OntoRaster: Extending VKGs with Raster Data

Arka Ghosh[1(✉)], Albulen Pano[2], Guohui Xiao[3], and Diego Calvanese[1,2]

[1] Department of Computing Science, Umeå Universitet, Umeå, Sweden
{arka.ghosh,diego.calvanese}@umu.se
[2] Faculty of Engineering, Free University of Bozen-Bolzano, Bolzano, Italy
{albulen.pano,diego.calvanese}@unibz.it
[3] Department of Information Science and Media Studies, University of Bergen, Bergen, Norway
guohui.xiao@uib.no

Abstract. The Virtual Knowledge Graph (VKG) paradigm facilitates access to large heterogeneous data sources by leveraging an OWL 2 QL ontology representing the domain knowledge and a set of declarative R2RML mapping assertions. We are interested in heterogeneous data sources consisting of relational data together with spatial geometrical data (a.k.a. *vector data*) and large multidimensional *raster data*. The latter forms of data pose a significant challenge for traditional DBMSs to manage effectively and are instead efficiently processed by tailored array database management systems (*ArrayDBMS*s). To query such data within the VKG paradigm, we propose a novel framework, called OntoRaster, that allows for integrated query processing of relational, raster, and vector data, by keeping each type of data in the system tailored for their efficient processing, while minimising costly data-transfer operations. In OntoRaster, we devised custom raster functions extending SPARQL to query raster data efficiently and developed mechanisms for delegating their computation to the ArrayDBMS. We have implemented the whole framework as an extension of the state-of-the-art VKG system *Ontop* and have demonstrated its effectiveness and efficiency through a curated case study.

Keywords: Virtual Knowledge Graphs · Spatial-Temporal Reasoning · Raster Data · Vector Data · Multidimensional Arrays · Query Answering

1 Introduction

Scientific and business applications produce data in different representations at a massive scale, leading to bottlenecks in data management and processing and giving rise to the notorious issue of data heterogeneity [27,28]. Resolution of data heterogeneity allows for multiple databases (DBs) to be integrated and their contents to be uniformly queried. In our study, we are considering two different types of data: traditional relational data and large-scale raster data represented as multidimensional arrays [19,25]. We are particularly interested in spatial data in the Geographic Information System (GIS) field, which are classified into two types: vector data and raster data. *Vector data* are a special kind of relational data that represent geometries (e.g., points, lines, and polygons) of real-world discrete features such as static locations (points), road networks

S. Kirrane et al. (Eds.): RuleML+RR 2024, LNCS 15183, pp. 108–123, 2024.
https://doi.org/10.1007/978-3-031-72407-7_9

(lines), and country boundaries (polygons), with additional attributes. Geospatial *raster data* are a notable example of generic raster data, which have become a prime source of modern-day big data, e.g., due to advancements in air-borne and space-borne remote sensing [23]. Geospatial raster data are characterised by differences in spatial, temporal, and spectral resolutions, varying conditions in observations (e.g., different atmospheric conditions), and diverse spatial data structures [18]. Tailored array DB systems (Array DBMSs) are used to store and manage multidimensional arrays due to their outstanding scalability, adaptability, and in-place data processing capacity [6,29]. In our work, we decided to exploit the capabilities of *rasdaman*[1] (for "raster data manager") [3,4], a domain-agnostic Array DBMS that supports a variety of array operations, such as aggregation, filtering, scaling, and extraction of sub-spaces.

The support for raster data has also been attempted in classical relational DB management systems (RDBMSs) using spatial extensions. To do so, large raster data must undergo a conversion (e.g., using *raster2pgsql*[2] for PostgreSQL extended with PostGIS) from their native format of multidimensional arrays to a suitable relational form (e.g., *postgis raster*) that can be queried using SQL. This conversion leads in general to a massive increment in the data volume, which might be difficult to manage in a relational table. Therefore, we take an alternative approach that does not rely on the RDBMS's internal materialisation of raster data.

To address data heterogeneity resulting from the combination of the different forms of data, we rely on the paradigm of *Virtual Knowledge Graphs* (VKGs) also known as *Ontology-Based Data Access* (OBDA) [22,33]. In VKGs, domain knowledge is conceptually represented as an *ontology*, typically expressed in the lightweight ontology language OWL 2 QL [21], while actual data are maintained in a relational *data source*, but are *not* materialised at the conceptual level (which justifies the term "virtual"). To establish the relationship between the ontology and the data at the source, VKGs rely on a declarative specification, provided in terms of a set of *mapping assertions*.

In the traditional VKG setting, date sources are restricted to RDBMSs, but such systems are inefficient at querying raster data in their native multidimensional array format [7,20], and at combining them with relational data, including vector data. Various recent proposals aim at addressing these challenges, as also discussed in [15]. Notably, a semantic data cube system, named *Plato* [9,10], has been proposed within the EU H2020 project DeepCube[3]. The system relies on a geospatial extension of the VKG system Ontop [8] and uses PostgreSQL Foreign Data Wrappers. It has been deployed in a case study on fire risk-management [11]. Similarly, another study [16] proposed an architecture to query raster data stored in an RDBMS using the VKG paradigm and extending the GeoSPARQL ontology[4] with a semantic representation model for raster data cubes. Our work differs from these works as we rely on rasdaman's array management abilities to query multidimensional raster data without any conversion.

In this paper, we present our ongoing work on the OntoRaster framework, which allows for integrated querying of both relational data and raster data by extending the

[1] http://www.rasdaman.org/.
[2] https://postgis.net/docs/using_raster_dataman.html#RT_Loading_Rasters.
[3] https://deepcube-h2020.eu/.
[4] https://www.ogc.org/standard/geosparql/.

VKG paradigm to handle also array-based sources. In OntoRaster, we still use a relational DBMS, namely PostgreSQL, as the main data source to which the VKG system connects, and we provide within this DBMS novel functionalities for virtual access to an Array DBMS, namely rasdaman. Specifically, we have extended the SPARQL query language with special raster functions (e.g., to compute multi-dimensional aggregations), and we have devised a set of stored procedures for PostgreSQL that translate such raster functions into array operations supported by rasdaman. These are executed directly on the Array DBMS, hence avoiding costly transfers and transformations into relational data of large arrays (unless such arrays need to be returned as an answer to a query, in which case the data transfer is unavoidable).

2 Preliminaries

In the standard VKG paradigm [22,33], existing heterogeneous data sources (e.g., relational) are accessed via a domain ontology that is linked to the source through semantic mappings, and exposes the underlying data as a (virtual) knowledge graph represented in RDF [26]. Formally, a *VKG specification* $\mathcal{P} = (\mathcal{O}, \mathcal{M}, \mathcal{S})$ consists of *(i)* an *ontology* $\mathcal{O}$ that represents intensional knowledge about the domain of interest and is expressed as a TBox in the lightweight ontology language OWL 2 QL [21], *(ii)* a relational *data source schema* $\mathcal{S}$, and *(iii)* a declarative mapping $\mathcal{M}$ that associates to each element (i.e., class or property) in $\mathcal{O}$ a (SQL) query over $\mathcal{S}$, specifying how to (virtually) populate that element through the data retrieved from the source.

In traditional VKGs, the mapping $\mathcal{M}$ consists of a set of R2RML [13] *mapping assertions* of the form $\mathtt{sql}(\boldsymbol{x}) \rightsquigarrow E(\boldsymbol{f}(\boldsymbol{x}))$, where $\mathtt{sql}(\boldsymbol{x})$ is a (SQL) query over $\mathcal{S}$ with answer variables $\boldsymbol{x}$, E is a class or property of $\mathcal{O}$, and $\boldsymbol{f}(\boldsymbol{x})$ denotes a set of so-called *IRI templates* applied to the variables in $\boldsymbol{x}$. Each IRI template is a function that constructs an ontology literal or an IRI identifying an ontology object, from the values in each answer to $\mathtt{sql}(\boldsymbol{x})$ instantiating $\boldsymbol{x}$. Then, a *VKG instance* is a pair $(\mathcal{P}, \mathcal{D}^{rel})$, where $\mathcal{D}^{rel}$ is a relational database instance conforming to $\mathcal{S}$. Thus, by applying the mapping assertions in $\mathcal{M}$ to $\mathcal{D}^{rel}$, one obtains a knowledge graph $\mathcal{M}(\mathcal{D}^{rel})$ (which is actually kept virtual).

Semantic queries formulated in SPARQL [17] are posed over $\mathcal{O}$ and are answered by accessing the data in $\mathcal{D}^{rel}$ through the mapping $\mathcal{M}$. Specifically, given a SPARQL query q over a VKG instance $\mathcal{J} = (\mathcal{P}, \mathcal{D}^{rel})$, we are interested in the *certain answers* to q over $\mathcal{J}$, denoted $\mathsf{cert}(q, \mathcal{J})$, which are the answers obtained by evaluating q over the knowledge base $(\mathcal{O}, \mathcal{M}(\mathcal{D}^{rel}))$ under the OWL 2 QL entailment regime [21]. Actual VKG systems such as Ontop [12,34] avoid costly materialization of the KG $\mathcal{M}(\mathcal{D}^{rel})$ and its storage in a triple store, and rather translate the SPARQL query into a relational SQL query that is directly evaluated by the underlying DBMS (e.g., PostgreSQL), thus ensuring also freshness of query answers concerning source updates.

2.1 Vector Data

Vector data are used to describe the spatial characteristics of discrete real-world phenomena, each conceived as a feature according to the *ISO 19123 Geographic Inf. Part 1*

OGC Geometry Type	Region Geometry Illustration	Exterior & interior boundaries	OGC WKT Literal Representation (i.e. regionWkt)	Description
Point		**NA**	**POINT** (x, y)	A WKT point (e.g. pin location)
LineString		**NA**	**LINESTRING**($POINT_{01}$,$POINT_{02}$,.....,$POINT_{m}$)	A LineString with at least two or more points (e.g., road network)
LinearRing		**exterior = 1 interior = 0**	**LINEARRING** ($POINT_{01}$,$POINT_{02}$,.....,$POINT_{n}$, $POINT_{01}$)	An enclosed LineString where start point = end point with zero measurable area
Polygon		**exterior = 1 interior = 2**	**POLYGON** (($LINEARRING_{01}$), [(LINEARRING)]*)	A LinearRing with a valid measurable area with zero or more holes or interiors (e.g., countries, lake within forest)
Multi-Polygon		**exterior = 3 interior = 3**	**MULTIPOLYGON** ((($POLYGON_{01}$), [(POLYGON)]*))	Collection of two or more Polygons with zero or more interiors or holes (e.g., scattered islands, enclaves)
Geometry-Collection		**exterior = 4 interior = 2**	**GEOMETRYCOLLECTION**([POINT*], [LINEARSTRING*], [POLYGON*], [MULTIPOLYGON*])	Heterogeneous collection of every OGC standard geometries

Fig. 1. Primary components of vector data in their respective OGC WKT forms

Standard[5]. Typical examples of such discrete features are rivers, lakes, and administrative regions. These are represented by a set of one or more geometric primitives, such as points, curves, and surfaces with their positional parameters either in Cartesian coordinates (`X` and `Y`) in a topological planar surface or geographic coordinates (`longitude` a.k.a. `geoX` and `latitude` a.k.a. `geoY`). Other characteristics of the discrete phenomenon are recorded as feature attributes. Vector data are represented in popular file formats like CSV, shapefiles, and GeoJSON or relational tables in a RDBMS with a spatial extension.

We consider geometries that conform to the *OGC ISO 19125 OpenGIS* Standard[6], which foresees *Well-Known Text* (WKT) literals[7] as the most common representation of geometries. According to this standard, a *Point* is a pair of longitude and latitude (which in the WKT representation are separated by blanks), while a *LineString* is a sequence of points separated by ','. A valid *Polygon* is a topologically closed planar surface defined by one exterior boundary and zero or more interior boundaries (where each interior boundary defines a hole in the polygon). Each boundary is a *LinearRing*, which is a LineString in which the last point must coincide with the first point. A *MultiPolygon* is a collection of one or more valid Polygons, separated by','. Finally, a *GeometryCollection* is a sequence of one or more of the previously introduced elements. Figure 1 illustrates some of the basic geometry elements of vector data (i.e., `regionGeometry`) with their representation as OGC WKT literals (i.e., `regionWkt`) with examples. All geometries in OGC ISO 19125 are a combination of these primary geometries.

[5] https://www.iso.org/obp/ui/en/#iso:std:iso:19123:-1:ed-1:v1:en.
[6] https://www.ogc.org/standard/sfa/.
[7] https://docs.ogc.org/is/18-010r7/18-010r7.html.

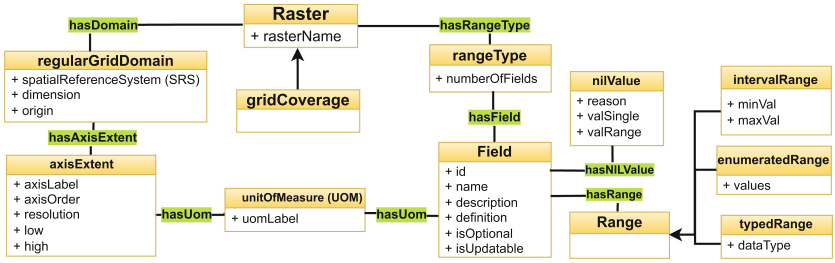

Fig. 2. Ontology for generic raster data, including grid coverage

2.2 Raster Data

Raster data, also referred to as *gridded data* [7], or *multidimensional discrete data* (MDD), represent real-world phenomena that vary continuously over space, time, and maybe even more dimensions, depending on the particular *domain of interest* (DOI) or field such as medical, astronomy, geospatial, etc. In this paper, we focus on raster data in the geospatial domain, also referred to as *datacube* [2] or gridded *coverage* as per *OGC Coverage Implementation Schema* (CIS) v1.1 Standard [5] adopted by ISO 19123 Geographic information - Schema for coverage geometry and functions. According to Part 2 of this ISO standard[8] the concept of *coverage* represents multidimensional grids of both regular and irregular type as the natural representation of various space-time-varying phenomena predominantly in the geospatial domain. Common examples include 1-D time series, 2-D imagery, 3-D `x/y/t` image time-series and `x/y/z` geophysical data, as well as 4-D `x/y/z/t` atmospheric climate and ocean data.

Inspired by [1], we have defined the *Raster Ontology* shown in Fig. 2, which describes n-dimensional generic raster data or coverage based on the OGC CIS v1.1 Standard. The ontology describes so far only regular gridded coverage or geospatial raster data. The `RegularGridDomain` and `RangeType` classes capture all the information about the domains and ranges of a grid coverage. For measurement units, we use the `UnitOfMeasure` class defined by the *Quantities, Units, Dimensions, and Types* (QUDT) ontology [24].

Coverages can be encoded in a suitable raster file format such as GML, NetCDF, GeoTIFF, HDF, etc., which greatly benefits for use case specific small datasets that can be accessed ad hoc via Python, R, Matlab scripts, etc. However, problems begin to emerge when data become very large in volume (e.g. terabytes) and it becomes challenging to query the required information [30]. Moreover, to deal with gridded coverage data, or raster data, one must also include metadata such as domain, range, and provenance information. In contrast to arrays, the structure of this metadata is substantially less regular, and it may be incomplete or vary between arrays.

In our research, we rely on rasdaman (for "raster data manager"), a domain-agnostic array DBMS that implements the OGC standards for gridded coverages, and manages such large data with its substantially rich array algebra. Rasdaman offers a SQL-like

[8] https://www.iso.org/obp/ui/en/#iso:std:70948:en.

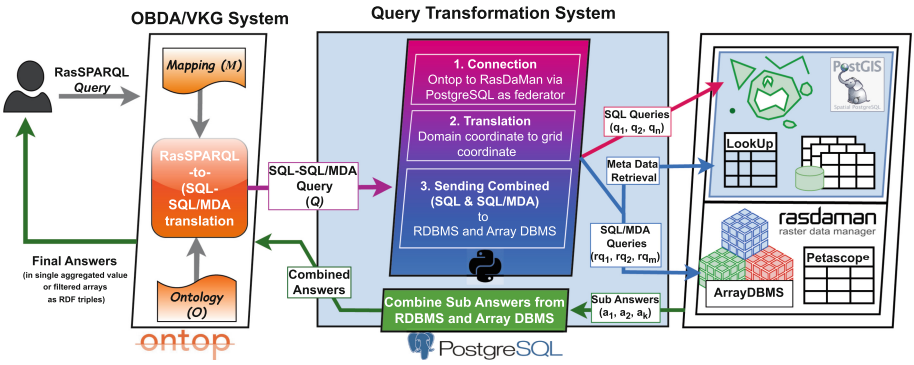

Fig. 3. The ONTORASTER framework extending the VKG framework

query language known as *RaSQL*[9] to query any kind of raster data of arbitrary dimensions [4] following *ISO 9075 SQL Part 15: Multi-Dimensional Arrays* (SQL/MDA)[10]. It features a geo service front-end component called *petascope*[11], which adds geo semantics on top of arrays, thereby enabling regular and irregular grids based on the OGC CIS v1.1. To implement the geo-semantics, petascope uses a relational database such as PostgreSQL to store related metadata for each raster dataset. This metadata is distributed across 62 relational tables in a separate database called *petascopedb* within PostgreSQL. Rasdaman also includes *rasdapy*[12], a client API that allows building and executing rasql queries using Python.

3 The OntoRaster Framework

We present the ONTORASTER framework, which extends the capabilities of a VKG engine to handle also multidimensional raster datasets. The architecture of ONTORASTER is shown in Fig. 3, and discussed below.

3.1 RasSPARQL: An Extension of SPARQL with Raster Functions

We introduce now RasSPARQL, which extends SPARQL and the OGC GeoSPARQL functions used to manage vector data [34], with the support for raster data and the corresponding functions. To provide such an extension, we have first devised and added to SPARQL, custom raster-based functions and corresponding SQL DB functions, to be handed over for execution to an Array DBMS. The currently supported RasSPARQL raster-based functions with their input arguments and output type are listed in Table 1.

Notice that the RasSPARQL functions `rasClipRaster()`, and `rasClipRaster AnyGeom()` return a portion of raster data as an array of values (e.g., pixels) by relying on rasdaman's embedded 'clip' function[13], which extracts a raster array based on

[9] https://doc.rasdaman.org/04_ql-guide.html.
[10] https://www.iso.org/obp/ui#iso:std:iso-iec:9075:-15:ed-2:v1:en.
[11] https://doc.rasdaman.org/02_inst-guide.html?highlight=petascope#petascope.
[12] https://pypi.org/project/rasdapy3/.
[13] https://doc.rasdaman.org/04_ql-guide.html#clipping-operations.

Table 1. RasSPARQL raster functions and respective PL/Python stored procedures

RasSPARQL function	Input arguments	Output type	PL/Python stored proc.
`rasDimension()`	`rasterName`	`xsd:string`	`query2string()`
`rasCellOp()`	`timeStamp, operator, operand, rasterName`	`xsd:string`	`query2array()`
`rasSpatialAverage()`	`timeStamp, regionGeometry, rasterName`	`xsd:double`	`query2numeric()`
`rasSpatialMinimum()`	`timeStamp, regionGeometry, rasterName`	`xsd:double`	`query2numeric()`
`rasSpatialMaximum()`	`timeStamp, regionGeometry, rasterName`	`xsd:double`	`query2numeric()`
`rasTemporalAverage()`	`startTime, endTime, regionGeometry, rasterName`	`xsd:double`	`query2numeric()`
`rasTemporalMinimum()`	`startTime, endTime, regionGeometry, rasterName`	`xsd:double`	`query2numeric()`
`rasTemporalMaximum()`	`startTime, endTime, regionGeometry, rasterName`	`xsd:double`	`query2numeric()`
`rasClipRaster()`	`timeStamp, regionGeometry, rasterName`	`xsd:string`	`query2array()`
`rasClipRasterAnyGeom()`	`timeStamp, regionGeometry, rasterName`	`xsd:string`	`query2array()`

the geometry (or shape) of a region: all pixels outside of the given region are set to null or 'NaN', while pixels on and within the region are preserved. We represent the returned array as a string since RDF does not support arrays yet.

3.2 Extending Ontop to Translate RasSPARQL to SQL-SQL/MDA

To properly deal with RasSPARQL functions within Ontop, we have implemented an extension of the system that, as a part of query reformulation, translates each such function into a corresponding SQL function and embeds it into the generated SQL/MDA query as indicated in Fig. 3. Notice that the RasSPARQL query is in general a SPARQL query that might contain both GeoSPARQL functions and raster functions. This query is translated to a plain SQL query containing corresponding PostGIS functions and PL/Python stored procedures that connect to rasdaman, as specified in the last column of Table 1. As an example, when Ontop parses a RasSPARQL query that embeds the raster function `rasSpatialAverage`, it translates it to a call to `query2numeric` which in turn executes the SQL/MDA standard rasql query over rasdaman.

3.3 Query Transformation System

We now describe how the generated SQL-SQL/MDA query produced by Ontop is processed as shown in Fig. 3. The PostGIS functions embedded in the SQL part can be directly processed by PostgreSQL through its PostGIS extension. Instead, we rely on PL/Python and PL/pgSQL stored procedures to ensure smooth retrieval of the metadata corresponding to the relevant raster data, and the ability to call suitable rasql queries that are directly executed by rasdaman.

PL/Python Stored Procedures. The stored procedures are specified in the PL/Python procedural language[14], which also supports all PostgreSQL and PostGIS functions, and *rasdapy* to connect to rasdaman. The procedures are stored in the VectorTablesDB database inside PostgreSQL (in a schema called `rasdaman_op`). We elaborate now on the stored procedures shown in Table 2.

[14] https://www.postgresql.org/docs/current/plpython.html.

Table 2. Selected PL/Python stored procedures to connect Ontop and rasdaman federated by PostgreSQL via rasdapy

Stored procedure	Input arguments	Output
`geo2grid_coords()`	`GEOMETRY regionGeometry, DOUBLE minLon, DOUBLE minLat, DOUBLE resLon, DOUBLE resLat`	`GEOMETRY regionGrid`
`query2numeric()`	`STRING rasqlQuery`	`DOUBLE value`
`query2array()`	`STRING rasqlQuery`	`DOUBLE[] array`

- `geo2grid_coords()`: Being a domain-agnostic array DBMS, rasdaman (rasql precisely) only supports array indices (`i` and `j`) or grid coordinates (`gridX` and `gridY`) and does not consider any domain-specific coordinates such as geo-coordinates (i.e., `longitude` and `latitude`) natively. Therefore, we devised this mapping function inspired by a generic affine transformation model [32], which translates the geo-coordinates to corresponding grid coordinates, taking into account the respective geographical coordinate reference system (CRS). Our current implementation assumes that both the vector and the raster data use the same CRS, which we require to be WGS84[15]. We will provide support for different CRSs in the future. In practice, this function takes five input arguments, namely the geometry of the chosen *region of interest* (ROI), and `minLon`, `minLat`, `resLon`, `resLat` of the selected raster data. It returns a translated grid geometry of the region (`regionGrid`) as output. This enables the user to send the geometry (polygon or multi-polygon) of any ROI to rasdaman as a part of the rasql query. The embedded geometry will be translated into grid coordinates to be used as an input to rasql's array operations to extract data from raster data. As an example, rasql's 'clip'operation can crop out a portion of raster data based on the translated geometry of a user's region.
- `query2numeric()`: Takes a rasql query as a string input and executes it over raster data stored inside rasdaman using rasdaman's supported array condenser operations[16], such as `avg_cells`, `max_cells`, etc. and retrieve aggregated numeric results back to PostgreSQL.
- `query2array()`: Evaluates rasql queries over raster arrays using rasdaman supported array operations[17] such as clip, concatenation, scaling and retrieve filtered arrays back to PostgreSQL.

PL/pgSQL Stored Procedures. - `timestamp2grid()`: This PL/pgSQL function translates timestamp from'DateTime' format to integer format (e.g., gridTime) that is comprehended by rasdaman since rasdaman treats the timestamp as an integer instead of the actual'DateTime' format.

These are just selected necessary stored procedures that make PostgreSQL a *federator* between rasdaman and Ontop. In the future, more stored procedures can be added based on the user's demands. Based on these stored procedures (both PL/Python and

[15] https://epsg.io/4326.
[16] https://doc.rasdaman.org/04_ql-guide.html#condensers.
[17] https://doc.rasdaman.org/04_ql-guide.html#array-operations.

PL/pgSQL), Ontop-generated SQL/MDA queries can be executed on any geospatial raster data using the array manipulation capabilities of rasdaman.

Raster LookUp Table Creation. As previously stated, the essential metadata stored in petascopedb must accompany the corresponding raster in order to interact with it. Every time a user queries specific raster data, the relevant metadata needs to be searched among 62 separate tables in petascopedb and attached with the corresponding raster data automatically for subsequent processing. If the user selects another raster dataset to query, the automatic search and combine procedure is repeated, resulting in overhead. To address this issue, we created a *Raster Lookup*, a single table that stores all necessary metadata in one place for every raster data stored in rasdaman. We have built this table inside the VectorTablesDB database from the petascopedb database using *dblink*[18] where petascopedb serves as a remote database. Whenever a new raster data is uploaded to rasdaman, we defined a PostgreSQL trigger to automatically update the *LookUp* table with the metadata of newly added raster data.

4 Case Study for the OntoRaster System

We demonstrate the proposed ONTORASTER framework on a case study where we integrate vector data (i.e., regions) with geospatial raster data and return their spatiotemporal patterns and correlations as a knowledge graph, for possible further processing. We build the mappings using the Protégé ontology editor [14] with the Ontop plugin [12] and set up a SPARQL endpoint available on GitHub[19] including the complete implementation as docker with the required data.

4.1 VKG Specification for OntoRaster

Ontology ($\mathcal{O}$). We rely on the GeoSPARQL ontology, representing the geometries of a vector region (See footnote 4).

Data Sources ($\mathcal{D}^{rel}$, $\mathcal{D}^{arr}$). Table 3 displays all the vector and raster datasets considered for the ONTORASTER framework demonstration. In our work, we represent vector data with their corresponding geometries (e.g., `regionGeometry`) and attributes (e.g., `regionId`, `regionName`), using one or more relational tables for every *area of interest* (AOI), stored in a DB named *VectorTablesDB* within PostgreSQL, as shown in Fig. 4. For the case study, we consider Sweden, Bavaria (Germany), and South Tyrol (Italy) as our AOIs with their corresponding municipalities as ROIs (resulting in ∼500 unique ROIs in total).

Related vector data are downloaded as shapefiles from GADM[20]. Then utilizing the `shp2pgsql`[21] data loader we import each shapefile as a suitable default SQL binary object format (with `regionGeometry` in hex format, i.e., `regionHex`) into separate

[18] https://www.postgresql.org/docs/current/contrib-dblink-function.html.
[19] https://github.com/aghoshpro/OntoRaster.git.
[20] https://gadm.org/index.html.
[21] https://postgis.net/docs/using_postgis_dbmanagement.html#shp2pgsql_usage.

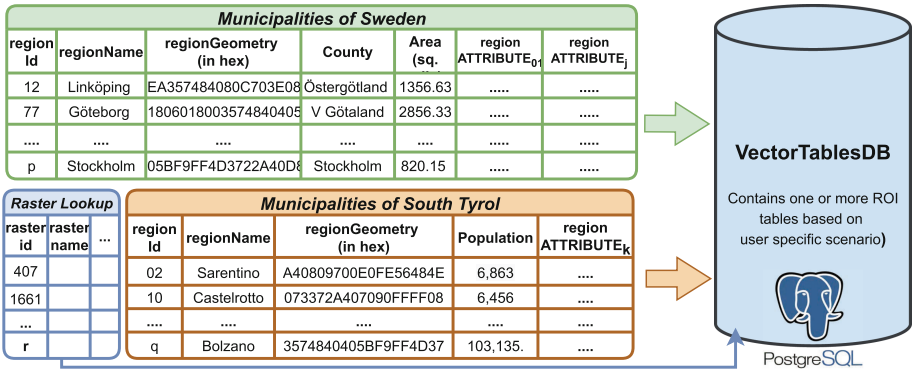

Fig. 4. VectorTablesDB containing tables for Areas of Interest and Raster Lookup

tables into VectorTablesDB, as shown in Fig. 4. Notice that we have used convenient names for the tables and their columns in VectorTablesDB, However, any other choice would have been possible, as long as such names are used in a coherent way in the source queries of the VKG mappings that provide the link between vector and raster data (see, e.g., Fig. 5).

As for raster data, we use the MODIS land surface temperature dataset [31] from NASA's *Earth Science Data Systems* (ESDS) covering the respective AOIs, i.e., Sweden, Bavaria, and South Tyrol. This is 3-D raster data with a spatial dimension (2-D) and a temporal dimension (1-D) with temperature as a field (or band). OntoRaster also supports an arbitrary number of fields, such as precipitation, elevation, etc., sharing the same axis extents (cf. the ontology in Fig. 2). The *Raster Lookup* table, of which we show in Table 4 only five out of seventeen columns, maintains the necessary metadata of the raster datasets and is stored in VectorTablesDB, as shown in Fig. 4.

Mappings ($\mathcal{M}$). We show in Fig. 5 only one example mapping, in which the class `Region` is mapped with its respective data properties such as id, name, and geometry in OGC WKT geometry serialization, i.e., `regionWkt`, for all regions of Sweden. Based on this mapping, at query execution time, the geometry is retrieved using the input region name . The SQL source query of the mapping contains a `CASE` statement that

Table 3. Heterogeneous datasets used for the case study

Areas of Interest (AOI)	Sweden		Bavaria (Germany)		South Tyrol (Italy)	
Datasets	Vector	Raster	Vector	Raster	Vector	Raster
Type	Municipalities	Temperature	Municipalities	Temperature	Municipalities	Temperature
Native Format	.shp	NetCDF	.shp	NetCDF	.shp	NetCDF
Logical Format	relational	MDA	relational	MDA	relational	MDA
Array Dim (t x lon x lat)	–	217 × 1586 × 1648	–	305 × 584 × 396	–	305 × 252 × 106
Spatial Resolution	None	1 km × 1 km	None	1 km × 1 km	None	1 km × 1 km
Temporal Resolution	None	daily	None	daily	None	daily
Temporal Coverage	None	1 year	None	1 year	None	1 year
Features	290	1,001,057,824	96	70,006,640	116	8,038,275

Table 4. Raster Lookup

raster_id	raster_name	min_lon	max_lon	res_lon	...
407	Surface_Temperature_Sweden	10.9583	24.1749	0.00833	...
1661	Bavaria_Temperature_MODIS_1km	8.9749	13.8416	0.008333332653	...
800	South_Tyrol_Temperature_MODIS_1km	10.3833	12.4833	0.0083332586793	...

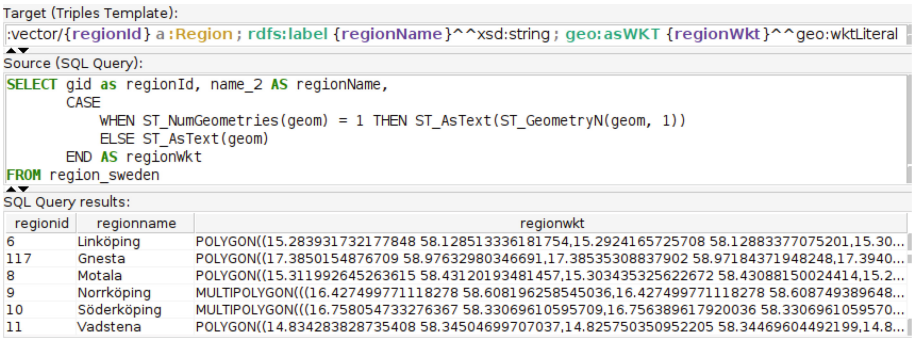

Fig. 5. Example mapping to retrieve region id, name and geometry

checks whether the geometry of the input region is a polygon or a multi-polygon, as they need to be treated differently in the `geo2grid_coords` function. This is necessary to transform any single polygon represented as a multi-polygon (technically possible but a bad representation) into its actual polygonal format. Similar types of mapping apply for all other vector regions, including the regions of Bavaria and South Tyrol.

In our framework, we have set up a number of mappings to connect the data attributes of the `Region` and `Raster` classes, for which we refer the reader to the GitHub repository (See footnote 19). We will further extend this set of mappings in the future.

4.2 Queries (Q)

To demonstrate our results, and the capability of OntoRaster for query answering over virtually integrated vector data and raster data under the VKG paradigm, we provide RasSPARQL queries of four types: simple, with spatial aggregation, with temporal aggregation, and with spatial raster filtering. All queries can be found on GitHub (See footnote 19). Here we show only a few of them. The used prefixes are listed in Table 5 and refer to the base namespaces of the OntoRaster and auxiliary vocabularies, such as RDFS and GeoSPARQL.

Q1 is a simple raster query that retrieves dimension information of the input raster data. Query *Q2* conducts element-wise operations on raster arrays.

Queries *Q3–Q5* perform spatial aggregations and return a single aggregated value from a chosen raster dataset over a vector region at a specific timestamp.

Table 5. List of prefixes used for RasSPARQL queries

Prefix	IRI Namespace
:	https://github.com/aghoshpro/OntoRaster/
geo	http://www.opengis.net/ont/geosparql#
rdfs	http://www.w3.org/2000/01/rdf-schema#
rasdb	https://github.com/aghoshpro/RasterDataCube/

– ***Q3***: Find the spatial average temperature over München on 24 July 2023, using the available temperature raster data over Bavaria.

```
SELECT ?answer {
  ?region a :Region ; rdfs:label ?regionName ; geo:asWKT ?regionWkt .
  ?gridCoverage a :Raster ; rasdb:rasterName ?rasterName .
  FILTER (?regionName = 'München')
  FILTER (CONTAINS(?rasterName, 'Bavaria')
  BIND ('2023-07-24T00:00:00+00:00'^^xsd:dateTime AS ?timeStamp)
  BIND (rasdb:rasSpatialAverage(?timeStamp, ?regionWkt, ?rasterName)
      AS ?answer)
}
```

Similarly, *Q4* and *Q5* enable the user to find the maximum and minimum temperature values by using the respective RasSPAQRL spatial functions `rasSpatialMaximum` and `rasSpatialMinimum` over `'München'`, or any other region from VectorTablesDB, combined with the corresponding raster datasets (i.e., `Bavaria`, `Sweden`, or `South-Tyrol`).

Queries *Q6–Q8* perform temporal aggregations and return a single aggregated value by integrating a user-specific vector region and a raster dataset over a time interval between the given start and end times.

– ***Q7***: Find the maximum temperature over Göteborg between 5 April 2022 and 19 June 2022, using the available temperature raster data over Sweden.

```
SELECT ?answer {
  ?region a :Region ; rdfs:label ?regionName ; geo:asWKT ?regionWkt .
  ?gridCoverage a :Raster ; rasdb:rasterName ?rasterName .
  FILTER (?regionName = 'Göteborg')
  FILTER (CONTAINS(?rasterName, 'Sweden'))
  BIND ('2022-04-05T00:00:00+00:00'^^xsd:dateTime AS ?startTimeStamp)
  BIND ('2022-06-19T00:00:00+00:00'^^xsd:dateTime AS ?endTimeStamp)
  BIND (rasdb:rasTemporalMaximum(?startTimeStamp, ?endTimeStamp,
         ?regionWkt, ?rasterName) AS ?answer)
}
```

Queries *Q9* and *Q10* perform a 'clipping' operation by cropping a portion of the actual raster data based on the geometrical extent of the specified region at a given time, and return a filtered array that covers the input region.

– ***Q9***: Clip an available temperature raster data for South Tyrol with region Bolzano on 24 September 2023 at time 00:00:00, and return filtered arrays.

```
SELECT ?answer {
  ?region a :Region ; rdfs:label ?regionName ; geo:asWKT ?regionWkt .
  ?gridCoverage a :Raster ; rasdb:rasterName ?rasterName .
  FILTER (?regionName = 'Bolzano')
  FILTER (CONTAINS(?rasterName, 'Tyrol'))
  BIND ('2023-09-24T00:00:00+00:00'^^xsd:dateTime AS ?timeStamp)
  BIND (rasdb:rasClipRaster(?timeStamp, ?regionWkt, ?rasterName)
      AS ?answer)
}
```

– ***Q10***: Clip a portion of user-specific raster data based on a custom vector region on 24 July 2024 at time 00:00:00, and return filtered arrays.

```
SELECT ?answer {
  ?gridCoverage a :Raster ; rasdb:rasterName ?rasterName .
  FILTER (CONTAINS(?rasterName, 'Bavaria'))
  BIND ('POLYGON((11.324157714843748 48.29050321714061,
         11.911926269531248 48.279537342260085,
         11.88995361328125 48.01932418480118,
         11.340637207031248 48.01564978668938,
         11.324157714843748 48.29050321714061))' AS ?customRegionWkt)
  BIND ('2023-07-24T00:00:00+00:00'^^xsd:dateTime AS ?timeStamp)
  BIND (rasdb:rasClipRasterAnyGeom(?timeStamp, ?customRegionWkt,
        ?rasterName) AS ?answer)
}
```

4.3 Issues

Currently, the OntoRaster system provides support for the types of RasSPARQL queries shown in Sect. 4.2. However, the system is still under development and we intend to extend it to support more general forms of RasSPARQL queries, so as to accommodate arbitrary multivariate raster data of n-dimensions. We are also working to make our solution more robust by addressing the following issues:

– Multi-polygons with holes and multi-polygons with small polygons (sometimes the area is smaller than the pixel size of the corresponding raster arrays) still have limited support. Hence, the `geo2grid_coords` function needs to be improved to translate the aforementioned types of multi-polygonal geometry to an equivalent grid geometry.
– Multiple raster data sets with the same name have to be taken into account. A possible solution is to include a second filter argument in addition to the raster name, to distinguish different fields of the raster.
– Missing timestamps between the start time and end time of the raster data are not taken into account in the `timestamp2grid` function.
– RDF does not yet support array datatype, leading to optimisation issues when raster arrays that are retrieved as RDF strings require further processing.

– The inclusion of post-processing operations for raster data is crucial in converting the raw data values to a format that is convenient for the end user. But their usage differs for every raster dataset, may not always be included in the metadata, and whether to use them or not, depends on the end user's objective with the data. All these aspects need to be taken into account.

5 Conclusions

We have presented a novel extended VKG framework, OntoRaster to query for the first time multidimensional raster data combined with relational data, including vector data, on the fly by connecting the array DBMS rasdaman to the VKG system Ontop. To achieve this, we defined the RasSPARQL language, by extending SPARQL with custom raster functions to query over VKGs. Then we developed a query transformation system that includes several stored procedures defined in PL/Python and PL/pgsql that make PostgreSQL a federator between VKG engine Ontop and heterogeneous data sources (e.g., raster and relational). It guides the translated SQL-SQL/MDA query coming from Ontop to the respective data sources stored in RDBMS & array DBMS as mentioned in the RasSPARQL query by the user and executes them separately. After execution, the retrieved sub-answers (containing relational and raster arrays) are joined and then guided back to the Ontop to generate a virtual knowledge graph (VKG) that answers the user's semantic query. The most notable contribution of this article lies in its utilisation of the array handling capabilities of an array DBMS to perform queries on large raster data in their original data structure, i.e., multidimensional arrays, without any conversion. This eliminates the need for costly data transfer and transformation of raster data into relational databases. OntoRaster supports a wide range of potential queries, only limited to the data processing and functional capabilities of PostgreSQL (RDBMS) and rasdaman (array DBMS) for managing relational data and raster data, respectively.

We plan to continue the work on OntoRaster along the following lines:

– It is of interest to incorporate SPARQL endpoints of public knowledge graphs (e.g., DBPedia, LinkedGeoData, and Wikidata) into the OntoRaster framework, to enable the building of a coherent (partially virtual and partially materialized0 knowledge graph that can be queried holistically.
– The possibility to delegate further procession of the query results to BI Tools such as MS Power BI and Tableau.
– Publishing and displaying query results as an interactive visualisation on the web, using the OGC CoverageJSON[22].
– Translating natural language queries into RasSPARQL queries to explore VKG using NLP and LLMs.

Acknowledgments. This research has been partially supported by the Province of Bolzano and DFG through the project D2G2 (DFG grant n. 500249124), by the HEU project CyclOps (grant agreement n. 101135513), and by the Wallenberg AI, Autonomous Systems and Software Program (WASP), funded by the Knut and Alice Wallenberg Foundation.

[22] https://covjson.org/.

References

1. Andrejev, A., Misev, D., Baumann, P., Risch, T.: Spatio-temporal gridded data processing on the semantic web. In: Proceedings of IEEE International Conference on Data Science and Data Intensive Systems (DSDIS) (2015). https://doi.org/10.1109/DSDIS.2015.109
2. Baumann, P.: The datacube manifesto. Technical Report, EU EarthServer (2017). https://earthserver.eu/tech/datacube-manifesto/The-Datacube-Manifesto.pdf
3. Baumann, P.: RasDaMan - Raster data manager. Technical Report, 9.5.0, Zenodo (2018). https://doi.org/10.5281/zenodo.1163021
4. Baumann, P., Furtado, P., Ritsch, R., Widmann, N.: The rasdaman approach to multidimensional database management. In: Proceedings of the 12th ACM Symposium on Applied Computing (SAC), pp. 166–173 (1997). https://doi.org/10.1145/331697.331732
5. Baumann, P., Hirschorn, P.E., Masó, J.: OGC coverage implementation schema (CIS), v1.1. Technical Report, Open Geospatial Consortium (2017). http://docs.opengeospatial.org/is/09-146r8/09-146r8.html
6. Baumann, P., Holsten, S.: A comparative analysis of array models for databases. In: Kim, T., et al. (eds.) FGIT 2011. CCIS, vol. 258, pp. 80–89. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-27157-1_9
7. Baumann, P., Misev, D., Merticariu, V., Huu, B.P.: Array databases: concepts, standards, implementations. J. Big Data **8**(28) (2021). https://doi.org/10.1186/s40537-020-00399-2
8. Bereta, K., Xiao, G., Koubarakis, M.: Ontop-spatial: ontop of geospatial databases. J. Web Semantics **58** (2019). https://doi.org/10.1016/j.websem.2019.100514
9. Bilidas, D., Mantas, A., Yfantis, F., Stamoulis, G., Koubarakis, M.: Plato: a semantic data cube implementation using ontology-based data access technologies. In: Proceedings of the Conference on Big Data from Space (BiDS) (2023). https://cgi.di.uoa.gr/~koubarak/publications/2023/Plato-BiDS2023.pdf
10. Bilidas, D., Mantas, A., Yfantis, F., Stamoulis, G., Koubarakis, M.: The Semantic Data Cube System Plato and Its Applications. In: Proceedings of the IEEE International Geoscience and Remote Sensing Symposium (IGARSS) (2024). https://cgi.di.uoa.gr/~koubarak/publications/2024/plato_igrass.pdf
11. Bilidas, D., et al.: Fire risk management using data cubes, machine learning and OBDA systems. In: Proceedings of SIGSPATIAL, pp. 1–4. ACM (2023). https://doi.org/10.1145/3589132.3625615
12. Calvanese, D., et al.: Ontop: answering SPARQL queries over relational databases. Semantic Web J. (2017). https://doi.org/10.3233/SW-160217
13. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF mapping language. W3C Recommendation, World Wide Web Consortium (2012). http://www.w3.org/TR/r2rml/
14. Gennari, J.H., et al.: The evolution of Protégé: an environment for knowledge-based systems development. Int. J. Hum.-Comput. Stud. (2003). https://doi.org/10.1016/S1071-5819(02)00127-1
15. Ghosh, A., Simkus, M., Calvanese, D.: Semantic querying of integrated raster and relational data: a virtual knowledge graph approach. In: Proceedings of the 17th International Rule Challenge and 7th Doctoral Consortium at RuleML+RR. CEUR Workshop Proceedings, vol. 3485. CEUR-WS.org (2023). https://ceur-ws.org/Vol-3485/paper8240.pdf
16. Hamdani, Y., Xiao, G., Ding, L., Calvanese, D.: An ontology-based framework for geospatial integration and querying of raster data cube using virtual knowledge graphs. ISPRS Int. J. Geo-Inf. **12**(9), 375 (2023). https://doi.org/10.3390/ijgi12090375
17. Harris, S., Seaborne, A.: SPARQL 1.1 query language. W3C Recommendation, World Wide Web Consortium (2013). http://www.w3.org/TR/sparql11-query

18. Lewis, A., et al.: Rapid, high-resolution detection of environmental change over continental scales from satellite data - the earth observation data cube. Int. J. Digit. Earth (2016). https://doi.org/10.1080/17538947.2015.1111952
19. Lu, M., Appel, M., Pebesma, E.: Multidimensional arrays for analysing geoscientific data. ISPRS Int. J. Geo-Inf. **78**(8), 313 (2018). https://doi.org/10.3390/ijgi7080313
20. Marathe, A.P., Salem, K.: Query processing techniques for arrays. Very Large Database J. 68–91 (2002). https://doi.org/10.1007/s007780200062
21. Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 Web Ontology Language Profiles (Second Edition). W3C Recommendation, World Wide Web Consortium, December 2012. http://www.w3.org/TR/owl2-profiles/
22. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. J. Data Semantics **10**, 133–173 (2008). https://doi.org/10.1007/978-3-540-77688-8_5
23. Quartulli, M., Olaizola, I.G.: A review of EO image information mining. ISPRS J. Photogrammetry Remote Sens. **75**, 11–28 (2013). https://doi.org/10.1016/j.isprsjprs.2012.09.010
24. Ray, S.: Quantities, Units, Dimensions and Types (QUDT). Technical Report, Fairsharing.org (2011). https://doi.org/10.25504/FAIRsharing.d3pqw7
25. Rusu, F.: Multidimensional array data management. Foundations Trends Databases **12**(1–3), 69–220 (2023). https://doi.org/10.1561/1900000069
26. Schreiber, G., Raimond, Y.: RDF 1.1 Primer. W3C Working Group Note, World Wide Web Consortium, June 2014. http://www.w3.org/TR/rdf11-primer/
27. Sheth, A.P., Larson, J.A.: Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Comput. Surv. **22**(3), 183–236 (1990). https://doi.org/10.1145/96602.96604
28. Spanos, D.E., Stavrou, P., Mitrou, N.: Bringing relational databases into the semantic web: a survey. Semantic Web J. **3**(2), 169–209 (2012). https://doi.org/10.3233/SW-2011-0055
29. Stonebraker, M., Brown, P., Poliakov, A., Raman, S.: The architecture of SciDB. In: Proceedings of SSDBM (2011). https://doi.org/10.1007/978-3-642-22351-8_1
30. Tan, Z., Yue, P., Gong, J.: An array database approach for earth observation data management and processing. ISPRS Int. J. Geo-Inf. **6**(7) (2017). https://doi.org/10.3390/ijgi6070220
31. Wan, Z., Hook, S., Hulley, G.: MODIS/terra land surface temperature daily L3 global 1km grid V061 (2021). https://doi.org/10.5067/MODIS/MOD11A1.061
32. Warmerdam, F.: The geospatial data abstraction library. In: Hall, G.B., Leahy, M.G. (eds.) Open Source Approaches in Spatial Data Handling. Advances in Geographic Information Science, vol. 2, pp. 87–104. Springer, Berlin (2008). https://doi.org/10.1007/978-3-540-74831-1_5
33. Xiao, G., et al.: Ontology-based data access: a survey. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI) (2018). https://doi.org/10.24963/ijcai.2018/777
34. Xiao, G., et al.: The virtual knowledge graph system ontop. In: Pan, J.Z., et al. (eds.) ISWC 2020. LNCS, vol. 12507, pp. 259–277. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-62466-8_17