Ontop of Geospatial Databases

Konstantina Bereta and Manolis Koubarakis

National and Kapodistrian University of Athens
{Konstantina.Bereta,koubarak}@di.uoa.gr

Abstract. We propose an OBDA approach for accessing geospatial data stored in relational databases, using the OGC standard GeoSPARQL and R2RML or OBDA mappings. We introduce extensions to an existing SPARQL-to-SQL translation method to support GeoSPARQL features. We describe the implementation of our approach in the system ontop-spatial, an extension of the OBDA system Ontop for creating virtual geospatial RDF graphs on top of geospatial relational databases. We present an experimental evaluation of our system using and extending a state-of-the-art benchmark. To measure the performance of our system, we compare it to a state-of-the-art geospatial RDF store and confirm its efficiency.

1 Introduction

Currently, there is emerging interest of scientific communities from various domains that produce and process geospatial data (e.g., earth scientists) to publish data as linked data and combine it with other data sources. Responding to this trend, the Semantic Web community has been very active in the geospatial domain, proposing data models, query languages, and systems for the representation and management of geospatial data. Notably, this research has led to the development of extensions of RDF and SPARQL, such as stRDF/stSPARQL and GeoSPARQL, that handle geospatial data. Similarly, research on geospatial relational databases has been going on for a long time and has resulted in the implementation of several efficient geospatial DBMS.

Despite the extensive research performed in the fields of relational databases and the Semantic Web on the development of solutions for handling geospatial data efficiently, to the best of our knowledge, there is no OBDA system that enables the creation of virtual, geospatial RDF graphs on top of geospatial databases. This would be very useful for scientists that produce and process geospatial data, as they mainly store this data in relational geospatial databases (e.g., PostGIS) or in other geospatial data formats that are easily imported into such databases (e.g., shapefiles). With the existing solutions in place, these scientists are forced to materialize their data as RDF in order to publish it as linked data and/or use it in combination with other data sources. Sometimes this is not practical and discourages users from using Semantic Web technologies. This issue applies to the OBDA paradigm in general, but it has more impact in the

geospatial domain due to the reasons we have just described. We address these issues by extending the OBDA paradigm with geospatial support.

The contributions of this paper are the following:

- We introduce extensions to an existing SPARQL-to-SQL translation method in order to perform GeoSPARQL-to-SQL translation.
- We describe the implementation of our approach in the system Ontop-spatial, which is the first OBDA system for GeoSPARQL.
- We present an experimental evaluation of our system extending the benchmark Geographica [7], comparing the performance of ontop-spatial with the state-of-the-art geospatial RDF store Strabon [8]. The results show that, in most cases, ontop-spatial outperforms Strabon.

Ontop-spatial is available as free and open source software at the following link: `https://github.com/ConstantB/ontop-spatial`. It was developed for the Statoil use case of the EU FP7 project Optique[1], and then it was also used in the urban accountant, land management, and crisis mapping services of the EU FP7 project MELODIES[2], as well as in the maritime domain [4].

The organization of the rest of the paper is as follows. In Section 2 we present related work and background. In Section 3 we explain the GeoSPARQL-to-SQL translation. In Section 4 we present the system Ontop-spatial and we mention the real-world use cases in which it has been used. In Section 5 we present the experimental evaluation of our system. In Section 6 we conclude the presentation of our approach discussing its advantages and limitations, as well as its possible extensions.

2 Related work and background

The first area of work related to our own is research on extensions of the data model RDF and the query language SPARQL with geospatial features.

The data model stRDF and the query language stSPARQL are extensions of RDF and SPARQL 1.1 respectively, developed for the representation and querying of spatial [8] and temporal data (i.e., the valid time of triples [3]). Another framework that has been developed for the representation and querying of geospatial data on the Semantic Web is GeoSPARQL [2], which is an OGC standard. GeoSPARQL and stSPARQL were developed independently, but they have a lot of features in common, the most important of which are that they both adopt the OGC standards WKT and GML for representing geometries, and that they both support spatial analysis functions as extension functions. Their main differences derive from the fact that stSPARQL extends SPARQL 1.1, so it inherits and extends important features of SPARQL 1.1, providing support for spatial updates and spatial aggregates. Also, GeoSPARQL does not offer valid time support. Both stSPARQL and GeoSPARQL have extended SPARQL 1.1

[1] `http://optique-project.eu/`
[2] `http://www.melodiesproject.eu/`

with the topological functions defined in the OGC standard "OpenGIS Simple Feature Access for SQL" [1], and they also support the Egenhofer [6] and the RCC-8 [13] topological relation families as SPARQL 1.1 extension functions.

Since in the rest of the paper we will refer to the notation and concepts defined or followed by stSPARQL and GeoSPARQL, we briefly present them below for the convenience of the reader.

Spatial literal. A spatial literal represents the serialization of a geometry. In stSPARQL, it is a literal of type `strdf:geometry` or its subtypes `strdf:WKT` or `strdf:GML`, as defined in [8]. Similarly, in GeoSPARQL it is a literal of type `geo:wktLiteral` or `geo:gmlLiteral`.

Spatial term. A spatial term is either a spatial literal or a variable that can be bound to a spatial literal.

Spatial filter. A spatial filter is a Boolean binary function $SF(t1, t2)$, where $t1, t2$ are spatial terms and SF is one of the Boolean functions of the Geometry extension of GeoSPARQL, namely `geof:sfEquals, geof:sfDisjoint, geof:stIntersects, geof:sfTouches, geof:sfCrosses, geof:sfWithin, geof:sfContains, geof:sfOverlaps,` and the respective *Egenhofer* and *RCC8* relation functions. These functions are defined in the Requirements 22, 23 and 24 of the GeoSPARQL standard.

Spatial selection. A spatial selection in GeoSPARQL/stSPARQL is a SELECT query with a FILTER which is a Boolean binary function with arguments a variable and a constant.

Spatial join. A spatial join in these languages is a query with a FILTER which is a Boolean binary function whose all arguments are variables. The definition of the spatial join in SPARQL corresponds to the definition of the spatial join in the geospatial extensions of the relational model. In the rest of this paper, spatial joins will often be denoted as $\bowtie_{sf}$, where sf is a spatial filter.

In the context of this paper, we will only consider GeoSPARQL (and, as a result, the geospatial part of stSPARQL). GeoSPARQL consists of the following six components:

- The *Core component*, which defines high level RDFS/OWL classes for spatial objects.
- The *Topology vocabulary extension*, which defines RDF properties for asserting and querying topological relations between spatial objects.
- The *Geometry extension*, which defines RDFS data types for serializing geometry data, geometry-related RDF properties, and non-topological spatial query functions for geometry objects.
- The *Geometry Topology extension*, which defines topological query functions.
- The *RDFS entailment extension*, which includes the RDF and RDFS reasoning requirements.
- The *Query Rewrite extension*, which defines rules for transforming *qualitative* spatial queries into equivalent *quantitative* queries.

The work surveyed above on extending RDF and SPARQL with geospatial functionality also gave rise to the implementation of geospatial RDF stores such

as Parliament, uSeekM and Virtuoso, that implement a subset of GeoSPARQL, and Strabon [8] that implements both GeoSPARQL and stSPARQL.

There have also been systems that enable the translation of geospatial data from their native formats to RDF. GeoTriples [9] is a tool for the conversion of geospatial data from a variety of source formats (shapefiles, relational databases, XML files, etc.) to RDF using GeoSPARQL and stSPARQL vocabularies and R2RML mappings.

Another category of systems that are related to our work is SPARQL-to-SQL systems such as Ontop [14, 5], Ultrawrap [15], D2RQ[3] and Morph [12]. These systems offer no geospatial functionality.

3 GeoSPARQL-to-SQL translation

In the work described in [14, 5], the authors present techniques for SPARQL-to-SQL translation using R2RML mappings. In this paper we extend their approach to support the GeoSPARQL-to-SQL translation using R2RML mappings. In this section we briefly describe how we translate the spatial extensions introduced in GeoSPARQL to Datalog and then in turn to the respective spatial extensions of SQL. A more detailed presentation of our extensions to the work described in [14, 5] is omitted due to space and will appear in a longer version of this paper.

The work of [14, 5] in the context of OBDA system Ontop follows the same semantics as [11] for the translation of SPARQL to Datalog. Definition 20 in [14, 5] describes the valuation of filter expressions, considering only numeric binary operators in filters. We present below how to extend this definition by considering spatial filters as defined in GeoSPARQL.

Definition 1 *Evaluation of Spatial Filter Expressions.*

Let SF be a GeoSPARQL spatial filter, let v, u be variables, L_{gs} the set of literals of the datatypes `geo:wktLiteral` *and* `geo:gmlLiteral` *and $c \in L_{gs}$. The valuation of SF on a substitution θ returns one of three values $\top, \bot$ and ϵ as shown below.*

$$(SF(v,c))\theta = \begin{cases} \top & \text{if } v \in dom(\theta) \text{ and } SF(v\theta, c) = true \\ \epsilon & \text{if } v \notin dom(\theta) \text{ or } v\theta = null \\ \bot & \text{otherwise} \end{cases}$$

$$(SF(v,u))\theta = \begin{cases} \top & \text{if } v, u \in dom(\theta) \text{ and } SF(v\theta, u\theta) = true \\ \epsilon & \text{if } v \text{ or } u \notin dom(\theta) \text{ or } u\theta = null \text{ or } v\theta = null \\ \bot & \text{otherwise} \end{cases}$$

GeoSPARQL to Datalog. In the approach described in [14, 5], the SPARQL query is translated into a set of rules that comprise a Datalog program preserving the semantics of the original query. The translation algorithm is a modified version of the one presented in [11]. The intention behind this step is to optimise

[3] `http://d2rq.org/`

the query before it gets translated into an SQL query that is eventually executed by the DBMS. The deviations of the original SPARQL-to-SQL translation algorithm of [11] proposed in [14, 5] lead to a more compact encoding of rules, due to the fact that the final goal is to translate the Datalog program in SQL instead of executing it as in [11]. We follow the same approach and we extend the algorithm of [14, 5] to take into account the spatial filters defined above.

We extend the algorithm by introducing a new set of distinguished predicates, namely the *distinguished spatial predicates*. We define a distinguished spatial predicate for each GeoSPARQL spatial filter [2]. Then the GeoSPARQL to Datalog translation algorithm is like the algorithm of [14, 5] for SPARQL and results in Π_{QGS}, a Datalog program that corresponds to a geospatial query.

Datalog to SQL. In a similar way as in the GeoSPARQL-to-Datalog translation, we extend Definition 18 of [14, 5] in order to consider distinguished spatial predicates as well: Every distinguished spatial predicate occurring in a Datalog program Π_{QGS} is translated into the equivalent geospatial SQL operator.

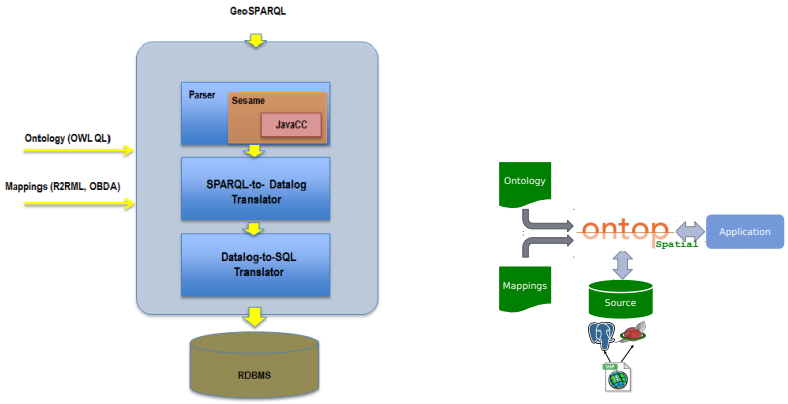
Mappings. In our framework we allow exactly the same mapping languages used in [14, 5], namely R2RML mappings and OBDA mappings (mapping language native to Ontop).

The mapping languages offer functionalities that are useful to in our geospatial setting. For example, when geometry columns (e.g, columns storing geometries in Well-Known-Binary format) of geospatial relational tables are present in the mappings, we allow geometries to be mapped as WKT GeoSPARQL literals. Similarly, we allow the presence of geospatial SQL operators in the mappings, enabling users to manipulate their geospatial data on-the-fly (e.g., transformation of the geometries into a different Coordinate Reference System) before they are mapped to RDF.

4 Implementation

We implemented the theoretical extensions of the SPARQL-to-SQL translation framework of [14, 5] discussed in Section 3 as an extension of the system Ontop with geospatial features focusing on *spatial selections* and *spatial joins*. We chose to extend Ontop instead of systems offering similar functionality because (i) it is open source, robust and extensible, (ii) it offers a wide range of functionalities that are useful for geospatial applications (reasoning, multiple APIs), and (iii) it implements significant SPARQL-to-SQL optimizations, producing queries that can be executed efficiently by the underlying DBMS as reported in [14, 5].

Ontop-spatial supports the following components of GeoSPARQL: *Core*, *Topology Vocabulary extension*, *Geometry Topology extension*, *RDFS entailment extension* and the spatial filters defined in the *Geometry Extension*. It is also, to the best of our knowledge, the first GeoSPARQL implementation that supports the *Query Rewrite extension* of GeoSPARQL. The high level architecture of the system as well as an abstract overview can be seen in Figures 1(a) and 1(b) respectively.



(a) Ontop-spatial architecture (b) Abstract overview

Fig. 1: Ontop-spatial

In the following, we highlight the components of Ontop that we have extended as they are placed in the query processing workflow:

- The virtual Ontop repository takes as input an ontology and a mapping file. Mappings can be either OBDA or R2RML.
- Once Ontop-spatial receives a GeoSPARQL query, the query gets parsed. We modified the Sesame parser used by Ontop (and the javacc parser that the respective Sesame library uses), in order to extend its syntax to support geospatial operations in the filter clause of the query. Additionally, qualitative geospatial queries, (i.e., queries containing geospatial triple patterns such as `ex:feauture1 geo:overlaps ex:feature2`) are also supported as standard SPARQL triple patterns, and get transformed into their quantitative equivalents (i.e., queries with spatial filters) in the following step.
- Conventionally, the next step in Ontop is to translate the SPARQL query and the R2RML mappings into a Datalog program so that the query can be represented formally and optimized following a series of optimization steps described in detail in [14, 5]. Ontop-spatial inherits these optimizations and extends the SPARQL-to-Datalog translation module. As explained in the previous section, the geospatial filters are transformed into Datalog using distinguished geospatial predicates. The same distinguished geospatial predicates are used in the case of the qualitative geospatial queries as well. As a result, both quantitative and qualitative representations of a GeoSPARQL query are transformed into the same SQL query in the following step.
- The optimized version of the query, as derived from the previous step, gets translated into SQL. Every geospatial Datalog predicate is mapped to the respective geospatial SQL operator, following the syntax of the underlying DBMS. The DBMS adapter has been extended in order to be able to identify geospatial columns in the database of the user. The PostgreSQL adapter has been modified and the Spatialite adapter has been added.

– The SQL query gets eventually executed in the underlying DBMS. Currently, the spatially-enabled databases that Ontop-spatial supports are the geospatial extensions of PostgreSQL and Sqlite, namely PostGIS and Spatialite respectively. More geospatial databases will be supported in the future.
– After the evaluation of the spatial SQL query in the DBMS, Ontop-spatial gets the results and sends them to the user. If geometries need to be projected, the SQL query that is produced returns the result as WKT. This enables Ontop-spatial to be used as a GeoSPARQL endpoint, that could serve as input endpoint for applications like linked geospatial data visualizers [10] to display the geometries that are returned as a result of a GeoSPARQL query.

Like the default version of Ontop, Ontop-spatial can be used as a web application (using Sesame workbench), as a Sesame library, as a Protege plugin, or it can be executed from the command line. The virtual geospatial graphs created by Ontop can also be materialized, creating an RDF dump, so that it can then be imported in a geospatial RDF store.

Ontop-spatial is available as free and open source software at the following link: `https://github.com/ConstantB/Ontop-spatial`.

Ontop-spatial in use. The motivation behind the development of Ontop-spatial was the Statoil use case of the project Optique, in order to address the issue of creating virtual RDF graphs on top of large databases that contain geometries and get frequently updated. Ontop-spatial is also being used in the urban accountant, land management and crisis mapping services of the EU FP7 project MELODIES[4]. Finally, ontop-spatial has recently be used in the Maritime security domain, in collaboration with Airbus and the University of Bolzano [4].

5 Evaluation

We conducted an empirical evaluation of our implementation based on the philosophy of Geographica[5], a benchmark for testing the performance of geospatial RDF stores [7]. Geographica consists of a *micro benchmark* and *a macro benchmark*. The *micro benchmark* is designed for testing basic geospatial operators, such as spatial selections and spatial joins. The *macro benchmark* tests the performance of the evaluated systems using queries that correspond to real application scenarios. As our aim is not to test geospatial RDF stores as done in [7], we use a modified benchmark based on the micro benchmark of Geographica as we explain later in this section.

Since there was no alternative OBDA systems that allow for posing GeoSPARQL queries over geospatial relational databases, we decided to evaluate Ontop-spatial in comparison with a geospatial RDF store. We consider that the spatiotemporal RDF store Strabon [8] is a good representative of the

[4] `http://www.melodiesproject.eu/software-tools`
[5] `http://geographica.di.uoa.gr/`

family of the geospatial RDF stores to compare with as (i) it is a state-of-the-art geospatial RDF store both in terms of functionality and performance [8, 7] (ii) it supports a big subset of GeoSPARQL (apart from stSPARQL), and (iii) it uses a spatially-enabled DBMS as back-end, performing a SPARQL-to-SQL translation following a specific storage scheme as explained in [8]. This enables us to use the same DBMS (PostGIS with the same configuration and tuning) and perform a comprehensive comparison.

5.1 Datasets

Geospatial data come, in most cases, in native geospatial data formats. In a real-world scenario, a user that works with geospatial data obtains it as files in a geospatial data format (e.g., a shapefile) and stores it either in a GIS or a spatially-enabled relational database. Later on, he may convert the data into RDF and store it in a geospatial RDF store in order to combine it with other linked data.

The benchmark Geographica is based on such real-world geospatial application scenarios and for the experimental evaluation of Ontop-spatial we will also follow this approach: We will import real geospatial datasets in a spatially-enabled relational database and use it as the back-end of Ontop-spatial.

We chose to use the datasets of Geographica that are available in their original format (shapefiles). These datasets are the Corine Land Cover dataset of Greece, which is provided by the European Environment Agency (EEA), the Greek Administrative Geography (GAG), and the Hotspots dataset provided by the National Observatory of Athens. We complemented these data sources with the original raw files of OpenStreetMap data about Greece which are available as shapefiles.[6] Geographica uses the RDF versions of the same subset of the OSM datasets created by the project LinkedGeoData[7]. For the rest of this paper, we will refer to this dataset using the acronym of the resulting, RDF-ized version (LGD). We added more OSM categories to our workload (e.g., buldings, waterways, etc.), as we will exploit the fact that each one is contained in a different shapefile (so it will be imported into a different table), to stress our system as we explain later on in this section.

For the evaluation of Ontop-spatial, we imported the shapefiles in a PostGIS database using the `shp2pgsql` command as described here: `https://github.com/ConstantB/Ontop-spatial/wiki/Shapefiles`. In this way, each shapefile is loaded into a separate table in the database. Each one of these tables contains a column where geometries are stored in binary format (WKB) and an index has been built on that column. Then, we created the minimum set of mappings in order to pose the queries of the benchmark. We used PostgreSQL version 9.1.13 and PostGIS 2.0.3, performing the fine tuning configurations suggested here: `http://geographica.di.uoa.gr`.

[6] `http://download.geofabrik.de/europe/greece.html`
[7] `http://linkedgeodata.org/`

Table 3 shows information about the datasets described above, such as the disk size that each of these tables occupy, the number of tuples and the average number of points per geometry. Notice that the LGD dataset consists of 7 shapefiles/tables which is important in the OBDA setting as we will explain later on. Also, LGD-Places and LGD-Points contain only point geometries.

In order to compare the performance of our system with Strabon, we materialized the virtual geospatial RDF graphs produced by Ontop-spatial and stored them in Strabon, so that both the virtual RDF graphs produced by Ontop-spatial and the graphs stored in Strabon contain exactly the same information. The produced RDF dump consists of 5.620.482 triples and contains 855.502 geometries. The total PostGIS database size (in terms of disk usage) of Ontop-spatial is 700 MB. The respective size of the PostGIS database that was produced after loading the RDF dump to Strabon is 1665 MB, which is more than twice the disk space compared to the original database produced by importing the shapefiles directly. The reason is that in the first case the database stores the data, while in the second case the database stores the equivalent set of triples. This kind of overhead is common in RDF stores that use a relational database as back-end. Also, Strabon inherits the *per_predicate* storage scheme of the Sesame RDBMS package, so every predicate is stored in a different table and additional tables are used for dictionary encoding. According to this storage scheme, all geometries are stored in a table called *geo_values* in WKB format and the respective column is indexed using an R-tree-over-GiST index, as described in [8].

5.2 Queries

The GeoSPARQL queries that we used for the experimental evaluation of our system are a set of *spatial selections* and a set of *spatial joins*. We used some of the queries of Geographica, and some queries that are appropriate in the OBDA setting as we will explain in the rest of this section. The queries used in our evaluation are presented in Tables 1 and 2. Each query has a numeric identifier, a mnemonic label, a number that shows how many BGPs it consists of and a number that shows how many results it returns.

Both spatial selection and spatial join queries contain a spatial filter that checks if a spatial relation holds between two geometries that are given as arguments to the respective GeoSPARQL function. In the case of spatial selections, one of the arguments is a variable and the other one is a constant, which can be either a line (queries suffixed with "L" in the query label) or a polygon (using "P" suffix). In spatial join queries, both arguments of the respective spatial binary operator are variables. The first set of queries that we consider contains simple geospatial queries, i.e., queries consisting of a single triple pattern to retrieve the geometries of a dataset and a spatial filter (spatial selections 00-14 and spatial joins 00-03). Note that spatial joins require at least two triple patterns to retrieve the geometries that will be bound to the variables that are involved in the spatial filter. This kind of queries test the response time of the compared systems to perform "pure" geospatial queries (i.e., involving the least possible

```
mappingId       lgd_buildings_geometry
target          lgd:{gid} lgd:asWKT {geom}^^geo:wktLiteral .
source          select gid, geom from buildings

mappingId       lgd_landuse_geometry
target          lgd:{gid} lgd:asWKT {geom}^^geo:wktLiteral .
source          select gid, geom from landuse
```

Fig. 2: Examples of geospatial mappings for two LGD tables

```
select ?s1 ?o1 where {
       ?s1 lgd:asWKT ?o1 .
 filter(geosparql:FUNCTION(SPATIAL_CONSTANT,?o1)).}
```

Fig. 3: Template for spatial selection queries

number of triple patterns, focusing as much as possible on the evaluation of the spatial condition).

The next set of queries that we consider tackles an important issue that is crucial in OBDA systems: the generation of `Union` operators, deriving from the ontology and the schema of the database in the SPARQL-to-SQL translation phase. For example, the LGD dataset consists of 7 shapefiles, each one containing a column where geometries are stored. But according to the ontology, the data property that connects a spatial object with its geometry is universal for all spatial objects in the dataset. We present the mappings for two of these tables/shapefiles in Figure 2.

Let us now consider the template for spatial selection queries in Figure 3. The translated SQL query corresponding to a GeoSPARQL query following this template would create unions in order to fetch results deriving from all the tables it has been mapped to, that is, all seven LGD tables, and then apply the spatial selection to this union. This is the case for spatial selection queries 15-19. In order to test how our system responds by increasing/decreasing the number of unions produced in the translated query, we add an additional, thematic filter that selects a different number of LGD categories each time, thus affecting a different number of tables, and producing different number of unions, respectively. For example, consider query 19 which is shown in Listing 1.1, which contains an OR-condition in the second filter, so the respective translated query contains a union.

Listing 1.1: Query 19

```
select distinct ?s1  where {
?s1 lgd:asWKT ?o1 .
?s1 rdf:type ?type .
filter(geof:sfIntersects(GEOMETRY,?o1))
filter( ?type = lgd:Road ||
?type = lgd:Waterway ) }
```

Listing 1.2: Spatial join query 6

```
select ?s1 ?s2 where {
?s1 lgd:asWKT ?o1 .
?s2 lgd:asWKT ?o2 .
(geo:sfIntersects(?o1,?o2))
}
```

The queries 15, 16, 17, and 18 produce 6, 4, 3, and 4 unions respectively. The presence of unions has a negative impact on the query response time, but things get even worse when unions appear in spatial joins (e.g., spatial join query 6). Since variables appear in the spatial filters that serve as the conditions of the spatial joins, all combinations of the respective tables that are involved in the corresponding mappings should be spatially joined pairwise. For example, consider the spatial join query 6 which is given in Listing 1.2. This query performs a spatial join with the condition `intersects` in *all* LGD tables that are involved in the mappings containing the predicate *lgd:asWKT*. This join is translated into the corresponding relational algebra expression as follows:

$(L_{buildings} \cup L_{luse} \cup ... \cup L_{waterways}) \bowtie_{sf} (L_{buildings} \cup L_{luse} \cup ... \cup L_{waterways})$

where $L_{buildings}$, L_{luse},..., $L_{waterways}$, etc are LGD tables and sf is spatial operator corresponding to $geof : sfIntersects$ from the query. The query engine evaluates this relational algebra expression as unions of joins and all involved tables get spatially joined pairwise.

Last, in order to measure how the selectivity of the queries affect the performance of the systems, we included the spatial selection queries 20 and 21 involve the computation of the intersection of all kinds of LGD areas with a specific polygon. This polygon is large in the case of spatial selection query 20 so that many geometries will be returned, while in spatial selection query 21 this polygon is small enough so that very few LGD areas intersect with it.

5.3 Results

Experimental set up. The experiments were carried out on a server with the the following specifications: Intel(R) Xeon(R) CPU E5620 @ 2.40GHz, 12MB L3, RAID 5, 32GB RAM and OS: Ubuntu 12.04. All experiments were carried out with both cold and warm cache. Queries are first executed in cold cache and then in warm cache. The queries for which the system under test times out (the time out threshold is set to 40 minutes) are not executed in warm cache. All queries and code we used to execute the experiments in both systems, can be found in the "experiments" branch of the github repository of Ontop-spatial (folder "benchmark") at `https://github.com/ConstantB/Ontop-spatial`.

Query response time. The results of our experimental evaluation can be seen in Figures 4 - 5. Response time is measured in nanoseconds and presented in logarithmic scale. A general observation is that the query response time of Ontop-spatial is better than the one of Strabon, especially when big datasets are involved, both for spatial selections and spatial joins. Strabon times out after 40 minutes in spatial join queries 6 and 7. In spatial selection queries 2-5, although Ontop-spatial achieves better response time than Strabon in cold cache, it gets outperformed in warm cache, as intermediate results (which are not many as the dataset involved in this query is relatively small), are more likely to be found in the cache, increasing the hit rate of the cache and decreasing I/O requests. However, such differences between executions in warm and cold cache are eliminated in larger datasets. In what follows we explain why Ontop-spatial outperforms Strabon.

No	Query	#BGPs	results
00	Equals_GADM_P	1	0
01	Contains_GADM_P	1	9
02	Contains_GADM_P	1	0
03	Equals_GADM_L	1	1
04	Overlaps_GADM_L	1	0
05	Contains_GADM_L	1	0
06	Intersects_CLC_L	1	5
07	Contains_CLC_L	1	0
08	Equals_CLC_L	1	5
09	Overlaps_CLC_L	1	0
10	Overlaps_CLC_P	1	132
11	Intersects_CLC_P	1	533
12	Contains_CLC_P	1	401
13	Equals_CLC_P	1	0
14	Intersects_LGD_P	2	2749
15	Intersects_LGD_B	2	2749
16	Intersects_LGD_PL	2	2626
17	Intersects_LGD_P	2	2522
18	Intersects_LGD_LU	2	2722
19	Intersects_LGD_ROA	2	2387
20	Intersects_LGD_bigP	1	729189
21	Intersects_LGD_P2	3	5

Table 1: Spatial selections description

No	Query	#BGPs	results
00	Within_CLC_GADM	2	34114
01	Intersects_GADM_GADM	2	1556
02	Overlaps_GADM_CLC	2	17035
03	Intersects_LGD_GADM	3	154725
04	Intersects_LGD_LGD_Mus	4	2
05	Intersects_LGD_GADM	2	819319
06	Intersects_LGD_LGD	1	3686229
07	Crosses_LGD_LGD_Roads	4	178602

Table 2: Spatial joins description

Dataset	Size	Tuples	$Avg\frac{\#points}{geometry}$
CLC	283MB	44834	187.84
Hotspots	35 MB	37048	5
GAG	24 MB	326	3020.14
LGD-Buildings	42 MB	155474	6.5
LGD-Landuse	20 MB	40220	19.4
LGD-Places	2.4 MB	13043	1
LGD-Points	12 MB	61664	1
LGD-Railways	2 MB	4996	13.3
LGD-Roads	250 MB	514403	19
LGD-Waterways	16 MB	20565	39.84

Table 3: Workload characteristics

Listing 1.3: Spatial join query 2

```
select ?s1 ?s2 where {
?s1 clc:asWKT ?o1 .
?s2 gag:asWKT ?o2 .
 filter(geof:sfWithin(?o1, ?o2))}
```

Listing 1.4: Spatial join query 4

```
select ?s1 ?s2 where {
?s1 lgd:asWKT ?o1 .
?s1 rdf:type lgd:Building .
?s1 lgd:type "Museum" .
?s2 lgd:asWKT ?o2 .
?s2 rdf:type lgd:Landuse .
filter(geof:sfIntersects(?o1,?o2))}
```

Listing 1.5: Ontop-spatial SQL query

```
SELECT
 1 AS "s1QuestType", NULL AS "s1Lang",
 ('http://geo.linkedopendata.gr/clc/'
 || REPLACE(..... || '/') AS "s1",
 1 AS "s2QuestType", NULL AS "s2Lang",
 ('http://geo.linkedopendata.gr/gag/ont/'
 || REPLACE(...'/') AS "s2"
FROM
clc QVIEW1,
gag QVIEW2
WHERE
QVIEW1."gid" IS NOT NULL AND
QVIEW1."geom" IS NOT NULL AND
QVIEW2."gid" IS NOT NULL AND
QVIEW2."geometry" IS NOT NULL AND
(ST_Within(QVIEW1."geom",QVIEW2."geometry"))
```

Listing 1.6: Strabon SQL query

```
SELECT a0.subj,
u_s2.value,
a2.subj,
u_s1.value
FROM aswkt_855211 a0
INNER JOIN geo_values l_o2
ON (l_o2.id = a0.obj)
INNER JOIN geo_values l_o1 ON
 ((ST_Within(l_o1.strdfgeo,
 l_o2.strdfgeo)))
INNER JOIN aswkt_135992 a2
ON (a2.obj = l_o1.id)
LEFT JOIN uri_values u_s2
ON (u_s2.id = a0.subj)
LEFT JOIN uri_values u_s1
 ON (u_s1.id = a2.subj)
```

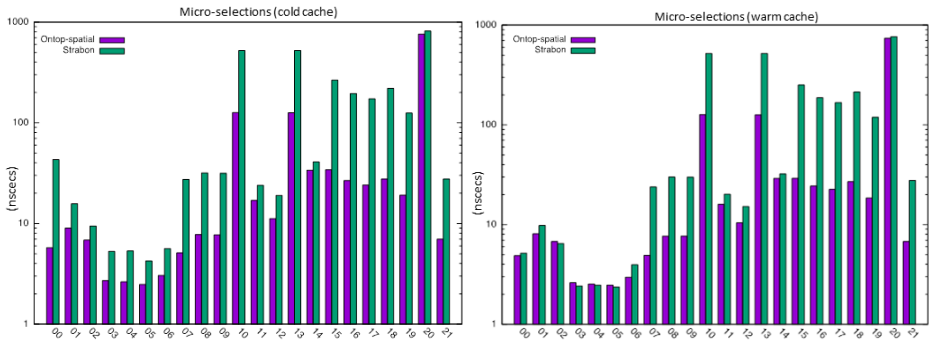


Fig. 4: Spatial Selections experiment (cold and warm cache)

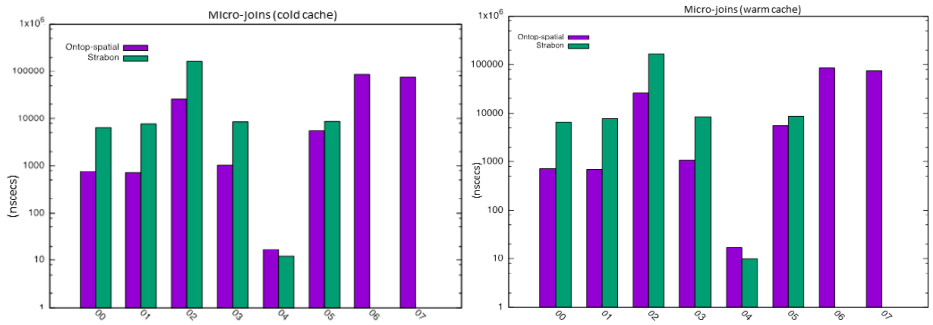


Fig. 5: Spatial Joins experiment (cold and warm cache)

The queries provided in Listings 1.5 and 1.6 are the SQL translations of the GeoSPARQL spatial join query 2, which is provided in Listing 1.3. One can observe that Ontop-spatial produces the same query as one would have written by hand in a geospatial relational database. Strabon produces some extra joins, as a result of the star schema that it follows in the database (and has been inherited from the Sesame RDBMS that Strabon is built on), i.e., each predicate is stored in a different table and there are some additional tables used for dictionary encoding (tables storing URIs, one table for each different datatype, etc.). This has a negative impact on performance when many intermediate results are produced. In Strabon, geometries are stored in a single table, named *geo_values*, and are indexed on the geometry column using an R-tree-over-GiST index. On the other hand, Ontop-spatial stores each shapefile in a different table, and geometries are stored in a sepate column for each table, and a separate R-tree-over-GiST index is constructed for the geometries of each shapefile/table. As Table 3 shows, there are cases where geometries of a shapefile/table are of the same type (e.g., all contain points/linestrings/polygons), allowing Ontop-spatial to build smaller and more efficient indices.

Nevertheless, in spatial join query 4, Strabon outperforms Ontop-spatial. The query is provided in Listing 1.4. Using this query, we want to retrieve the land use of areas that intersect with Museums. This is a very selective query with respect to the thematic condition, so the PostGIS optimizer correctly chooses to perform the thematic conditions first so that only the geometries of Museums will be checked in the spatial condition that follows, and the R-tree index will be used. Both systems execute the query very fast, with Strabon achieving nearly

4 times better performance than Ontop-spatial, as the overhead of the extra joins it performs, as described above, is reduced because very few intermediate results are produced. Also, dictionary decoding helps Strabon to perform string comparison (for value "Museum") only once, in order to retrieve the id of that value and then perform thematic joins efficiently using the id (numeric) value.

Queries 15-19 have filters that select different kinds of LGD categories. Query response time increases every time many LGD categories are involved (Query 15 asks for all categories), producing the respective number of unions in the case of Ontop-spatial and more intermediate results for Strabon, forcing more geometries to be checked in the spatial filter. On the contrary, query response time decreases when less LGD categories need to be selected.

The results of union-queries are more interesting in the case of spatial joins, shown in Figure 5. One would expect that unions with spatial joins, as in the case of the spatial join query 6, would dramatically decrease the performance of Ontop-spatial. Indeed, query response time increases in the case of queries like query 6, but Ontop-spatial still performs better than Strabon. The explanation for this lies in the fact that each time a spatial join is performed between two different LGD tables, the optimizer chooses the one having the smaller index (and usually smaller geometries, in this case) to be nested inside the inner branch of the nested loop, where it performs an index scan. This has greater impact on the execution time of geospatial queries, as the evaluation of spatial joins is more expensive due to the cost of the evaluation of the spatial conditions.

In spatial selection query 20, the performance of the two systems is very close, while in the more selective version of the same query, i.e., spatial selection query 21, the gap in the execution times between Ontop-spatial and Strabon increases again. This happens because nearly every geometry in the workload is included in the results of the spatial selection query 20, so spatial indices are not useful in this case.

Overall, we observe that importing the shapefiles to a database and then using an OBDA approach is very efficient, as in most cases, the information that is contained in a shapefile is compact and homogeneous, as we often have one shapefile per data source. So, the SQL queries that are produced based on such a schema contain reduced amount of joins and can be executed efficiently.

6 Discussion and Conclusions

In this paper, we describe how we extended the techniques of [14, 5] to develop the first geospatially-enabled OBDA system, named Ontop-spatial. By extending the OBDA system Ontop, Ontop-spatial inherits the advantages of using RDB2RDF systems in real use cases: i) RDB-to-RDF workflow becomes less complicated, without having to use different tools for converting data into RDF and then storing it in RDF stores, ii) no data needs to be transfered, as existing databases are used as input to the system, and iii) mappings provide a layer of abstraction between the data manipulation/database experts and the end users.

These advantages have even greater impact when dealing with geospatial data. The domains where geospatial data are produced and used are dominated by geospatial databases and other tabular file formats that could easily be imported to a database (e.g., shapefiles). GIS practitioners use geospatial relational databases in their day-to-day tasks, either directly or as the back-end of applications to store and manipulate data (e.g., GIS have connectors for geospatial relational databases). Ontop-spatial provides a solution for combining the advantages of geospatial relational databases, for example, the wide variety of geospatial data operators and the performance achieved by the use of spatial indices, with the data modeling advantages of the RDF data model. Moreover, Ontop-spatial allows for encapsulating geospatial data manipulation functions offered by geospatial extensions to SQL (e.g., functions for transforming geometries to a different coordinate reference system) in the mappings.

On the other hand, Ontop-spatial inherits the disadvantages of the OBDA systems as well. First, in order to combine information coming from different geospatial sources, the data should be imported in databases. Second, as the database is given as input to the system, it is read-only and Ontop-spatial does not support SPARQL store or update operations; all updates should be done directly on the database level. Third, the performance of the system is heavily dependent on the ontology, the schema of the database, and the mappings, as we explained in the previous sections, which applies for OBDA approaches in general. However, our experiments showed that in many cases, our geospatially enchanced OBDA approach achieves significantly better performance than the state-of-the-art geospatial RDF store Strabon. The main reasons for this are summarized as follows:

- The database schema that is produced simply by importing the shapefiles to the database is in most cases suitable for OBDA approaches, as shapefiles contain compact and homogeneous information per dataset.
- The database produced by storing the materialized RDF dump that ontop exports in Strabon is bigger than the database that results from importing the shapefiles, even though only the RDF triples that were involved in the OBDA mappings (i.e., the *virtual* RDF triples) were exported. This happens because of i) the normalization imposed by the RDF data model itself (i.e., triples) and ii) the additional tables used for dictionary encoding.
- The additional joins that are created in the translated SQL queries of Strabon and the fact that geometries are stored in a single table where geospatial operators are performed increase even by more than an order of magnitude in very large workloads with many and complicated geometries, when many intermediate results are produced in queries.

In future work, we plan to continue the development of Ontop-spatial in the directions of i) fully supporting GeoSPARQL and stSPARQL (i.e., adding also valid time support), and ii) creating a distributed version of our extension exploiting the fact that the union-all spatial queries are parallelizable.

Acknowledgement. This work is partially supported by the EU projects Optique (318338) and MELODIES (603525). We would like to thank the Ontop development team for their support.

References

1. Open Geospatial Consortium. OpenGIS Simple Features Specification For SQL. OGC Implementation Standard (1999)
2. Open Geospatial Consortium. GeoSPARQL - A geographic query language for RDF data. OGC Candidate Implementation Standard (2012)
3. Bereta, K., Smeros, P., Koubarakis, M.: Representation and Querying of Valid Time of Triples in Linked Geospatial Data. In: Extended Semantic Web Conference 2013. vol. 7882, pp. 259–274. Springer Berlin Heidelberg (2013)
4. Bruggemann, S., Bereta, K., Xiao, G., Koubarakis, M.: Ontology-Based Data Access for Maritime Security, pp. 741–757. Springer International Publishing (2016)
5. Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodriguez-Muro, M., Xiao., G.: Ontop: Answering SPARQL queries over relational databases. Semantic Web Journal, 2016. (To appear)
6. Egenhofer, M.: A formal definition of binary topological relationships. In: Foundations of Data Organization and Algorithms, Lecture Notes in Computer Science, vol. 367, pp. 457–472. Springer Berlin Heidelberg (1989)
7. Garbis, G., Kyzirakos, K., Koubarakis, M.: Geographica: A Benchmark for Geospatial RDF stores (long version). Lecture Notes in Computer Science, vol. 8219, pp. 343–359. Springer (2013)
8. Kyzirakos, K., Karpathiotakis, M., Koubarakis, M.: Strabon: A Semantic Geospatial DBMS. In: ISWC. LNCS, vol. 7649, pp. 295–311. Springer (2012)
9. Kyzirakos, K., Vlachopoulos, I., Savva, D., Manegold, S., Koubarakis, M.: Geotriples: a tool for publishing geospatial data as RDF graphs using R2RML mappings. In: Proceedings of the ISWC 2014 Posters & Demonstrations Track , Riva del Garda, Italy, October 21, 2014. pp. 393–396 (2014)
10. Nikolaou, C., Dogani, K., Bereta, K., Garbis, G., Karpathiotakis, M., Kyzirakos, K., Koubarakis, M.: Sextant: Visualizing time-evolving linked geospatial data. J. Web Sem. 35, 35–52 (2015)
11. Polleres, A.: From SPARQL to rules (and back). In: Proceedings of the 16th International Conference on World Wide Web. pp. 787–796. WWW '07, ACM, New York, NY, USA (2007)
12. Priyatna, F., Corcho, O., Sequeda, J.: Formalisation and experiences of R2RML-based SPARQL to SQL query translation using Morph. In: Proc. of the 23rd International Conference on World Wide Web. pp. 479–490. ACM, NY, USA (2014)
13. Randell, D.A., Cui, Z., Cohn, A.G.: A spatial logic based on regions and connection. In: Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92). Cambridge, MA, October 25-29, 1992. pp. 165–176 (1992)
14. Rodriguez-Muro, M., Rezk, M.: Efficient SPARQL-to-SQL with R2RML mappings. Journal of Web Semantics 33(1) (2015)
15. Sequeda, J., Miranker, D.P.: Ultrawrap: SPARQL execution on relational data. Journal of Web Semantics 22 (2013)