Semantic Querying of Integrated Raster and Relational Data: A Virtual Knowledge Graph Approach

Arka Ghosh[1,*], Mantas Šimkus[1] and Diego Calvanese[1,2]

[1]Department of Computing Science, Umeå Universitet, Umeå, Sweden
[2]Research Centre for Knowledge and Data, Faculty of Engineering, Free University of Bozen-Bolzano, Bolzano, Italy

Abstract
Ontology-based data access (OBDA) facilitates access to heterogeneous data sources through the mediation of an ontology (e.g. OWL), which captures the domain of interest and is connected to data sources through a declarative mapping. In our study, large, heterogeneous earth observational (EO) data, known as *raster data*, and geometrical data, known as *vector data*, are considered as (heterogeneous) data sources. Raster data represent, e.g., Earth's natural phenomena, such as surface temperature, elevation, or air pollution, as multidimensional arrays. In contrast, vector data depict, e.g., locations, networks, or regions on Earth, using geometries. Domain experts, such as earth scientists and GIS practitioners, still struggle to undertake advanced studies by querying large raster and vector data in an integrated way because, unlike relational data, they come in diverse formats and different data structures. In our approach to integration, we use a geospatial extension of an RDBMS to represent vector data as relational data, and a domain-agnostic array DBMS to handle raster data. Our aim is to extend the OBDA paradigm to effectively deal with relational, vector, and raster data in a combined way, while leveraging the built-in capabilities of data management tools relevant to each type of data. We also plan to develop techniques to calculate on the fly for each user query posed over the ontology an optimal query plan that exploits, at best, the query processing capabilities of each tool, while limiting costly data transfer operations between tools.

Keywords
Ontology-Based Data Access (OBDA), Knowledge Representation, Spatial-temporal reasoning, Relational Data, Vector Data, Raster Data, Multi-Dimensional Arrays, Artificial Intelligence (AI), Virtual Knowledge Graph (VKG)

1. Introduction

In the age of Big Data, different types of data are being generated at a rate that is beyond human comprehension, and this makes effective data management challenging. In particular, data heterogeneity is becoming an issue that needs to be addressed via suitable data integration techniques. In the study, we are proposing, we rely on the paradigm of *Virtual Knowledge Graphs* (VKGs), which allows for flexible and efficient management of large amounts of richly

RuleML+RR'23: 17th International Rule Challenge and 7th Doctoral Consortium, September 18–20, 2023, Oslo, Norway
*Corresponding author.
arka.ghosh@umu.se (A. Ghosh); mantas.simkus@umu.se (M. Šimkus); diego.calvanese@unibz.it (D. Calvanese)
https://www.umu.se/en/staff/arka-ghosh/ (A. Ghosh)
0000-0003-3789-0900 (A. Ghosh); 0000-0003-0632-0294 (M. Šimkus); 0000-0001-5174-9693 (D. Calvanese)

structured data. The VKG paradigm, also known as *Ontology-Based Data Access* (OBDA) [1, 2, 3], has emerged as a proposal to simplify access to relational data for end-users by letting them formulate high-level queries over a conceptual representation of the domain of interest, provided in terms of an ontology [2, 1].

In traditional OBDA, domain knowledge is represented at the conceptual level as an *ontology* $\mathcal{O}$, e.g., expressed in the lightweight ontology language OWL 2 QL [4], while actual data are maintained in a relational *data source* $\mathcal{D}$, but are *not* materialised at the conceptual level (which justifies the term "virtual"). To establish the relationship between the ontology $\mathcal{O}$ and the data $\mathcal{D}$ at the source, OBDA relies on a declarative specification, provided in terms of a set $\mathcal{M}$ of *mapping assertions.* Each such mapping assertion $Q(\vec{x}) \rightsquigarrow E(\vec{f}(\vec{x}))$ specifies how the data retrieved from $\mathcal{D}$ by means of a SQL query Q should be used to populate a class or property E of the ontology. The atom $E(\vec{f}(\vec{x}))$ in the right-hand-side of the mapping assertion refers to the answer variables $\vec{x}$ of the query Q, and might make use of so-called *iri-templates* $\vec{f}$. An iri-template f is a function that constructs, from the values instantiating $\vec{x}$ in answer to Q, an ontology literal or an IRI identifying an ontology object. Thus, by applying the mapping $\mathcal{M}$ to the source database $\mathcal{D}$, one obtains a knowledge graph $\mathcal{M}(\mathcal{D})$, and user queries formulated in the standard Semantic Web query language SPARQL [5] over the ontology, are answered over this knowledge graph (KG). Concretely, in an OBDA system, the KG stays virtual and, exploiting both the ontology $\mathcal{O}$ and the mapping $\mathcal{M}$, the OBDA system translates the SPARQL query into a SQL query expressed over $\mathcal{D}$, which is then directly evaluated by the underlying (open-source or commercial) relational database management system (RDBMS), such as PostgreSQL, storing $\mathcal{D}$.

1.1. Geospatial data: vector and raster data

The starting point of our study has been the OBDA/VKG framework, as described above, where data is stored in a plain relational database. However, different data types often have specific, predefined semantics that must be considered explicitly. The data might come in specific formats, comply with specific models, and be equipped with specific types of operations affected by the special semantics. Such operations must also be supported by the underlying system that manages the data storage. A notable example is *geospatial data*, which we are considering in our work. Many real-world applications such as climate change [6], wildfire risk assessment [7], crop yield mapping [8], etc., require to combine such data with different heterogeneous data sources to facilitate complex analysis. Thus it is essential to be able to reason about these geospatial entities and eventually answer queries posed by end users to guide decision-making processes [9]. Geospatial data are represented as (multi-dimensional) raster data and as vector data, depending on the particular characteristics of the type of information [10].

Vector data comprises geometries such as points, lines, and polygons, and collections of these elements with their positional parameters (e.g., longitudes and latitudes) that characterise them on the earth's surface. Examples of vector data are static locations (points), roads and river networks (lines), and boundaries of countries, provinces, municipalities, lakes, and islands (polygons). So, vector data (consisting of geometry and attribute data) can be connected through specific extensions such as PostGIS, Oracle Spatial, etc., to relational database management systems like PostgreSQL, OracleDB, etc. Then, one can execute queries on these vector data using the relational query language SQL, suitably extended with specific geospatial functions.

In contrast, geospatial *raster data* are defined as multi-dimensional arrays (also known as gridded data, or datacubes) [11], where each cell of the arrays represents a value associated with a natural phenomenon (like surface temperature, soil moisture, elevation, vegetation indices, or air pollutants), combined with location information (e.g., longitude and latitude). The cells cover a portion of the Earth's surface, and the raster's spatial resolution determines each cell's size. A higher spatial resolution necessitates a larger number of cells per unit of area, hence larger-sized raster data. The temporal resolution of raster data is related to the time necessary to capture one single raster image, e.g., by a satellite's sensors. When working with geospatial raster data, one must also consider coordinate reference systems, map projections, transformations, raster extent, etc., since these aspects influence the raster data semantics.

1.2. Integrating relational, vector, and raster data

Generally, traditional relational databases support raster data arrays using the spatial extension, but not every format is supported. Additionally, the large arrays that represent raster data need to undergo a conversion from their native format (such as NetCDF[1], GeoTIFF[2], GeoPackage[3], or HDF[4]) to a suitable relational form that can be queried using SQL. This conversion leads to a significant increase in the size of the data. Moreover, raster data sets are often growing fast in size and quantity, producing terabytes of data daily, due to improvements in remote sensing and instrumentation (both air-borne and space-borne) [12]. It has been estimated that the archived amount of raster data will soon reach the zettabyte scale [13]. Storing massive amounts of ever-growing raster data together with relational data in a RDBMS and executing queries over them will not be efficient and scalable.

For the above reasons, we have decided to use *rasdaman*[5] [14, 15](for "raster data manager"), a domain-independent array-based DBMS, as the container for raster data in this study. Rasdaman provides flexible, scalable storage management and extensive array algebra to manipulate enormous multi-dimensional raster arrays, along with a query language known as *rasql*[6] to query over the stored raster data. Rasdaman is domain-independent, hence suitable for all applications where raster data management is a concern. For this reason, however, it does not comply natively with the coordinate reference system (CRS) of stored geospatial raster data. It has a Petascope component that adds geo semantics, such as support for the OGC standard interfaces WCS[7], WCPS[8], WCS-T[9], and WMS[10]. Currently, Petascope supports grid topologies whose axes align with the stored raster's CRS axes. But it does not support vector data, which we consider essential in our setting.

[1]https://www.unidata.ucar.edu/software/netcdf/
[2]https://www.ogc.org/standard/geotiff/
[3]https://www.geopackage.org/
[4]https://www.hdfgroup.org/solutions/hdf5/
[5]http://www.rasdaman.org/
[6]https://doc.rasdaman.org/04_ql-guide.html
[7]https://www.ogc.org/standard/wcs/
[8]https://www.ogc.org/standard/wcps/
[9]http://www.opengis.net/doc/IS/wcs-t/2.0
[10]https://www.ogc.org/standard/wms/

2. Research challenges

The OBDA/VKG approach can facilitate the integration of these diverse geospatial datasets provided by relational and array-based DBMSs by making use of mapping assertions that populate the defined geo-ontologies using both relational and raster data. In addition, it is of interest to include in the integration architecture also open SPARQL endpoints of public KGs (e.g., DBPedia, LinkedGeoData, and Wikidata). Overall, this will enable the creation of a cohesive KG that can be queried and analysed holistically. However, the challenge of virtually integrating raster data stored in an array-based DBMS with vector and relational data, possibly including also public KGs, has not been addressed before. This leads to our first research question:

RQ$_1$: *Which is an effective architecture for the integration of different data formats relevant to geospatial data, in particular, raster data, geometric vector data, general relational data, and open KGs, using the VKG paradigm?*

The proposed architecture will be evaluated by considering different concrete industrial standards (e.g., CityGML[11] for 3D city data), by developing ontologies for them and by linking those ontologies to established relational storage formats for the relevant data.

Next, we want to process spatial-temporal queries over these integrated raster-vector data efficiently. To efficiently execute a user query (or, more precisely, the query obtained through the initial ontology-rewriting step performed by a VKG system), we have to exploit the capabilities of both relational and array-based DBMSs and devise methods to compute for a given user query an efficient query plan in such a heterogeneous setting. This requires understanding how the query can be broken down into sub-queries that get executed by the individual systems and how the obtained intermediate results can be combined to form the final result that the user asked for. The challenge lies in doing so while maximising the computation that can be delegated to each DBMS while simultaneously minimising costly data transfer operations and the data that have to be materialised at the level of the integration system. This leads to the second research question, which is actually tightly connected to the first one:

RQ$_2$: *How can we devise effective query plans that guide the structuring and efficient processing of spatial queries over integrated data?*

To address this question, techniques for efficient query-answering over integrated geodata sources and open SPARQL endpoints of public KGs will be developed. This calls for studying how to federate both relational databases and SPARQL endpoints. Good performance will be the critical factor, and to achieve it, special attention must also be paid to the partial materialisation of results of frequent but expensive operations.

The aforementioned research questions give an overview of our ongoing research. In Section 3, we have covered some well-known literature that is related to our field of study. We will show how our virtual approach under the VKG paradigm is different and unique compared to the existing approaches. In Section 4, we discuss the proposed research methodology, integration pipeline of relational and raster data, and the issues arising in this context. Section 5 points out the future steps of our research.

[11]https://www.ogc.org/standard/citygml/

3. Related work

The rapid expansion in geospatial data's diversity, richness, and quantity has created new information demands. In this section we discuss some previous studies on geospatial data from the data and semantic technologies perspectives. Many real-world applications now require geospatial data from multiple sources and formats for complicated analyses and decision-making [16]. Thus, merging these diverse data becomes more challenging when they are very large [17]. Although there is a lot of study on the size, processing, analysis, and heterogeneity of geospatial data, most of it focuses on the vector or raster models independently [17]. These two data models are rarely handled together. Even international standards such as *OGC® WFS* [18] or *OGC® WCS 2.0 Interface Standard - Earth Observation Application Profile* [19] separate both models and do not provide languages, data structures, or algorithms to execute queries using information from both models simultaneously. For instance, one well-known solution for queries that involve both raster and vector datasets is to transform the vector dataset into a raster dataset and then use a raster algorithm to answer the query. This solution applies, e.g., to the zonal statistics operation of Map Algebra over raster and vector data in geographical information system (GIS) software such as ArcGIS [20] and GRASS [21]. Brown et al. [22] describe a data model representing vector and raster data with a data abstraction based on multidimensional arrays in SciDB. This work describes data types, storage structures, and operators for querying vector and raster data, although no implementation details are provided (neither of the data structures nor of the algorithms needed to support the model and the queries).

Brisaboa et al. [23] present a framework to store and manage vector and compressed raster data and an algorithm for query answering. This algorithm returns the elements of the vector dataset overlapping regions of the raster dataset that fulfil a range constraint. However, their solution does not return the exact cells of the raster, fulfilling the range constraint provided by the vector region. In this study, the vector dataset is indexed with an R-tree and the raster dataset is represented and indexed with a compact data structure called k^2*-acc* [24], which needs a separate tree-based data structure for each distinct value in the raster. This way of representing a raster is inefficient as the k^2-acc data structure compresses well when the dataset has few distinct values. However, when the number of distinct values gets large, the represented raster dataset requires substantially more space than its uncompressed representation [25], and most queries scale poorly. To overcome the limitation of the k^2-acc raster indexing, Silva-Coira et al. [17] and Ladra et al. [25] have proposed the k^2*-raster* data structure to represent the raster dataset by compressing and indexing simultaneously. This makes the query evaluation on the raster data faster, provided the algorithm operating on the k^2-raster is well designed.

The most popular way to integrate raster and vector data is by converting one data set into another so that both datasets transform into the same representation. Many existing raster-based systems rasterize (i.e., translate into pixels) the vector data to the exact resolution of the raster data and then process them using a raster-raster join. Similarly, vector-based systems vectorize (i.e., translate into polygons) the raster data by converting each pixel to a point and then joining both data sets using a vector-vector join. These two approaches work well for small and medium-resolution data since the conversion process does not dramatically increase the data size [26]. However, these methods are no longer viable due to the recent accessibility of high-resolution satellite data. In fact, Singla et al. [27] mention that the size of the converted data increases quadratically with the raster image resolution. Singla et al. [26] proposed a Raptor Join operator that utilises a novel index structure called ‘Flash Index‘ that joins raster and vector data in their native formats. Raptor join takes vector data and metadata of raster files as input and outputs pixel ranges that match the resolution of the raster and vector data. This technique is designed as a relational join operator and employs an in-situ approach, making it suitable for ad-hoc queries. The work [26] is a possible approach to our research question RQ_1, but we didn't find any source of implementation so that we can reproduce the results. The second issue is that, given our approach, no sufficient background is given on how well other geospatial functions of geospatial database tools can process the output in subsequent steps.

From the Semantic Web technology perspective, the work [28] describes an OBDA approach that enables queries expressed in the OGC standard language GeoSPARQL [29] to be executed over geospatial relational databases storing vector or raster data, by performing on-the-fly GeoSPARQL-to-SQL translation utilising ontologies and R2RML mappings [30]. Another recent study [31] proposed the *GeoLD* query engine powered by GeoSPARQL [29] for scientific raster array data using the *Coverage to RDF Mapping Language* (C2RML), an extension of the R2RML language. Rasdaman is used to store the raster data, and its WCPS implementation Petascope to access them through the web interface. Andrejev et al. [32] suggests that users should combine array and metadata within single queries to get answers to complex problems with just a single round trip, rather than through iterative communication with several servers employing different models and retrieval paradigms. Our current research aims to integrate metadata and geo-semantics with respective raster arrays during integration with relational data (including vector data) on the fly, to answer user queries, hence relying on virtualisation.

4. Ongoing research

4.1. Initial approach

The OBDA/VKG framework is compatible with relational data sources and its functionalities. So, we planned an experimental scenario with relational vector data for all countries worldwide (e.g., national boundaries) and with raster data for the entire world, over the past ten years, but for only two parameters (such as soil temperature and soil moisture). We stored more than 40 GB of such raster data using ‘raster2pgsql’, a raster loader executable that loads raster data into a table in a SQL-suitable format using the PostGIS extension of PostgreSQL.

Table 1 gives an overview of the data stored in PostgreSQL. Here, we give an idea of the size of the raster data and of the difficulty in dealing with them. For example, the earth observational (EO) satellite captures global soil temperature four times a day (temporal resolution: 3 hrs/day) over ten years (temporal coverage). This means we have four raster images daily over ten years, for a total of 30,000+ raster images. The first issue that we encountered was related to the raster data ingestion inside PostgreSQL using the raster loader ‘raster2pgsql’. It took a massive amount of conversion time (from the native raster format, i.e., GeoTIFF, to the SQL-suitable PostGIS raster format) and then required indexing the data in a PostgreSQL table. This type of raster conversion has also increased the size of the actual raster data (the actual size was 20 GB in the native format, which became more than 50 GB after conversion). So, raster data materialization in a RDBMS is not very practical when raster data are extensive. Additionally,

Table 1
Initially, geospatial datasets are stored only in PostgreSQL

	World Countries	Global Soil Moisture [33]	Global Soil Temp. [33]
Type	vector	raster	raster
Native Format	.shp	GeoTIFF	GeoTIFF
Logical Format	relational	PostGIS raster	PostGIS raster
Spatial Resolution	None	0.25° x 0.25°	0.25° x 0.25°
Temporal Resolution	None	3 hours per day	3 hours per day
Temporal Coverage	None	Jan 2011 to July 2021	Jan 2011 to July 2021
CRS	EPSG:4326 - WGS 84	EPSG:4326 - WGS 84	EPSG:4326 - WGS 84
No. of files	1	30000+	30000+
Feature Rows	652	6452039	6281495
Size	130 MB	20 GB	22 GB

the number of raster data parameters depends on the application scenario, and may be much larger than the two we considered in our experiment. E.g., in [34], the number of parameters is 30, and this would make the ingestion into PostgreSQL even more problematic.

Once both types of data were stored in PostgreSQL, we have executed spatial-temporal queries of different complexity that return information on how soil temperature and moisture vary by country, region, and continent. Some examples are the following:

- **Spatial query**: List the European countries and their respective average soil moisture where the average temperature equals or exceeds 15°C.
- **Spatial-temporal query**: List the countries, their respective continents, and average soil moisture where the average temperature equals or exceeds 15°C over the last five years.

We obtained some results in the form of tables or single numerical values and filtered arrays. However, problems arose when we conducted a complicated spatial-temporal query incorporating both vector data and a vast quantity of raster data to get the results. The main issues are increased memory use and query execution time. We didn't always get results due to the poor query plan, which caused out-of-memory situations. This approach may still work for small raster data but is practically inefficient for ever-growing raster datasets with multiple parameters. So, in our current approach, we have decided to use PostgreSQL to represent relational data (vector data) and rasdaman to represent raster data, and leverage their capabilities to query over the combined geospatial data. This approach will be discussed next.

4.2. Current approach

The main idea is to reformulate a user query into an efficient spatial-temporal query that integrates both raster data and vector data and produces results as a relational table, without actually materializing the data required to process the query. But before query reformulation, we need to combine both relational and raster data efficiently, based on the OBDA framework, as addressed in research question RQ_1. In order to facilitate this, we have devised a pipeline, illustrated by Figure 1, that can connect both databases using a Python wrapper, and retrieve

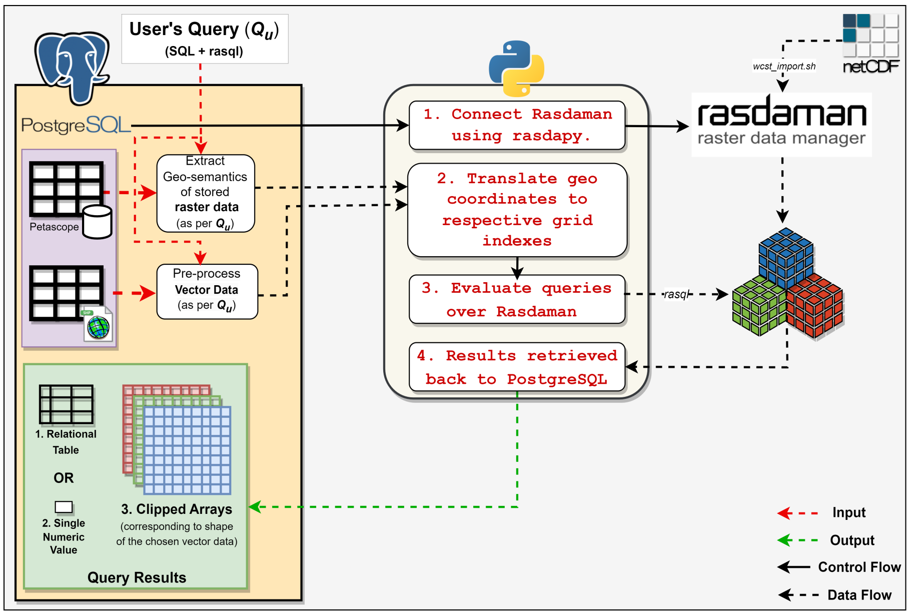

User's Query (Q_u)
(SQL + rasql)
PostgreSQL
Petascope
Extract Geo-semantics of stored raster data (as per Q_u)
Pre-process Vector Data (as per Q_u)
1. Connect Rasdaman using rasdapy.
2. Translate geo coordinates to respective grid indexes
3. Evaluate queries over Rasdaman
4. Results retrieved back to PostgreSQL
wcst_import.sh
netCDF
rasdaman raster data manager
rasql
1. Relational Table
OR
2. Single Numeric Value
3. Clipped Arrays (corresponding to shape of the chosen vector data)
Query Results
Input
Output
Control Flow
Data Flow

Figure 1: Integration pipeline: relational data (vector) + raster data

the final result on the fly based on the user query. This pipeline can use PostgreSQL's query language capabilities (i.e., SQL and PostGIS functions) and rasdaman's array manipulation functionalities (i.e., array algebras, rasql condensers[12]) to pre-process the input data and post-process the output result, if necessary. These pre- and post-processing are necessary to clean the data originating from the real world, to avoid certain issues, which we will discussed later.

We now discuss the pipeline, which we break down into three main parts as follows:

- **Relational data perspective (PostgreSQL)**: The geometries of the respective regions of interest are represented as several distinct pairs of longitudes and latitudes. We use the PostGIS functions '`ST_AsText`' and '`ST_Dump`'[13] to convert the region geometries into a set of OGC Well-Known Text (WKT) representations [35], which is the first input based on the user's query Q_u to the Python wrapper. Petascope, a separate database within PostgreSQL, provides the necessary geo semantics[14] of the respective raster datasets stored inside rasdaman. This geo semantics contain metadata, such as the raster dataset name, CRS, raster extent, array size, and spatial and temporal dimensions, which are required to translate geo coordinates to respective grid indices. Then, these grid indices are sent to rasdaman via the Python wrapper to clip required raster data according to the shape of the region of interest mentioned in the user's query Q_u.
- **Python wrapper**: We have used the 'plpyhton3' procedural language[15], which supports PostgreSQL and PostGIS functions wrapped by Python, to define stored procedures

[12]https://doc.rasdaman.org/04_ql-guide.html#condensers
[13]https://postgis.net/workshops/postgis-intro/geometries.html
[14]https://doc.rasdaman.org/stable/01_introduction.html?highlight=geo%20semantics#features
[15]https://www.postgresql.org/docs/current/plpython.html

(e.g., `connect_rasdaman`, `geo_to_grid_translation`) inside PostgreSQL, to connect with rasdaman using the 'rasdapy' package. rasdapy[16] is a client API for rasdaman that enables building and executing rasql queries with Python. This python wrapper facilitates the following functionalities:

1. Connect PostgreSQL to Rasdaman using rasdapy.
2. Translate geo coordinates to respective grid indexes.
3. Evaluate rasql queries over raster arrays stored inside rasdaman.
4. Retrieve the result to PostgreSQL as a single value or table, or as arrays.

- **Raster data perspective (rasdaman)**: We have used raster data represented in the NetCDF[17] format inside rasdaman through the 'wcst_import.sh' utility[18]. Rasql only works with grid coordinates (or indices) of stored arrays; it does not comply with geo-coordinates (i.e., a sequence of longitude-latitude pairs), so any geometrical shape represented as geo-coordinates fetched from the relational database must be translated into corresponding grid coordinates to incorporate vector data inside rasql to work. After getting the grid indices of the region of interest, we use rasdaman's arithmetic operators (e.g., `+`, `-`, `*`, `>`, and `<`), to pre-process and apply built-in aggregation functions (e.g., average, maximum, and minimum), also known as rasql condensers[19], to filter out the required raster data.

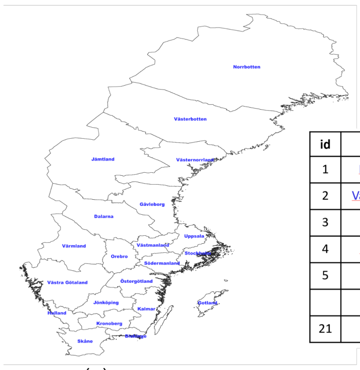
Finally, using this pipeline, we have integrated relational and raster data, and have retrieved the result as a single numerical value, a table, or an array, based on the form of the user query Q_u. These results can be used as a data source to generate a virtual knowledge graph using ontologies via mappings, following the OBDA paradigm, as shown in Figure 3 in Section 5. For the case where an array is returned, we need to address some issues, which are mentioned later in the paper. We demonstrate the application of our pipeline in the next section.

4.3. Case study: Sweden

We use Sweden and its provinces and municipalities as the regions of interest in our study to demonstrate the ongoing progress of the current research and related issues. The issues we mention are not limited to this use case, but apply more in general. We have collected vector data for Sweden's provinces and municipalities from the GADM[20] database [Version 4.1, updated on 16 July 2022], illustrated in Figure 2. The GADM database contains maps of the administrative areas of all countries at all subdivision levels, with high spatial resolutions and a comprehensive set of attributes in different file formats, such as shapefile (.shp), .kmz, and .geojson. Vector data are represented as relational tables in PostgreSQL using the PostGIS extension, as depicted in Figure 2.

As for raster data, we are using data from the NASA Earth Data Explorer, specifically snow data [36] and surface temperature [37], and for the elevation of Sweden, we are using the European Digital Elevation Model [EU-DEM Version 1.1] provided by EU-Copernicus Land

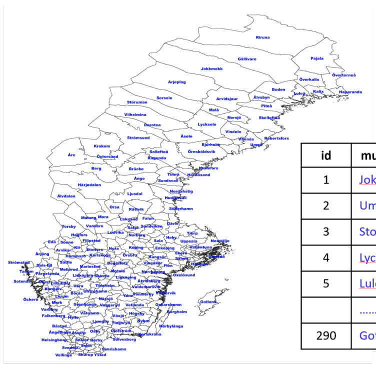
[16]https://pypi.org/project/rasdapy3/
[17]https://www.unidata.ucar.edu/software/netcdf/
[18]https://doc.rasdaman.org/05_geo-services-guide.html?highlight=wcst_import#data-import
[19]https://doc.rasdaman.org/04_ql-guide.html#condensers
[20]https://gadm.org/maps/SWE_1.html



(a)

id	provinces	geometry
1	Norbotten	polygon01
2	Vasterbotten	polygon02
3	Dalarna	polygon03
4	Gävleborg	polygon04
5	Örebro	polygon05
	....	....
21	Skåne	polygon21

(b)



(c)

id	municipalities	geometry
1	Jokkmokk	m_polygon01
2	Umeå	m_polygon02
3	Stockholm	m_polygon03
4	Lycksele	m_polygon04
5	Luleå	m_polygon05
	....	....
290	Gothenburg	m_polygon290

(d)

Figure 2: Relational data depicting geometric polygons of 21 provinces and 290 municipalities of Sweden

Monitoring Service 2023, European Environment Agency (EEA)[21]. We have stored these large data arrays in rasdaman. Table 2 shows the details of all the datasets that we have used for Sweden.

With these datasets stored in two databases, PostgreSQL and rasdaman, we wish to combine them to execute some user-defined spatial-temporal queries concerning Sweden. In the next subsection, we demonstrate some example queries that we tested using the pipeline we have devised, along with the related issues that came up. These issues are not specific to our use case, but apply in general when processing spatial-temporal queries.

4.3.1. First results

Consider the following example query:

> Query Q_1: *What are the average, maximum, and minimum surface temperatures in °C of the municipalities Dorotea and Linköping of Sweden?*

[21]https://land.copernicus.eu/imagery-in-situ/eu-dem/eu-dem-v1.1?tab=metadata

Table 2
Datasets used for Sweden

Attributes	Provinces	Municipalities	Snow Cover [36]	Surface Temp [37]	Elevation
Type	vector	vector	raster	raster	raster
Native Format	.shp	.shp	NetCDF	NetCDF	GeoTIFF
Logical Format	relational	relational	arrays	arrays	arrays
Spatial Resolution	None	None	500m x 500m	500m x 500m	25m x 25m
Temporal Resolution	None	None	daily	daily	None
Temporal Coverage	None	None	Apr 22 to Apr 23	Apr 22 to Apr 23	2011
CRS	EPSG:4326	EPSG:4326	EPSG:4326	EPSG:4326	EPSG:3035
Size	6 MB	8 MB	338 MB	246 MB	7 GB
Features	21	290	3,992,564,680	1,001,057,824	1,896,048,396

Table 3
Results of Q_1

municipalities	avgtemp_°C	maxtemp_°C	mintemp_°C
Dorotea	8.75790963603404	13.050000000000011	0.010000000000047748
Linköping	10.882990371389285	17.53000000000003	2.670000000000016

Q_1 is a simple spatial query that integrates vector data (such as 'geom') and raster data (such as 'Surface_Temperature_Sweden') based on the name of the municipality (e.g., `Dorotea`) retrieved from the vector data. This query also uses pre-processing (e.g., '`*0.02`') and post-processing (e.g., '`-273.15`' for conversion from degrees Kelvin to degrees Celsius) based on the geo semantics from petascope. Query Q_1 is built using the capabilities of SQL (e.g., `ST_AsText`) and rasql condensers (e.g., `avg_cells`, `max_cells`, `min_cells`), wrapped by user-defined Python functions (e.g., `geo_index2grid_index`, `aggregated_result_numeric`). The result of Q_1 is retrieved as a relational table, as shown in Table 3. Then, we use this result and define mappings to populate geo ontologies to create the respective knowledge graph.

In the described pipeline, we have observed the following issues:

1. *Multipolygons to polygon:* The municipalities Dorotea and Linköping are landlocked regions of Sweden and thus have only one polygon. But municipalities like Stockholm, Göteborg, Umeå, etc., are coastal regions and thus have many scattered island features, e.g., archipelago. These features are represented as a list of several distinct polygons (called *multi-polygons*) under the same name (e.g., Umeå has 40 unique regions). Our pipeline has to consider this during query processing.
2. *Pre-processing and post-processing:* Sometimes, the vector and raster data must be cleaned to make them suitable for the pipeline and the desired output.
3. *Metadata retrieval on-the-go:* Retrieval of geo semantics or metadata from the petascope table should be automatic, based on the chosen raster dataset mentioned in the user query, to translate geo-coordinates to grid indices effectively.
4. *Process filtered arrays:* The result retrieved as filtered arrays based on the user query is in the string format, and for this, there is limited or no scope for optimization for further processing, analysis and visualization.

The following are a few of many example queries designed for the use case, which we only partially resolved utilising the pipeline above, due to the difficulties we have described.

- Query Q_2: List those capitals of respective provinces of Sweden whose spatial average elevation is above 10 meters.
- Query Q_3: List those municipalities of Sweden and their respective maximum snow coverage, where the average surface temperature is less than -15°C.
- Query Q_4: What is the average surface temperature of Sweden's municipalities with neighbouring municipalities with an average elevation above 10 meters?

In order to automate and expand the proposed pipeline, so that it can be seamlessly applied to answer queries by combining arbitrary relational data (in PostgreSQL) and raster data (in rasdaman), we need to address the above mentioned issues.

5. Next steps

Finding suitable solutions to the issues mentioned in the previous section will set the basis for extending the OBDA/VKG framework so as to deal with combined vector and raster data. We illustrate below the problems that we have identified in this context, and that we intend to address in our followup research.

5.1. Defining mappings and ontologies

We are planning to use C2RML, an extension of the original R2RML mapping language for raster coverage, as described in [31]. As for the ontologies, we can rely on existing ontologies for the geospatial domain, or, when needed, develop tailored ontologies expressed in OWL 2 QL [4], the standard profile of the OWL 2 language based on the *DL-Lite* family of Description Logics [38].

5.2. Query planning and optimisation

To compute query plans that optimise query execution, we plan to make use of techniques that rely on *Magic Set Transformation* (MST) [39] and *Answer Set Programming* (ASP) [40]. MST is a technique used in rule-based query optimisation in deductive databases, i.e., systems that use rules to derive new information from existing facts. MST transforms a query into an equivalent form that produces the same result but is more efficient to evaluate. ASP is a declarative programming paradigm for knowledge representation and reasoning. It allows one to declare a set of rules and restrictions and then locate all feasible response sets that satisfy those rules and constraints. ASP can be used in query planning and optimisation by writing inquiries as logic programs and using an ASP solver to find answer sets. By encoding queries and optimisation constraints as ASP programs, one can take advantage of ASP's sophisticated reasoning capabilities to compute optimal query plans. We will also introduce a suitable cost function to evaluate the execution cost of spatial-temporal queries in terms of memory usage and execution time.

5.3. Extending the Ontop system to support spatial-temporal queries

To the best of our knowledge, the work on *Ontop-Spatial* [28] is the only one that has extended the OBDA/VKG paradigm to work over geospatial vector and raster data by relying on a relational database system (namely, PostgreSQL). It builds on the Ontop VKG system [41, 42] and exploits its query rewriting features. In our approach, we also intend to extend the VKG system Ontop, to work over geospatial data by transforming vector data in the relational format. However, the key distinguishing feature of our proposal is that we will keep raster data in its native form (i.e., as multi-dimensional array data), and we will rely on an array-based DBMS (like rasdaman) and its specific query processing capabilities to manage and compute queries over the raster data. The resulting data consisting of relational tables and (clipped) multidimensional arrays is then considered as further input data for the VKG system. The extended VKG framework that we propose with the additional inputs to be considered, is illustrated in Figure 3.

We are collaborating with the Technical University of Munich (TUM), Germany, in the proposed research.

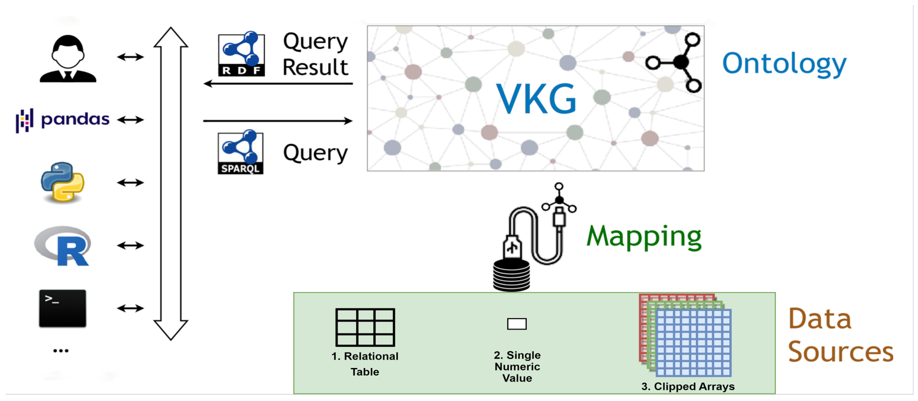


Figure 3: Extended VKG framework: the established OBDA/VKG framework will be extended to process the results (including multidimensional arrays) retrieved from the above pipeline.

Acknowledgments

This work was supported by the Wallenberg AI, Autonomous Systems, and Software Program (WASP), funded by the Knut and Alice Wallenberg Foundation, and by the Province of Bolzano and DFG through project D2G2 (DFG grant no. 500249124). Šimkus was supported by the Austrian Science Fund (FWF) project P30873.

References

[1] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, R. Rosati, Linking data to ontologies, J. on Data Semantics 10 (2008). doi:10.1007/978-3-540-77688-8_5.

[2] G. Xiao, D. Calvanese, R. Kontchakov, D. Lembo, A. Poggi, R. Rosati, M. 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. doi:10.24963/ijcai.2018/777.

[3] G. Xiao, L. Ding, B. Cogrel, D. Calvanese, Virtual Knowledge Graphs: An overview of systems and use cases, Data Intelligence 1 (2019) 201–223. doi:10.1162/dint_a_00011.

[4] B. Motik, B. Cuenca Grau, I. Horrocks, Z. Wu, A. Fokoue, C. Lutz, OWL 2 Web Ontology Language Profiles (Second Edition), W3C Recommendation, World Wide Web Consortium, 2012. Available at http://www.w3.org/TR/owl2-profiles/.

[5] S. Harris, A. Seaborne, SPARQL 1.1 Query Language, W3C Recommendation, World Wide Web Consortium, 2013. Available at http://www.w3.org/TR/sparql11-query.

[6] S. Savi, A. Buter, T. Heckmann, J. Theule, L. Mao, F. Comiti, Multi-temporal analysis of morphological changes in an alpine proglacial area and their effect on sediment transfer, Catena 220 (2023). doi:10.1016/j.catena.2022.106701.

[7] M. B. Joseph, M. W. Rossi, N. P. Mietkiewicz, A. L. Mahood, M. E. Cattau, L. A. St. Denis, R. C. Nagy, V. Iglesias, J. T. Abatzoglou, J. K. Balch, Spatiotemporal prediction of wildfire size extremes with bayesian finite sample maxima, Ecological Applications 29 (2019).

[8] B. Maestrini, B. Basso, Predicting spatial patterns of within-field crop yield variability, Field Crops Research 219 (2018) 106–112.

[9] M. Alirezaie, A. Kiselev, M. Längkvist, F. Klügl, A. Loutfi, An ontology-based reasoning framework for querying satellite images for disaster monitoring, Sensors 17 (2017) 2545. doi:10.3390/s17112545.

[10] H. Couclelis, People manipulate objects (but cultivate fields): Beyond the raster-vector debate in GIS, in: A. U. Frank, I. Campari, U. Formentini (Eds.), Theories and Methods of Spatio-Temporal Reasoning in Geographic Space, Springer, 1992, pp. 65–77.

[11] P. Baumann, D. Misev, V. Merticariu, B. P. Huu, Array databases: Concepts, standards, implementations, J. of Big Data 8 (2021) 1–61. doi:10.1186/s40537-020-00399-2.

[12] Y. Li, T. R. Bretschneider, Semantic-sensitive satellite image retrieval, IEEE Trans. on Geoscience and Remote Sensing 45 (2007) 853–860. doi:10.1109/TGRS.2007.892008.

[13] M. Quartulli, I. G. Olaizola, A review of EO image information mining, ISPRS J. of Photogrammetry and Remote Sensing (2013).

[14] P. Baumann, A. Dehmel, P. Furtado, R. Ritsch, N. Widmann, The multidimensional database system rasdaman, in: Proc. of the 19th ACM Int. Conf. on Management of Data (SIGMOD), ACM Press, 1998, pp. 575–577. doi:10.1145/276304.276386.

[15] P. Baumann, Rasdaman - raster data manager, 2018. doi:10.5281/zenodo.1163021.

[16] S. Grumbach, P. Rigaux, L. Segoufin, Manipulating interpolated data is easier than you thought, in: Proc. of the 26th Int. Conf. on Very Large Data Bases (VLDB), 2000, pp. 156–165. URL: http://www.vldb.org/conf/2000/P156.pdf.

[17] F. Silva-Coira, J. R. Paramá, S. Ladra, J. R. López, G. Gutiérrez, Efficient processing of raster and vector data, Plos One 15 (2020) e0226943. doi:10.1371/journal.pone.0226943.

[18] Open Geospatial Consortium, OGC® Web Feature Service 2.0 Interface Standard – With Corrigendum, https://docs.ogc.org/is/09-025r2/09-025r2.html, 2014.

[19] Open Geospatial Consortium, OGC® Web Coverage Service 2.0 Interface Standard - Earth Observation Application Profile, https://docs.ogc.org/is/10-140r2/10-140r2.html, 2018.

[20] Environmental Systems Research Institute Inc., How zonal statistics works, arcgis 10.8, https://desktop.arcgis.com/en/arcmap/latest/tools/spatial-analyst-toolbox/how-zonal-statistics-works.htm, 2021. (Accessed on 13 July 2023).

[21] M. Neteler, GRASS GIS manual:v.rast.stats, https://grass.osgeo.org/grass82/manuals/v.rast.stats.html, 2022. (Accessed on 13 July 2023).

[22] P. G. Brown, Overview of SciDB: Large scale array storage, processing and analysis, in: Proc. of the 31st ACM Int. Conf. on Management of Data (SIGMOD), 2010, pp. 963–968.

[23] N. R. Brisaboa, G. de Bernardo, G. Gutiérrez, M. R. Luaces, J. R. Paramá, Efficiently querying vector and raster data, The Computer Journal 60 (2017) 1395–1413.

[24] G. De Bernardo, S. Álvarez-García, N. R. Brisaboa, G. Navarro, O. Pedreira, Compact querieable representations of raster data, in: Proc. of the 20th Int. Symp. on String Processing and Information Retrieval (SPIRE), 2013, pp. 96–108.

[25] S. Ladra, J. R. Paramá, F. Silva-Coira, Scalable and queryable compressed storage structure for raster data, Information Systems 72 (2017). doi:10.1016/j.is.2017.10.007.

[26] S. Singla, A. Eldawy, T. Diao, A. Mukhopadhyay, E. Scudiero, The Raptor join operator for processing big raster + vector data, in: Proc. of the 29th Int. Conf. on Advances in Geographic Information Systems (SIGSPATIAL), ACM, 2021, pp. 324–335. doi:10.1145/

3474717.3483971.

[27] S. Singla, A. Eldawy, Raptor zonal statistics: fully distributed zonal statistics of big raster+ vector data, in: IEEE Int. Conf. on Big Data (Big Data), IEEE, 2020, pp. 571–580.

[28] K. Bereta, G. Xiao, M. Koubarakis, Ontop-spatial: Ontop of geospatial databases, J. of Web Semantics 58 (2019). doi:10.1016/j.websem.2019.100514.

[29] Open Geospatial Consortium, GeoSPARQL - A Geographic Query Language for RDF Data, https://www.ogc.org/standard/geosparql/, 2012. (Accessed on 15 July 2023).

[30] S. Das, S. Sundara, R. Cyganiak, R2RML: RDB to RDF Mapping Language, W3C Recommendation, World Wide Web Consortium, 2012. Available at http://www.w3.org/TR/r2rml/.

[31] S. B. Almobydeen, J. R. Viqueira, M. Lama, GeoSPARQL query support for scientific raster array data, Computers & Geosciences 159 (2022) 105023.

[32] A. Andrejev, D. Misev, P. Baumann, T. Risch, Spatio-temporal gridded data processing on the Semantic Web, in: IEEE Int. Conf. on Data Science and Data Intensive Systems, 2015, pp. 38–45. doi:10.1109/DSDIS.2015.109.

[33] H. Beaudoing, M. Rodell, NASA/GSFC/HSL, GLDAS Noah Land Surface Model L4 3 hourly 0.25 x 0.25 degree, Version 2.1, 2020. doi:10.5067/E7TYRXPJKWOQ.

[34] I. Prapas, S. Kondylatos, I. Papoutsis, FireCube: A daily datacube for the modeling and analysis of wildfires in Greece, 2022. doi:10.5281/zenodo.4943353.

[35] Open Geospatial Consortium, Well-known text (WKT) representation of coordinate reference systems, https://docs.ogc.org/is/18-010r7/18-010r7.html, 2019.

[36] D. K. Hall, G. A. Riggs., MODIS/Terra Snow Cover Daily L3 Global 500m SIN Grid, Version 6, 2016. doi:10.5067/MODIS/MOD10A1.006.

[37] Z. Wan, S. Hook, G. Hulley, MODIS/Terra Land Surface Temperature/Emissivity Daily L3 Global 1km SIN Grid V061, 2021. doi:10.5067/MODIS/MOD11A1.061.

[38] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family, J. of Automated Reasoning 39 (2007) 385–429. doi:10.1007/s10817-007-9078-x.

[39] I. S. Mumick, S. J. Finkelstein, H. Pirahesh, R. Ramakrishnan, Magic is relevant, SIGMOD Record 19 (1990) 247–258.

[40] W. Faber, An introduction to Answer Set Programming and some of its extensions, in: Reasoning Web: Declarative Artificial Intelligence – 16th Int. Summer School Tutorial Lectures (RW), volume 12258 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 149–185. doi:10.1007/978-3-030-60067-9_6.

[41] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro, G. Xiao, Ontop: Answering SPARQL queries over relational databases, Semantic Web J. 8 (2017) 471–487. doi:10.3233/SW-160217.

[42] G. Xiao, D. Lanti, R. Kontchakov, S. Komla-Ebri, E. Güzel-Kalayci, L. Ding, J. Corman, B. Cogrel, D. Calvanese, E. Botoeva, The virtual knowledge graph system Ontop, in: Proc. of the 19th Int. Semantic Web Conf. (ISWC), volume 12507 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 259–277. doi:10.1007/978-3-030-62466-8_17.