and Sparqlify. The geographic area in the deployment is easily configurable by adjusting the download link of OSM as in Listing 1. The default area is the country of Monaco. We have deployed an instance (currently with only Monaco data) of the new version of LinkedGeoData online for demonstration.[32] Fig. 6 shows a screenshot of this deployment using Ontop, with an example query asking for a university and the restaurants around it.

## 6. Evaluation

In this section, we evaluate the performance of query answering over the LinkedGeoData VKGs created by both Sparqlify and Ontop-spatial v4. All the evaluation results are easily reproducible following the detailed online appendix.[33]

*Hardware.* The evaluation has been done on a machine with 4 cores (Intel(R) Xeon(R) Gold 6154 CPU @ 3.00 GHz), 16GB RAM, and 350GB SSD hard disk, running the operating system Ubuntu 18.04 and the DBMS PostgreSQL 12.3.

*Datasets.* We use three test geographical areas of North-East Italy, Italy, and Germany from OSM, which fit in our testing hard disk. The datasets were downloaded from Geofabrik[34] on 1 August 2020. The datasets are loaded to a PostgreSQL database as described in Section 5. The statistics on each OSM dataset, and the size of tables and views in the PostgreSQL database are shown in Table 3.

*Queries.* We have defined 7 GeoSPARQL query templates, as shown in Table 4, following common patterns of usage of LinkedGeoData. Each template has some parameters, which are either classes or spatial filters. The detailed queries and the parameters used for evaluation are provided in Appendix. For example, as shown in Table A.6, Q1 can be instantiated into 25 queries on each dataset (5 classes of "Amenities" $\times$ 5 locations). For Q1 to Q4, we also provide the SPARQL queries of the Sparqlify version that uses the corresponding `bif:` functions. For Q5 to Q7, this is not possible because there are no Sparqlify counterparts for the standard functions `geof:sfWithin`, `geof:sfContains`, and `geof:buffer`.

```
lgd:node6865960209 a lgdo:Amenity, geo:SpatialObject, lgdm:Node, geo:Feature, lgdo:University;
    rdfs:label "Università Internazionale di Monaco"@it, "International University of Monaco";
    lgdo:gadmSameAs <http://gadm.geovocab.org/services/withinRegion?lat=43.7344681000000008&long=7.41787760000000063#point>;
    lgdo:version "4"^^xsd:int;
    lgdo:wikipedia "fr:Université internationale de Monaco";
    dcterms:modified "2021-03-07T19:22:05"^^xsd:dateTime;
    dcterms:contributor lgd:user0;
    foaf:phone "+377 97 986 996";
    geo:hasGeometry lgdm-geom:node6865960209.

lgdm-geom:node6865960209 a geo:SpatialObject, geo:Geometry;
    geo:asWKT "POINT(7.4178776 43.7344681)"^^geo:wktLiteral .
```

**Fig. 5.** Example triples in the LinkedGeoData (Virtual) Knowledge Graph.

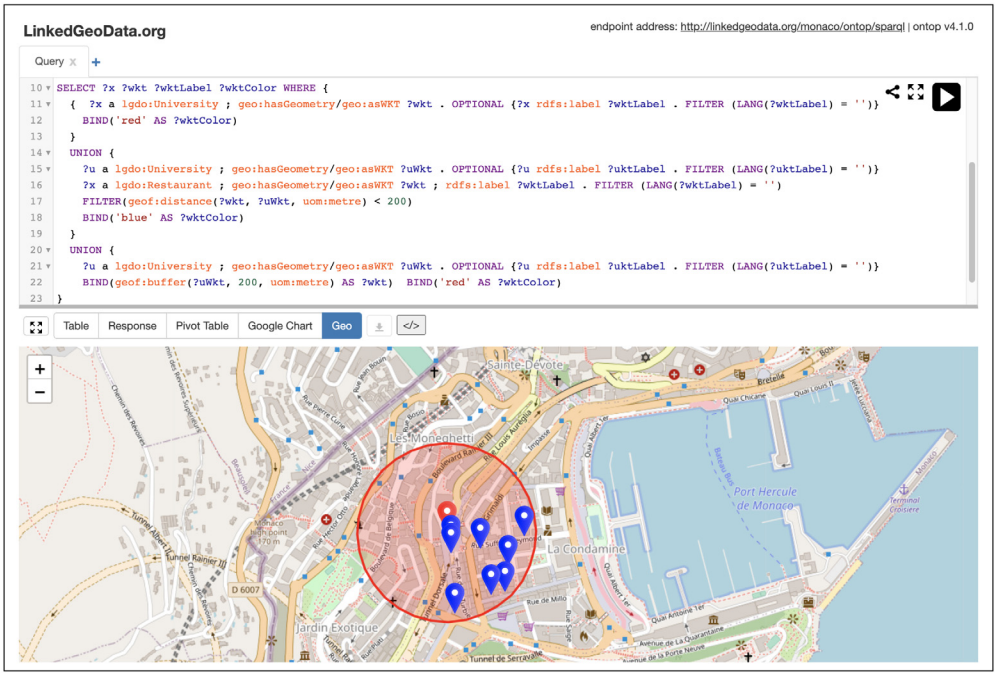


**Fig. 6.** SPARQL endpoint of LinkedGeoData powered by Ontop. This query retrieves universities and nearby restaurants in a 200 m radius.

**Table 4**
GeoSPARQL queries.

| | Operators | Query description |
|---|---|---|
| Q1 | Distance | Find OSM entities of a given class within a predefined distance |
| Q2 | Distance | Find OSM entities of a given class within a predefined distance from a given DBpedia location |
| Q3 | Intersection | Find OSM entities of a given class that intersect with a given polygon |
| Q4 | Intersection | Find OSM entities of two given linestring classes that intersect |
| Q5 | Within | Find OSM entities of a given class within a given polygon |
| Q6 | Contains | Find OSM entities of a given class that are contained in a given polygon |
| Q7 | Buffer+Within | Find OSM entities of a given class within a 500 metre buffer of a given location |

*Results.* The evaluation results are reported in Table 5 and Fig. 7. In Table 5, "OM" and "UF" refer to "out of memory" and "unsupported functions" respectively. Note that each number is an average of the running times with all possible parameters. We provide all the SQL queries rewritten by Ontop and Sparqlify from SPARQL queries in the online appendix.

For queries Q1 to Q4, Ontop performs better thanks to its optimization techniques. In particular, it can be seen from the online appendix that Sparqlify uses SQL queries from the mapping directly as subqueries and joins them, while Ontop performs sophisticated SQL translations using structural and semantic optimization techniques, which leads to queries that can be evaluated more efficiently by the database. Query Q4, which computes a huge number of intersections, can still be handled by Ontop (in between 20mins and 3 h), but Sparqlify runs out of memory. Ontop can also handle efficiently queries Q5, Q6, and Q7. Overall, the results indicate that the VKG approach in LinkedGeoData is able to support GeoSPARQL queries that combine topological and non-topological operations on the database.

## 7. Discussion & future work

This paper presents the latest development of the LinkedGeoData project using the virtual knowledge graph (VKG) technology. It confirms that VKG is an efficient and lightweight approach to expose large geodatasets as a unified Knowledge Graph. Below we discuss some issues we have encountered and present some future research directions.

*Language tags.* Through Sparqlify, all label mappings in a specific language can be performed with a single line `?o = plainLiteral`

[32] https://linkedgeodata.org/monaco/

[33] https://github.com/ontop/ontop-examples/tree/master/jows-2021-linkedgeodata

[34] http://download.geofabrik.de/

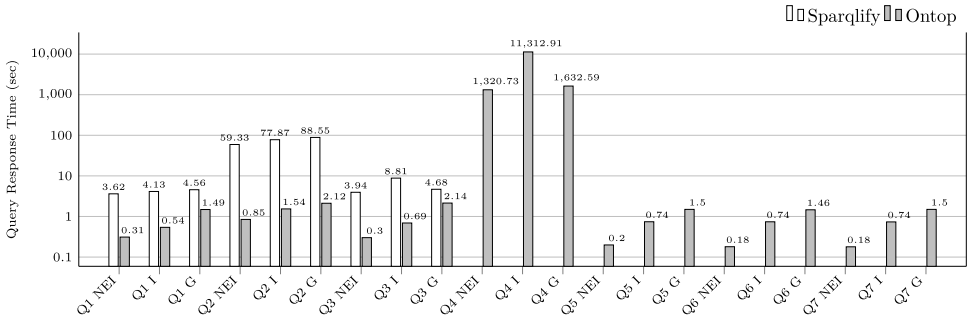


**Fig. 7.** Evaluation results. NEI = Northeast Italy, I = Italy, G = Germany.

**Table 5**
Average query response time (s).

| Dataset | Query | Sparqlify | Ontop |
|---|---|---|---|
| North-East Italy | Q1 | 3.616 | 0.311 |
| | Q2 | 59.329 | 0.846 |
| | Q3 | 3.944 | 0.301 |
| | Q4 | OM | 1320.733 |
| | Q5 | UF | 0.199 |
| | Q6 | UF | 0.18 |
| | Q7 | UF | 0.179 |
| Italy | Q1 | 4.131 | 0.54 |
| | Q2 | 77.873 | 1.536 |
| | Q3 | 8.807 | 0.691 |
| | Q4 | OM | 11312.905 |
| | Q5 | UF | 0.742 |
| | Q6 | UF | 0.74 |
| | Q7 | UF | 0.735 |
| Germany | Q1 | 4.564 | 1.485 |
| | Q2 | 88.553 | 2.117 |
| | Q3 | 4.677 | 2.141 |
| | Q4 | OM | 1632.594 |
| | Q5 | UF | 1.496 |
| | Q6 | UF | 1.461 |
| | Q7 | UF | 1.496 |

"OM" = "out of memory"; "UF" = "unsupported functions".

`(?label,?language)` to assign a label, i.e., data property and the respective language annotation at once. This feature is currently supported neither by R2RML nor by Ontop. R2RML would require the introduction of an attribute like `rr:languageColumn` on term maps in order to mitigate this issue. With Ontop a separate mapping for every language is required. Hence a large number of SQL unions has to be performed to accommodate all OSM languages. In LinkedGeoData, 89 mappings were needed for all the supported languages (including the case with no language). This issue has been discussed in the community[35] and we expect that the next version of the R2RML standard will include a feature to simplify the mapping of language annotations.

*Constraints over RDF terms.* A further feature of Sparqlify that currently has no counterpart neither in R2RML nor in Ontop is the declaration of IRI prefixes that term mappings can produce. For example, if a database column already contains IRIs (such as interlinks to DBpedia) and a triple mapping uses a term mapping of the form `?o = uri(?linkTarget)`, then without further metadata such a triple mapping qualifies as a candidate for answering any query's triple pattern that allows for an IRI in the respective component. However, if we inferred that a query's triple pattern can only match IRIs in one set of prefixes (such as `lgd`), and we also knew that the triple mapping produces IRIs in a different set of prefixes (such as `dbr`), then we can use this information to optimize the pruning of candidate triple mappings in the query rewriting phase.

*Faceted search.* A long-envisioned goal of LinkedGeoData is to facilitate live exploration of reasonably sized subsets of LinkedGeoData's Virtual Knowledge Graph, e.g., by means of client-side SPARQL-based *faceted search*, as demonstrated in [33]. Client-side SPARQL-based data exploration means that clients can be independent of LinkedGeoData and still enable exploration of its data in an ad-hoc fashion using only a single open standard protocol and query language, i.e., SPARQL. Due to the lack of support for aggregation functions in Sparqlify, this was so far not possible. Preliminary experiments with Ontop and our faceted search benchmark generator framework [34] showed that queries were already answered correctly, however the performance was not yet sufficient for interactive purposes. Hence, further analysis of the bottlenecks across LinkedGeoData's VKG stack together with the corresponding optimizations are worthwhile.

*Data quality.* A noteworthy issue with OSM is data quality, which is a result of the volunteered nature of the data collected. OSM reports that data quality is constantly improving also due to greater usage of open government data.[36,37] Still, the classes in the LinkedGeoData ontology derived from OSM data contain for example the following pairs of differentiated classes, which are obviously identical: "Vending+machine" vs "VendingMachine", "Wlan" vs "WLAN", and "Clothes%3A+women" vs "Clothes%3Awomen". We are going to tackle such data quality issues in the future.

*Improving interlinking.* A relevant point for future work is improving the interlinking. In [3], thousands of interlinks to DBpedia and GeoNames were first generated using an interlinking engine and subsequently manually verified, which is not a scalable model. Fortunately, nowadays the Wikidata[38] community maintains links from Wikipedia to OSM. Hence, in the near future, we will provide an extra docker container capable of performing continuous integration of LinkedGeoData-to-DBpedia links by regularly retrieving Wikidata-to-OSM links, storing them into the LinkedGeoData database, and exposing them through the VKG as well.

*Integration of the SANSA project.* SANSA (Semantic Analytics Stack)[39] is an open-source software project aimed at enabling analytics based on the RDF data model with open source Big Data frameworks (primarily Apache Spark[40]). Its architecture features

35 https://github.com/kg-construct/mapping-challenges/issues/18

36 https://welcome.openstreetmap.org/working-with-osm-data/how-good-is-osm/

37 https://www.missingmaps.org/osmstats/

38 https://wikidata.org/

39 http://sansa-stack.net/

40 https://spark.apache.org/

layers ranging from RDF import/export over SPARQL querying to machine learning. For querying, there are several partitioning strategies, some of which partition RDF data into Spark SQL tables, which are then mapped using R2RML. Previously, only Sparqlify was supported [24]. Ontop in SANSA is currently under evaluation, and it can be expected that its support for aggregation functions as well as the GeoSPARQL support in combination with the performance improvements will advance the state of the art in Big Data RDF processing. As it is easy to materialize compressed datasets from LinkedGeoData whose size exceeds hundreds of GB, SANSA may be a viable choice for performing analytics on LinkedGeoData datasets efficiently.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgements

This research has been partially supported by the EU H2020 project INODE (863410), by the Italian PRIN project HOPE (2017MMJJRE), by the European Regional Development Fund (ERDF) Investment for Growth and Jobs Programme 2014–2020 through the projects IDEE (FESR1133) and GEOBIMM (FESR1151), by the Free University of Bozen-Bolzano through the projects KGID (RTD 2019 Computer Science project), GeoVKG (CRC 2019 project), and STyLoLa (ID 2017 project) and through the Open Access Publishing Fund, and by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

## Appendix. GeoSPARQL queries for evaluation

We provide the seven GeoSPARQL query templates used for evaluation. Some templates use parameters `$class` and `$geo` for filtering based on OWL classes and locations, respectively. Possible values used in the experiments are given in Table A.6. All these queries are supported by Ontop. For Q1 to Q4, we also provide the Sparqlify variant.

*Query 1.* Find OSM entities of a given class within a predefined distance.

```
SELECT * WHERE {
  ?a a $class ;  geo:hasGeometry/geo:asWKT ?ag ;
     rdfs:label ?name .
  FILTER(lang(?name) = "" || lang(?name) = "it")
  BIND (geof:distance(
       '$geo'^^geo:wktLiteral, ?ag, uom:degree)
       AS ?distance)
  FILTER (?distance <= 0.1)
}
```

For Sparqlify, the filter is

```
FILTER(bif:st_intersects('$geo'^^geo:wktLiteral,
                         ?ag, 0.1)) .
```

*Query 2.* Find OSM entities of a given class within a predefined distance from a given DBpedia location.

```
SELECT ?ag ?name WHERE {
  ?a a $class ; geo:hasGeometry/geo:asWKT ?ag ;
     rdfs:label ?name .
  FILTER(lang(?name) = "" || lang(?name) = "it")
  ?b owl:sameAs
       <http://dbpedia.org/resource/$geo> ;
     geo:hasGeometry/geo:asWKT ?bg .
  BIND (geof:distance(?bg, ?ag, uom:degree)
        AS ?distance)
  FILTER (?distance <= 0.1)
}
```

For Sparqlify, the filter is

```
FILTER(bif:st_intersects(?bg, ?ag, 0.1)) .
```

*Query 3.* Find OSM entities of a given class that intersect with a given polygon.

```
SELECT * WHERE {
  ?a a $class ; geo:hasGeometry/geo:asWKT ?ag ;
     rdfs:label ?name .
  FILTER(lang(?name) = "" || lang(?name) = "it")
  FILTER(geof:sfIntersects(?ag,
                           '$geo'^^geo:wktLiteral))
}
```

For Sparqlify, the filter is

```
FILTER(bif:st_intersects(?g,
                         bif:st_geomFromText("$geo")))
```

*Query 4.* Find OSM entities of two given linestring classes that intersect.

```
SELECT * WHERE {
  ?a a lgdo:Motorway ;
     geo:hasGeometry/geo:asWKT ?ag ;
     rdfs:label ?aname .
  FILTER(lang(?aname) = "" || lang(?aname) = "it")
  ?b a lgdo:Canal ;
     geo:hasGeometry/geo:asWKT ?bg ;
     rdfs:label ?bname .
  FILTER(lang(?bname) = "" || lang(?bname) = "it")
  FILTER(geof:sfIntersects(?ag, ?bg))
}
```

For Sparqlify, the filter is

```
FILTER(bif:st_intersects(?ag, ?bg))
```

*Query 5.* Find OSM entities of a given class within a given polygon.

```
SELECT * WHERE {
  ?a a $class ; geo:hasGeometry/geo:asWKT ?ag ;
     rdfs:label ?name .
  FILTER(lang(?name) = "" || lang(?name) = "de")
  FILTER(geof:sfWithin(?ag, '$geo'^^geo:wktLiteral))
}
```

*Query 6.* Find OSM entities of a given class that are contained in a given polygon.

```
SELECT * WHERE {
  ?a a $class ; geo:hasGeometry/geo:asWKT ?ag ;
     rdfs:label ?name .
  FILTER(lang(?name) = "" || lang(?name) = "de")
  FILTER(geof:sfContains('$geo'^^geo:wktLiteral,?ag))
}
```

*Query 7.* Find OSM entities of a given class within a 500 m buffer of a given location.

```
SELECT * WHERE {
  ?a a $class ; geo:hasGeometry/geo:asWKT ?ag ;
     rdfs:label ?name .
  FILTER(lang(?aname) = "" || lang(?aname) = "it")
  BIND (geof:buffer('$geo'^^geo:wktLiteral, 500,
        uom:metre) AS ?cg)
  FILTER (geof:sfWithin(?ag, ?cg))
}
```

**Table A.6**
Parameters for the testing queries. "Amenities" = {`lgdo:Restaurant`, `lgdo:Bar`, `lgdo:Bank`, `lgdo:Pharmacy`, `lgdo:Library`}. The column "#" is the number of combinations of possible values.

| Dataset | Query | `$geo` | `$class` | # |
|---|---|---|---|---|
| North-East Italy | Q1 | POINT(11.0452369 45.886548) POINT(11.1257601 46.0664228) POINT(11.1594185 46.6695547) POINT(11.3547801 46.4981125) POINT(13.2358377 46.0634632) | Amenities | 25 |
| | Q2 | Rovereto, Trento, Bolzano, Udine, Merano | Amenities | 25 |
| | Q3 | POLYGON(11 46.45, 11.6 46.45, 11.6 46.64, 11 46.64, 11 46.45) | Amenities | 5 |
| | Q4 | | | 1 |
| | Q5 | POLYGON((11 46.45, 11.6 46.45, 11.6 46.64, 11 46.64, 11 46.45)) | Amenities | 5 |
| | Q6 | POLYGON((11 46.45, 11.6 46.45, 11.6 46.64, 11 46.64, 11 46.45)) | Amenities | 5 |
| | Q7 | POINT(13.2358377 46.0634632) | Amenities | 5 |
| Italy | Q1 | POINT(15.08738 37.5022355) POINT(11.0452369 45.886548) POINT(7.7748827 43.8198253) POINT(9.1128513 39.2169525) POINT(18.1718482 40.3570373) | Amenities | 25 |
| | Q2 | Cagliari, Catania, Lecce, Rovereto, Sanremo | Amenities | 25 |
| | Q3 | POLYGON((7.3 44.5, 8.5 44.5, 8.5 45.5, 7.3 45.5, 7.3 44.5)) | | 5 |
| | Q4 | | | 1 |
| | Q5 | POLYGON((7.3 44.5, 8.5 44.5, 8.5 45.5, 7.3 45.5, 7.3 44.5)) | Amenities | 5 |
| | Q6 | POLYGON((7.3 44.5, 8.5 44.5, 8.5 45.5, 7.3 45.5, 7.3 44.5)) | Amenities | 5 |
| | Q7 | POINT(18.1718482 40.3570373) | Amenities | 5 |
| Germany | Q1 | POINT(6.6441878 49.7596208) POINT(9.4333264 54.7833021) POINT(8.651177 49.872775) POINT(10.3166999 50.9833) POINT(11.3290855 50.9802813) | Amenities | 25 |
| | Q2 | Darmstadt, Eisenach, Flensburg, Trier, Weimar | Amenities | 25 |
| | Q3 | POLYGON((11.6 53.4, 13.65 53.4, 13.65 54.25, 11.6 54.25, 11.6 53.4)) | | 5 |
| | Q4 | | | 1 |
| | Q5 | POLYGON((11.6 53.4, 13.65 53.4, 13.65 54.25, 11.6 54.25, 11.6 53.4)) | Amenities | 5 |
| | Q6 | POLYGON((11.6 53.4, 13.65 53.4, 13.65 54.25, 11.6 54.25, 11.6 53.4)) | Amenities | 5 |
| | Q7 | POINT(11.3290855 50.9802813) | Amenities | 5 |

## References


[1] Jeremy Tandy, Linda van den Brink, Payam Barnaghi, Spatial Data on the Web Best Practices, W3C Working Group Note, W3C & OGC, World Wide Web Consortium & Open Geodata Consortium, 2017, URL https://www.w3.org/TR/sdw-bp/.

[2] Sören Auer, Jens Lehmann, Sebastian Hellmann, LinkedGeoData: Adding a spatial dimension to the web of data, in: Proc. of the 8th Int. Semantic Web Conf. (ISWC), in: Lecture Notes in Computer Science, vol. 5823, Springer, 2009, pp. 731–746, http://dx.doi.org/10.1007/978-3-642-04930-9_46.

[3] Claus Stadler, Jens Lehmann, Konrad Höffner, Sören Auer, LinkedGeoData: A core for a web of spatial open data, Semant. Web J. 3 (4) (2012) 333–354, http://dx.doi.org/10.3233/SW-2011-0052.

[4] Nicolas Tempelmeier, Elena Demidova, Linking OpenStreetMap with knowledge graphs—link discovery for schema-agnostic volunteered geographic information, Future Gener. Comput. Syst. 116 (2021) 349–364.

[5] Markus Nentwig, Michael Hartung, Axel-Cyrille Ngonga Ngomo, Erhard Rahm, A survey of current link discovery frameworks, Semant. Web J. 8 (3) (2017) 419–436, http://dx.doi.org/10.3233/SW-150210.

[6] Kevin Dreß ler, Axel-Cyrille Ngonga Ngomo, On the efficient execution of bounded Jaro-Winkler distances, Semant. Web J. 8 (2) (2017) 185–196, http://dx.doi.org/10.3233/SW-150209.

[7] Bayu Distiawan Trisedya, Jianzhong Qi, Rui Zhang, Entity alignment between knowledge graphs using attribute embeddings, in: Proc. of the 33rd AAAI Conf. on Artificial Intelligence (AAAI), vol. 33, 2019, pp. 297–304.

[8] Mohamed Sherif, Kevin Dreß ler, Panayiotis Smeros, Axel-Cyrille Ngonga Ngomo, Radon – Rapid discovery of topological relations, in: Proc. of the 31st AAAI Conf. on Artificial Intelligence (AAAI), 2017, pp. 175–181.

[9] Muhammad Saleem, Muhammad Intizar Ali, Aidan Hogan, Qaiser Mehmood, Axel-Cyrille Ngonga Ngomo, LSQ: the linked SPARQL queries dataset, in: Proc. of the 14th Int. Semantic Web Conf. (ISWC), in: Lecture Notes in Computer Science, vol. 9367, Springer, 2015, pp. 261–269, http://dx.doi.org/10.1007/978-3-319-25010-6_15.

[10] Guohui Xiao, Linfang Ding, Benjamin Cogrel, Diego Calvanese, Virtual knowledge graphs: An overview of systems and use cases, Data Intell. 1 (3) (2019) 201–223, http://dx.doi.org/10.1162/dint_a_00011.

[11] Guohui Xiao, Diego Calvanese, Roman Kontchakov, Domenico Lembo, Antonella Poggi, Riccardo Rosati, Michael Zakharyaschev, Ontology-based data access: A survey, in: Proc. of the 27th Int. Joint Conf. on Artificial Intelligence (IJCAI), IJCAI Org., 2018, pp. 5511–5519, http://dx.doi.org/10.24963/ijcai.2018/777.

[12] Ivan Ermilov, Michael Martin, Jens Lehmann, Sören Auer, Linked Open Data statistics: Collection and exploitation, in: Proc. of the 4th Int. Conf. on Knowledge Engineering and the Semantic Web (KESW), in: Communications in Computer and Information Science, vol. 394, Springer, 2013, pp. 242–249, http://dx.doi.org/10.1007/978-3-642-41360-5_19.

[13] Christian Bizer, Richard Cyganiak, D2RQ – Lessons learned, in: Proc. of the W3C Workshop on RDF Access To Relational Databases, 2007, Available at https://www.w3.org/2007/03/RdfRDB/papers/d2rq-positionpaper/.

[14] Freddy Priyatna, Oscar Corcho, Juan F. Sequeda, Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph, in: Proc. of the 23rd Int. World Wide Web Conf. (WWW), 2014, pp. 479–490, http://dx.doi.org/10.1145/2566486.2567981.

[15] Konstantina Bereta, Manolis Koubarakis, Ontop of geospatial databases, in: Proc. of the 15th Int. Semantic Web Conf. (ISWC), in: Lecture Notes in Computer Science, vol. 9981, Springer, 2016, pp. 37–52, http://dx.doi.org/10.1007/978-3-319-46523-4_3.

[16] Konstantina Bereta, Guohui Xiao, Manolis Koubarakis, Ontop-spatial: Ontop of geospatial databases, J. Web Semant. 58 (2019) http://dx.doi.org/10.1016/j.websem.2019.100514.

[17] Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, Guohui Xiao, Ontop: Answering SPARQL queries over relational databases, Semant. Web J. 8 (3) (2017) 471–487, http://dx.doi.org/10.3233/SW-160217.

[18] Guohui Xiao, Davide Lanti, Roman Kontchakov, Sarah Komla-Ebri, Elem Güzel-Kalayci, Linfang Ding, Julien Corman, Benjamin Cogrel, Diego Calvanese, Elena Botoeva, The virtual knowledge graph system Ontop, in: Proc. of the 19th Int. Semantic Web Conf. (ISWC), in: Lecture Notes in Computer Science, vol. 12507, Springer, 2020, pp. 259–277, http://dx.doi.org/10.1007/978-3-030-62466-8_17.

[19] John Herring, Simple Feature Access - Part 2: SQL Option, OGC Implementation Standard OGC 06-104r4, Open Geospatial Consortium, Open Geodata Consortium, 2010, URL https://www.ogc.org/standards/sfs.

[20] Dimitris Kontokostas, Patrick Westphal, Sören Auer, Sebastian Hellmann, Jens Lehmann, Roland Cornelissen, Amrapali Zaveri, Test-driven evaluation of linked data quality, in: Proc. of the 23rd Int. World Wide Web Conf. (WWW), 2014, pp. 747–758, http://dx.doi.org/10.1145/2566486.2568002.

[21] Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, et al., The FAIR guiding principles for scientific data management and stewardship, Sci. Data 3 (1) (2016) 1–9.

[22] Kleanthi Georgala, Mirko Spasic, Milos Jovanovik, Vassilis Papakonstantinou, Claus Stadler, Michael Röder, Axel-Cyrille Ngonga Ngomo, MOCHA2018: The mighty storage challenge at ESWC 2018, in: Revised Selected Papers of the 5th Semantic Web Challenges, in: Communications in Computer and Information Science, vol. 927, Springer, 2018, pp. 3–16, http://dx.doi.org/10.1007/978-3-030-00072-1_1.

[23] Claus Stadler, Jörg Unbehauen, Patrick Westphal, Mohamed Ahmed Sherif, Jens Lehmann, Simplified RDB2RDF mapping, in: Proc. of the Workshop on Linked Data on the Web (LDOW), in: CEUR Workshop Proceedings, vol. 1409, CEUR-WS.org, 2015, URL http://ceur-ws.org/Vol-1409/paper-09.pdf.

[24] Claus Stadler, Gezim Sejdiu, Damien Graux, Jens Lehmann, Sparklify: A scalable software component for efficient evaluation of SPARQL queries over distributed RDF datasets, in: Proc. of the 18th Int. Semantic Web Conf. (ISWC), in: Lecture Notes in Computer Science, vol. 11779, Springer, 2019, pp. 293–308, http://dx.doi.org/10.1007/978-3-030-30796-7_19.

[25] Stefan Brüggemann, Konstantina Bereta, Guohui Xiao, Manolis Koubarakis, Ontology-based data access for maritime security, in: Proc. of the 13th Extended Semantic Web Conf. (ESWC), in: LNCS, vol. 9678, Springer, 2016, pp. 741–757, http://dx.doi.org/10.1007/978-3-319-34129-3_45.

[26] Evgeny Kharlamov, Dag Hovland, Martin G. Skjæ veland, Dimitris Bilidas, Ernesto Jiménez-Ruiz, Guohui Xiao, Ahmet Soylu, Davide Lanti, Martin Rezk, Dmitriy Zheleznyakov, Martin Giese, Hallstein Lie, Yannis E. Ioannidis, Yannis Kotidis, Manolis Koubarakis, Arild Waaler, Ontology based data access in Statoil, J. Web Semant. 44 (2017) 3–36, http://dx.doi.org/10.1016/j.websem.2017.05.005.

[27] Linfang Ding, Guohui Xiao, Diego Calvanese, Liqiu Meng, Consistency assessment for open geodata integration: An ontology-based approach, GeoInformatica (2019) 1–26, http://dx.doi.org/10.1007/s10707-019-00384-9.

[28] Linfang Ding, Guohui Xiao, Diego Calvanese, Liqiu Meng, A framework uniting ontology-based geodata integration and geovisual analytics, Int. J. Geo-Inform. 9 (8) (2020) http://dx.doi.org/10.3390/ijgi9080474.

[29] Konstantina Bereta, Hervé Caumont, Ulrike Daniels, Erwin Goor, Manolis Koubarakis, Despina-Athanasia Pantazi, George Stamoulis, Sam Ubels, Valentijn Venus, Firman Wahyudi, The Copernicus App Lab project: Easy access to Copernicus data, in: Proc. of the 22nd Int. Conf. on Extending Database Technology (EDBT), OpenProceedings.org, 2019, pp. 501–511, http://dx.doi.org/10.5441/002/edbt.2019.46.

[30] Matthew Perry, John Herring, GeoSPARQL - A Geographic Query Language for RDF Data, OGC Implementation Standard OGC 11-052r4, Open Geospatial Consortium, Open Geodata Consortium, 2012, URL http://www.opengeospatial.org/standards/geosparql.

[31] Knut Stolze, SQL/MM Spatial - The standard to manage spatial data in a relational database system, in: Datenbanksysteme Für Business, Technologie Und Web, Tagungsband Der 10. BTW-Konferenz (BTW), in: LNI, vol. P-26, GI, 2003, pp. 247–264, URL https://dl.gi.de/20.500.12116/30056.

[32] Konstantina Bereta, George Stamoulis, Manolis Koubarakis, Ontology-based data access and visualization of big vector and raster data, in: Proc. of the 2018 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), 2018, pp. 407–410, http://dx.doi.org/10.1109/IGARSS.2018.8518255.

[33] Claus Stadler, Michael Martin, Sören Auer, Exploring the web of spatial data with Facete, in: Proc. of the 23rd Int. World Wide Web Conf. (WWW), 2014, pp. 175–178, http://dx.doi.org/10.1145/2567948.2577022.

[34] Claus Stadler, Simon Bin, Lisa Wenige, Lorenz Bühmann, Jens Lehmann, Schema-agnostic SPARQL-driven faceted search benchmark generation, J. Web Semant. 65 (2020) 100614.