Efficient Management for Geospatial and Temporal Data using Ontology-based Data Access Techniques

Konstantina Bereta*

National and Kapodistrian University of Athens, Greece
Department of Informatics and Telecommunications
`konstantina.bereta@di.uoa.gr`

Abstract. In this thesis, we propose an Ontology-based Data Access (OBDA) approach for accessing geospatial data stored in geospatial 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. Ontop-spatial is the first geospatial OBDA system and it outperforms state-of-the-art geospatial RDF stores. We also show how to answer queries with temporal operators in the OBDA framework, by utilizing the framework stRDF and the query language stSPARQL which we extend with some new features. Next, we extend the OBDA paradigm going beyond relational database management systems, and we present our OBDA solutions for creating virtual RDF graphs on top of various web data sources (e.g., HTML tables, Web APIs) using ontologies and mappings. Last, we describe how the work described in this thesis is applied in real-world application scenarios.

Keywords: Linked spatiotemporal data · Spatiotemporal databases · Semantic Web

1 Dissertation Summary

1.1 Motivation and Scope

The volume of geospatial and temporal data that is produced and consumed from applications is ever-increasing these years, as more and more applications use geospatial data that is produced from various scientific fields (e.g., Earth Observation, environment studies, earth sciences). At the same time, many applications and services, such as Google maps, heavily rely on the efficient integration and management of geospatial data from heterogeneous sources.

* Dissertation Advisor: prof. Manolis Koubarakis, Professor

Many research efforts during the past decades have concentrated in the efficient management and retrieval of geospatial data, resulting in the development of data models, query languages and systems with spatial and temporal support. In the area of the database community, the relational model has been extended with primitives that represent geospatial and temporal data, followed by the extension of SQL with geospatial and temporal support. These efforts led to the development of a wide variety of geospatially and temporally-enabled Database Management Systems (e.g., PostGIS, PostGIS-temporal, Spatialite, Oracle Spatial and Graph), while a lot of work in literature addresses the problem of spatial and temporal query optimisation.

In the process of data integration from different sources, a common data model should be used. For this purpose, the data model RDF [22] has been developed. The rationale behind the creation of the data model RDF is that every entity on the Internet is a resource that can be identified using a unique identifier (URI). This resource can then be described using triples. Triples are statements that consist of three parts: The *subject*, which is the resource that we want to describe, the *predicate*, and the *object*. The predicate denotes a property of the subject, and it is also a resource that can be described, while the object expresses the value of the property of the subject.

Ontologies are used to describe the schema of the data encoded in RDF. They describe the different kinds of entities that might exist in a dataset, i.e., the classes, the relations between them, the properties and the relations between the properties, and they might also include axioms that introduce additional definitions and constraints to the data. The query language that is used to query data encoded in RDF format is the SPARQL [17] query language, which is a declarative query language that shares many common syntactic primitives with SQL.

The data model RDF and the SPARQL query language do not support primitives or operators for the representation and querying of geospatial and temporal data. For this reason, many geospatial and temporal extensions of the framework of RDF and SPARQL started to emerge in the recent years. The highlights of these efforts are the following: (i) the development of the framework of stRDF and stSPARQL [20] that extends the data model RDF and the SPARQL query language with geospatial and temporal support, and (ii) the establishment of the query language GeoSPARQL [13].

The query language GeoSPARQL is a geospatial extension of the framework of RDF and SPARQL which introduced geospatial primitives and operators for the representation and querying of geospatial data. Soon after its development, GeoSPARQL was standardised by the Open Geospatial Consortium (OGC). GeoSPARQL extends the data model RDF introducing the following datatypes for the representation of geometries as literals: The Well-Known-Text (WKT) datatype, and the Geography Markup Language (GML) datatype. The WKT and GML serialisations are OGC standard formats for the representation of geometries as text. The query language GeoSPARQL also defines predicates that represent topological relations between resources with spatial extent or between geometries, while it also defines a set of operators that can be used as functions that extend SPARQL 1.1. These functions take one or two geometries as arguments, they calculate geometry metrics (e.g., area) or topological relations (e.g., overlapping geometries), and they can be included in filter or select clauses of SPARQL queries.

The data model stRDF and the query language stSPARQL is another extension of the framework of RDF and SPARQL with spatial support. The data model stRDF and the query language stSPARQL were developed at the same time but independently from GeoSPARQL, and shares some common characteristics.

Soon after the standardisation of the query language GeoSPARQL, the first GeoSPARQL implementations started to emerge, such as the following systems: GraphDB[1], Oracle Spatial and Graph[2], USeekM[3], while other triple-stores introduced spatial support using native spatial primitives. The system Strabon[4] [21] outperforms all of these systems, according to recent benchmarks [16] but it is also the most rich in functionalities, as it supports both GeoSPARQL and stSPARQL.

Although the data model RDF is widely there are cases when the data is stored in large databases that get frequently updated and users are often discouraged to convert their data into RDF and materialise it as triples every time their datasets get updated.

The research area of Ontology-based Data Access (OBDA) [25] addresses this problem by developing techniques for creating virtual RDF triples on top of geospatial databases, without materialising the data as RDF triples stored in a triple-store. In this setting, ontologies are used to describe the relational data conceptually, while mappings encode how relational data get translated into (virtual) RDF terms. Mappings are encoded in languages such as R2RML [15], which is a W3C standard. Despite the availability of various OBDA systems nowadays, the OBDA paradigm did not include geospatial or temporal support. In this dissertation, we remedy this issue as we describe in the following sections.

1.2 Dissertation contributions

The contributions of this dissertation are summarised as follows:

- We present a novel approach for extending the OBDA paradigm with geospatial support. We implement our techniques in the system Ontop-spatial, the first OBDA system that includes geospatial support. Our extensive experimental evaluation shows that Ontop-spatial outperforms traditional state-of-the-art geospatial RDF stores often by two orders of magnitude [4,7]. After

[1] `https://www.ontotext.com/`
[2] `https://www.oracle.com/technetwork/database/options/spatialandgraph/overview/index.html`
[3] `https://www.w3.org/2001/sw/wiki/USeekM`
[4] `http://www.strabon.di.uoa.gr/`

the development of Ontop-spatial, similar functionality was also added in the system Oracle Spatial and Graph[5].

- We extend the framework of stRDF and stSPARQL with additional temporal operators that facilitate the adoption and implementation of the framework in the OBDA setting.
- We generalise our approach by extending the OBDA paradigm to allow for the creation of virtual RDF graphs on top of data that is available on the Web, for example as HTML tables or accessible through Web APIs [5]. We implement our techniques as an extension of the system Ontop-spatial and we conduct an experimental evaluation of our method compared to a state-of-the-art approach offering similar functionality [23]. The results of the evaluation show that our system outperforms the system in comparison.
- We showcase the use of the techniques described above in real-world applications. Soon after the development of our techniques in the system Ontop-spatial, the system was used in a variety of real-world use cases from different domains such as land management [8], urban development, maritime security [9], as well as for facilitating the development of APIs for Copernicus data [3].

2 Results and Discussion

2.1 Geospatial Ontology-based Data Access

In this section, we describe how we extended the OBDA paradigm to support spatial queries on top of geospatial databases. More specifiically, we present the techniques of answering GeoSPARQL queries in OBDA by translating to SQL queries, which is based on the SPARQL-to-SQL algorithm used in Ontop [19,10]. The pseudo code of algorithm is outlined in Figure 1. As in the classical case, the algorithm takes as inputs a (Geo)SPARQL query, an ontology T, and a mapping M, and returns a SQL query. The algorithm consists of *(1)* an offline step, which is query-independent and preprocess the mapping and ontology and generates the so-called saturated mapping or T-mapping, and *(2)* an online step, which translates the input SPARQL query into an SQL query. We refer the readers to [10] for more details of the workflow. In the following, we discuss the GeoSPARQL specific steps, which are underlined in the pseudo code.

Ontology classification At line 2, the algorithm classifies the input ontology T union with the GeoSPARQL ontology $Tgeo$, and construct an explicit hierarchy of classes and properties. By assuming the built-in ontology $Tgeo$, the algorithm is able to support the Clauses $6-8$ of the GeoSPARQL standard.

GeoSPARQL query-rewrite At line 5, the algorithm expands the input GeoSPARQL query q using the rules *Rgeo*. We note that the resulting query is of polynomial size of the input query.

[5] `https://www.oracle.com/database/technologies/spatialandgraph.html`

Algorithm 1 Algorithm of Translating GeoSPARQL into SQL

```
Input: GeoSPARQL query $q$, Ontology $T$, Mapping $M$
Output: An SQL expression
1: // offline phase
2: $\underline{T_{classified} = \text{classify}(T \cup Tgeo)}$    ▷ ontology classification
3: $\underline{M_T \leftarrow saturate(M, T_{classified})}$    ▷ mapping saturation
4: // online phase
5: $\underline{Q \leftarrow rew_{geo}(q)}$    ▷ GeoSPARQL query write
6: $S \leftarrow$ list of nodes in $Q$ in a bottom-up topological order
7: sql $\leftarrow$ empty map from nodes to SQL expressions
8: for node $n \in S$ do
9:     if $n$ is triple pattern then    ▷ translating leaves
10:        sql$[n] \leftarrow$ replace-Tmap-def($n$, $M_T$)
11:    else    ▷ translating non-leaf nodes
12:        if $n$ = JOIN($n_1$,$n_2$) then
13:            sql$[n] \leftarrow$ InnerJoin(sql$[n_1]$, sql$[n_2]$)
14:        else if $n$ = OPTIONAL($n_1, n_2, e$) then
15:            sql$[n] \leftarrow$ LeftJoin(sql$[n_1]$, sql$[n_2]$, $e$)
16:        else if $n$ = UNION($n_1, n_2$) then
17:            sql$[n] \leftarrow$ Union(sql$[n_1]$, sql$[n_2]$)
18:        else if $n$ = FILTER($n_1, e$) then
19:            $\underline{\text{sql}[n] \leftarrow \text{Filter}(\text{sql}[n_1], e)}$
20:        else if $n$ = PROJECT($n_1, p$) then
21:            sql$[n] \leftarrow$ Project(sql$[n_1], p$)
22:        end if
23:    end if
24: end for
25: return sql$[S$.last()$]$
```

Table 1: GeoSPARQL Simple Feature functions of to SQL Functions

`GeoSPARQL function`	`OGC SFS SQL function`
`geof:sfEquals`	`ST_Equals`
`geof:sfDisjoint`	`ST_Disjoint`
`geof:stIntersects`	`ST_Intersects`
`geof:sfTouches`	`ST_Touches`
`geof:sfCrosses`	`ST_Crosses`
`geof:sfWithin`	`ST_Within`
`geof:sfContains`	`ST_Contains`
`geof:sfOverlaps`	`ST_Overlaps`

Spatial filter expressions At line 19, the algorithm transforms the SPARQL filters to its SQL equivalences. Now it also translates GeoSPARQL functions to the corresponding functions in the spatial extension of SQL. In Table 1, we provide a list of SPARQL Simple Feature functions defined in GeoSPARQL and their equivalences in SQL functions defined in OpenGIS SQL standard [18]

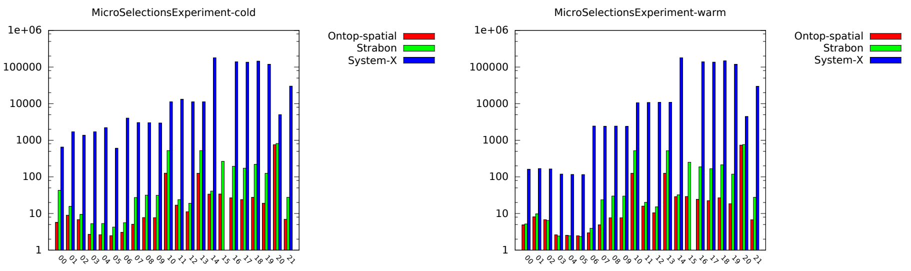

Fig. 1: Spatial Selections experiment (cold and warm cache)

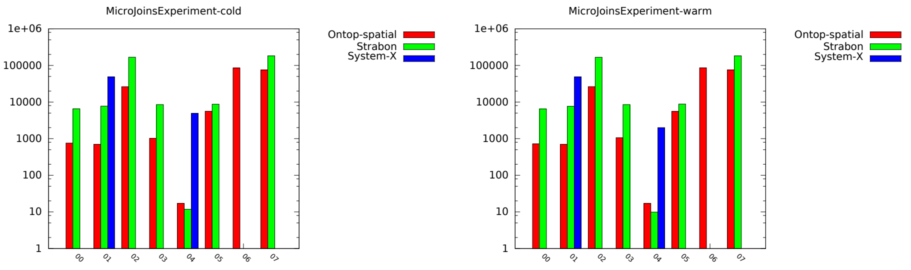

Fig. 2: Spatial Joins experiment (cold and warm cache)

The results of our experimental evaluation can be seen in Figures 1 - 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 and System-X, 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. System-X times out after 40 minutes in spatial join queries 0,1,2,3,6 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. System-X performs worse than Ontop-spatial and Strabon in both cases.

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. It is also evident from the experiments that forwarding geospatial query processing to a spatially-enabled DBMS as back-end can improve performance significantly, as they incorporate well-established optimization techniques in the area of geospatial query processing that have not yet been incorporated in native geospatial triple stores up to date.

3 Temporal Ontology-based Data Access

The problem of posing temporal SPARQL queries on top of temporal databases on-the-fly using the OBDA paradigm is even more challenging. First, there is no standard temporal extension of the framework of RDF and SPARQL, as in the case of the OGC standard GeoSPARQL. Second, in order to support valid time in the data model RDF, none of the proposed approches for translating temporally-enhanced queries into standard SPARQL queries suits the OBDA paradigm. Another feature that the OBDA systems lack is the support for SPARQL1.1. extension functions, which is needed for the support of the temporal extension functions defined in stSPARQL that are documented in [6].

Due to these challenges, we will not consider valid time for the rest of this chapter and we will focus in addressing these issues for supporting user-defined time. We propose an extension of the data model stRDF and the query language stSPARQL with additional, but *lightweight* temporal features that can be adopted by both RDF stores and OBDA systems.

We define the following temporal components in the query language stSPARQL.
Temporal predicates. We extend stSPARQL by defining a set of *temporal predicates* that are based on Allen's interval algebra [1]. These predicates are the following: `strdf:peridEquals`, `strdf:after`, `before`, `periodOverlaps`, `starts`, `finishes`, `periodContains`, `strdf:meets`, `during`, and `isMetBy`. The temporal predicates are equivalent to the temporal extension functions that are defined in stSPARQL and are described in [6]. These functions can either operate on intervals or time instants, where suits the case. For example, an interval can either contain another time interval (e.g., a literal of the `strdf:period` datatype or a literal of the `xsd:dateTime` datatype).
Temporal query rewrite component. The temporal query rewrite component of stSPARQL defines a set of rules for translating *qualitative* temporal queries, i.e., queries with temporal predicates, into *quantitative* ones, i.e., queries with temporal operators. This component of stSPARQL is similar to the query rewrite component of GeoSPARQL [13]. We denote as $R_{temporal}$ the set of rewriting rules R_i that we define for each temporal predicate i. Using these rules, a query q that contains a temporal predicate i will get transformed into the equivalent query q, by applying rule R_i to q. The query q' contains the temporal

extension function of stSPARQL that corresponds to the temporal operator i. In the following section we provide an example of how this new query rewriting component of stSPARQL participates in the overall evaluation of temporal stSPARQL queries. This component of stSPARQL is important, as it allows any OBDA system that implements it to answer temporal SPARQL queries transparently, without modifying their syntax.
Rewriting rules. We now explain how queries that contain temporal predicates (qualitative) get translated into queries that contain functions. We define a rule for each temporal predicate that we define, that corresponds to a temporal function in stSPARQL.

We now describe briedly the workflow of answering stSPARQL queries in an OBDA system like Ontop [24]. Once a temporal stSPARQL query q that contains the temporal predicate i arrives, it gets processed as follows:

- First, gets parsed as standard SPARQL query
- Second, the query gets translated into datalog, and so as the mappings. To achieve this, we have defined a set of temporal datalog predicates that correspond to the temporal predicates and temporal operators defined in stSPARQL.
- In the next step, the datalog representation of the query q gets translated into the datalog representation of query q', by applying rule R_i to q. The datalog representation of the query q' now contains the stSPARQL temporal operator that corresponds to the predicate i.
- Then, the datalog program that is created in the previous step, that contains the transformed query q' and the mappings (as described in [24], gets translated into SQL. To achieve this, we have map each datalog temporal predicate to the corresponding temporal SQL operator.

3.1 Querying the Web Using Ontologies and Mappings

We now describe our framework for querying data sources available on the Web using SPARQL. Our main goal is to design a framework for posing SPARQL queries on various data sources on-the-fly, while staying compatible with well-established Web standards. For this reason, we chose not to extend the SPARQL query language or a mapping language so that querying Web data on-the-fly is transparent to the users: they can write SPARQL queries caring only about *what* they want to retrieve, rather than *where* the desired data is stored or *how*, i.e., in what format. We encapsulate this knowledge by extending the query language SQL with user-defined operators.

The core concept of our approach is to model a data source as a virtual relational table. For this reason, we define a virtual table operator for each kind of data source. Each *virtual table operator* has the syntax: $\mathbf{VT} ::= \mathbf{vtable}(\mathbf{args}[][, \mathbf{f}])$, where the vector *args* denotes the arguments that are given as input to the virtual table operator, while f is optional, denoting the *cache update rate*.

The cache feature is useful in cases where: (i) not all data sources get updated with the same frequency, (ii) some data sources might not be accessible at the next query time (e.g., due to API limitations), or (iii) a minimal query execution time is required, due to a large number of queries, i.e., the frequency of queries is much higher than the update frequency of data sources. To support these cases, f indicates the length of the time window (in milliseconds), during which the retrieved data are temporarily stored. If the virtual table operator with the *same* input parameters ($args$) is invoked twice (or more) before this time window ends, the cached data will be used, improving query time. If the query is repeated after the end of the time window, the fresh data is fetched from the data source and gets stored in the system. If f has a negative value, nothing is stored and the virtual table operator fetches fresh data every time it is invoked. To support this functionality, we store meta-data that contain information about when and where data resulting from a virtual table signature was stored last time.

The result of a virtual table operator is a virtual table with the following schema: $\mathbf{VT}[\mathbf{tupleID}, \mathbf{cols}]$, where $tupleID$ is the unique identifier of a tuple and $cols$ are the requested attributes.

We now describe the implementation of the methodology described above. The system architecture consists of the following components:

• As back-end, it uses the MadIS[6] [12] system, an extensible relational database system built on top of the SQLite[7] database, with extensions implemented in Python via the SQLite wrapper APSW[8]. The SQLite database can be extended with user-defined operators that can be used as row, aggregate, or virtual table operators. The APSW SQLite wrapper provides an interface for implementing these operators in an extensible way through Python. Using MadIS, we define our own operators to create virtual tables and populate them with data that we retrieve from the Web. To query them, we use MadQL, the MadIS implementation of the extended-SQL language we described above, which contains the virtual table operators. We implemented a MadIS virtual table operator for each of the data sources we support (i.e., Twitter, Foursquare, webtables).

• Third party applications are external micro-services that could be invoked by a virtual table operator in MadIS. For example, in the Twitter use case, the `twitterapi` virtual table operator communicates with a Sentiment Analysis classifier to identify the sentiment of each tweet. In this way, we suggest an architecture for performing data analysis tasks that eliminates compatibility issues between the virtual table operator and any data analysis software: the server can be written in any language or platform, but the client can still use it as a service.

• The system Ontop[9] [11], a state-of-the-art, open-source OBDA system that supports both R2RML and the OBDA mapping language. Most specifically, we extended its geospatial extension named Ontop-spatial [4,8,7] in order to have

[6] `http://madgik.github.io/madis`
[7] `http://www.sqlite.org`
[8] `https://github.com/rogerbinns/apsw`
[9] `https://github.com/ontop/ontop`

geospatial support. To this end, we extended the MadIS JDBC connector so that it complies with Ontop, while Ontop was extended to use MadIS as a back-end. The latter modification is the most significant one, enabling Ontop to operate in a "database-agnostic" manner that supports non-materialized databases and relies on MadIS as back-end. The reason is that Ontop, like all other OBDA systems, originally connects only with populated and materialized databases, using their data for optimization, *before* a query is actually fired. Instead, our framework retrieves data only *after* a query is fired, creating a virtual table on-the-fly. As a result, no prior knowledge of the data can be used.

We conducted a thorough experimental study to measure the functionality and performance of our approach in comparison with the state-of-the-art related work [23]. The results showed that our approach achieves better performance and is more rich in funtionality than the state-of-the-art.

A more detailed documentation of this work can be found in [5].

4 Conclusions

In the context of this PhD thesis we describe techniques for efficient integration and querying of geospatial and temporal data. We focus in ontology-based data access techniques for creating virtual semantic graphs on top of relational geospatial and temporal databases, avoiding the conversion and materialisation of original data into RDF, using ontologies and mappings. We introduce the first geospatial OBDA system and we demonstrate its efficiency, comparing its performance with state-of-the-art RDF stores. Then, we introduce new temporal features to the temporal dimension of the data model stRDF and the query language stSPARQL, in order to facilitate the support of temporal SPARQL queries in OBDA systems.

The next step was to go beyond relational databases as data sources by extending the OBDA paradigm with the capability to create virtual RDF graphs on top of data that can be accessed via Web APIs, HTML tables, etc. We propose an architecture of a system that implements these techniques and we showcase its functionality using real-world scenarios. We conduct an experimental evaluation of the system and we compare our approach with a related approach offering similar functionality. The outcome of the evaluation proves that our system is more rich in functionality and also more efficient. Last but not least, we present real-world applications in which the approaches described in this thesis were used.

One possible direction for future work could to support distributed GeoSPARQL processing . A solution into this direction would be to use a distributed system with geospatial support as back-end, such as SpatialHadoop[10], Hive, GeoSpark[11], etc.

Another extension of the work described in this dissertation could be the further development of the raster support, extending the approaches that we

[10] `http://spatialhadoop.cs.umn.edu/`

[11] `https://datasystemslab.github.io/GeoSpark/`

proposed in this dissertation with more functionalities for raster data management, both in terms of representation and querying. For example, GeoSPARQL could be extended with capabilities based on the ones offered in systems described in [14,26,2].

References

1. James F. Allen. Maintaining knowledge about temporal intervals. *CACM*, 26(11), 1983.
2. Peter Baumann, Andreas Dehmel, Paula Furtado, Roland Ritsch, and Norbert Widmann. The multidimensional database system rasdaman. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA.*, pages 575–577, 1998.
3. Konstantina Bereta, Hervé Caumont, Ulrike Daniels, Erwin Goor, Manolis Koubarakis, Despina-Athanasia Pantazi, George Stamoulis, Sam Ubels, Valentijn Venus, and Firman Wahyudi. The copernicus app lab project: Easy access to copernicus data. In *Advances in Database Technology - 22nd International Conference on Extending Database Technology, EDBT 2019, Lisbon, Portugal, March 26-29, 2019*, pages 501–511, 2019.
4. Konstantina Bereta and Manolis Koubarakis. Ontop of Geospatial Databases. In *Proceedings of the 15th International Semantic Web Conference*, 2016.
5. Konstantina Bereta, George Papadakis, and Manolis Koubarakis. Sparqling-up the web on-the-fly using ontologies and mappings. In *Proceedings of the 31st International Workshop on Description Logics co-located with 16th International Conference on Principles of Knowledge Representation and Reasoning (KR 2018), Tempe, Arizona, US, October 27th - to - 29th, 2018*, 2018.
6. Konstantina Bereta, Panayiotis Smeros, and Manolis Koubarakis. Representation and Querying of Valid Time of Triples in Linked Geospatial Data. In *Extended Semantic Web Conference 2013*, volume 7882, pages 259–274. Springer Berlin Heidelberg, 2013.
7. Konstantina Bereta, Guohui Xiao, and Manolis Koubarakis. Ontop-spatial: Ontop of geospatial databases. *J. Web Semant.*, 58, 2019.
8. Konstantina Bereta, Guohui Xiao, Manolis Koubarakis, Martina Hodrius, Conrad Bielski, and Gunter Zeug. Ontop-spatial: Geospatial data integration using GeoSPARQL-to-SQL translation. In *Proceedings of the ISWC 2016 Posters & Demonstrations Track. Co-located with the 15th International Semantic Web Conference (ISWC 2016)*, volume 1690 of *CEUR Electronic Workshop Proceedings*, 2016.
9. Stefan Bruggemann, Konstantina Bereta, Guohui Xiao, and Manolis Koubarakis. *Ontology-Based Data Access for Maritime Security*, pages 741–757. Springer International Publishing, 2016.
10. Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. Ontop: Answering SPARQL queries over relational databases. *Semantic Web Journal*, 8(3):471–487, 2017.
11. Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. Ontop: Answering SPARQL queries over relational databases. *Semantic Web*, 8(3):471–487, 2017.

12. Yannis Chronis, Yannis Foufoulas, Vaggelis Nikolopoulos, and et al. A Relational Approach to Complex Dataflows. In *EDBT/ICDT Workshops*, 2016.
13. Open Geospatial Consortium. OGC GeoSPARQL - A geographic query language for RDF data. OGC Candidate Implementation Standard, 2012.
14. Philippe Cudré-Mauroux, Hideaki Kimura, Kian-Tat Lim, Jennie Rogers, Roman Simakov, Emad Soroush, Pavel Velikhov, Daniel L. Wang, Magdalena Balazinska, Jacek Becla, David J. DeWitt, Bobbi Heath, David Maier, Samuel Madden, Jignesh M. Patel, Michael Stonebraker, and Stanley B. Zdonik. A demonstration of scidb: A science-oriented DBMS. *PVLDB*, 2(2):1534–1537, 2009.
15. Souripriya Das, Seema Sundara, and Richard Cyganiak. R2rml: Rdb to rdf mapping language, 2012. W3C Rec.
16. George Garbis, Kostis Kyzirakos, and Manolis Koubarakis. Geographica: A Benchmark for Geospatial RDF stores (long version). volume 8219 of *Lecture Notes in Computer Science*, pages 343–359. Springer, 2013.
17. Steven Harris and Andy Seaborne. SPARQL 1.1 Query Language. W3C recommendation, March 2013.
18. John R. Herring. OpenGIS implementation specification for geographic information - simple feature access - part 2: SQL option. OpenGIS Implementation Standard 06-104r4, Open Geospatial Consortium Inc., 2010.
19. Roman Kontchakov, Martin Rezk, Mariano Rodriguez-Muro, Guohui Xiao, and Michael Zakharyaschev. Answering SPARQL queries over databases under OWL 2 QL entailment regime. In *Proc. of International Semantic Web Conference (ISWC 2014)*, Lecture Notes in Computer Science. Springer, 2014. (Accepted).
20. Manolis Koubarakis and Kostis Kyzirakos. Modeling and Querying Metadata in the Semantic Sensor Web: The Model stRDF and the Query Language stSPARQL. In Lora Aroyo and et al., editors, *ESWC*, volume 6088 of *LNCS*, pages 425–439. Springer, 2010.
21. Kostis Kyzirakos, Manos Karpathiotakis, and Manolis Koubarakis. Strabon: A Semantic Geospatial DBMS. In Philippe Cudré-Mauroux and et al., editors, *ISWC*, volume 7649 of *LNCS*, pages 295–311. Springer, 2012.
22. Frank Manola and Eric Mille. RDF primer. W3C Recommendation, World Wide Web Consortium, February 2004. Available at `http://www.w3.org/TR/rdf-primer-20040210/`.
23. Matthieu Mosser, Fernando Pieressa, Juan L. Reutter, Adrián Soto, and Domagoj Vrgoc. Querying apis with SPARQL: language and worst-case optimal algorithms. In *ESWC*, pages 639–654, 2018.
24. Mariano Rodríguez-Muro and Martin Rezk. Efficient SPARQL-to-SQL with R2RML mappings. *Web Semantics: Science, Services and Agents on the World Wide Web*, 33(1), 2015.
25. Guohui Xiao, Diego Calvanese, Roman Kontchakov, Domenico Lembo, Antonella Poggi, Riccardo Rosati, and Michael Zakharyaschev. Ontology-based data access: A survey. In *IJCAI-ECAI-18 – July 13-19 2018, Stockholm, Sweden*, 2018.
26. Ying Zhang, Martin L. Kersten, and Stefan Manegold. Sciql: array data processing inside an RDBMS. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 1049–1052, 2013.