GeoSPARQL Best Practice: A GeoSPARQL-enabled triple store as a backend of an OGC web service

This document presents best practices for the creation of Linked Data backends to OGC webservices.

C.1 Accessing spatial Features in a SPARQL endpoint

Spatial *Features* accessed via SPARQL endpoints [SPARQLPROT] are, as defined in the GeoSPARQL standard, instances of geo:Feature or of subclasses of it. They may include one or more geo:hasGeometry relations and further properties related to the Feature and may be grouped into geo:FeatureCollection instances where `geo:FeatureCollection` is a new class in GeoSPARQL 1.1 specifically for the description of collections of `geo:Feature` instances.

The following example SPARQL query retrieves all Feature Collections of a given SPARQL endpoint. A third-party webservice may convert the query results to one or many data formats.

```
SELECT ?fcollection ?item ?rel ?val ?geom
WHERE {
  ?fcollection rdf:type geo:FeatureCollection .
  ?item rdfs:subClassOf* geo:Feature .
  ?item dcterms:isPartOf ?fcollection .
  ?item ?rel ?val .
  OPTIONAL {?item geo:hasGeometry/geo:hasSerialization ?geom}
}
```

Similarly, for SQL backends of OGC webservices, geospatial *Features* are made explicit by defining an SQL query retrieving a set of triples which may be converted to a geodata format.

The following example SPARQL query retrieves a collection of Feature instances (each `item`) and, optionally, their serialized geometries (`geom`):

```
SELECT ?item ?rel ?val ?geom
WHERE {
  ?item rdfs:subClassOf* geo:Feature .
  ?item ?rel ?val .
  OPTIONAL {?item geo:hasGeometry/geo:hasSerialization ?geom}
}
```

This SPARQL query retrieves all Features' properties and their values directly connected to an instance of all classes which are subclasses of geo:Feature. The query result is a list of triples which may be converted to a `geo:FeatureCollection` instance.

`geo:FeatureCollection` matches the OGC API Features's [OGCAPIF] concept of a FeatureCollection and thus may be exposed using an OGC API Features webservice.

However, this may not deliver the expected result in all cases. There are several reasons for this:

- More information for the `geo:FeatureCollection` instance may be needed, in particular, an identifier and perhaps a label for it. If the back-end data store also contains information for the `geo:FeatureCollection` instance then this may be queried for to solve this issue. If not, the API might need to create such data
- Maybe only data in a certain namespace is of interest. The solution is to apply FILTER expressions to the SPARQL query
- The above query only queries directly connected properties and their values. Often geodata formats like GML are hierarchical and therefore need further resolution of property values depicted as URIs. If that is the case the query needs to be adjusted.

C.2 Mappings from CQL2 statements to GeoSPARQL queries

This section presents lists of equivalences between Common Query Language (CQL2) [CQLDEF] statements and GeoSPARQL statements.

C2.1 Query Parameters

Several query parameters may be given as parameters to the HTTP request of the OGC API Features service. These parameters have an influence on the SPARQL query to be executed for the retrieval of a FeatureCollection to be exposed using an OGC API Features service.

Query Parameter	Example	SPARQL Expression	Example	Comment
limit	limit=5	LIMIT	LIMIT 5	
offset	offset=10	OFFSET	OFFSET 10	
bbox	bbox=160.6,-55.95,-170,-25.89	FILTER(geo:sfIntersects())	FILTER(geo:sfIntersects(?geom,"POLYGON160.6 -55.95,160.6 -25.89, -170 -25.89, -170 -55.95, 160.6 -55.95"^^geo:wktLiteral))	WKT does not define a type boundingbox, therefore a bbox is converted to a Polygon
datetime	datetime=2018-02-12T23%3A20%3A52Z			

C2.2 Literal Values

CQL2 defines literal values for a variety of datatypes. The following table shows the equivalences of these values in RDF which may be used in any GeoSPARQL query.

CQL2 literal	Examples	(Geo)SPARQL literal	Examples
String	"This is a string"	xsd:string	"This is a string"^^xsd:string

Number	-100 3.14159	xsd:int xsd:integer xsd:double	"-100"^^xsd:integer "3.14159"^^xsd:double
Boolean	true false	xsd:boolean	"true"^^xsd:boolean "false"^^xsd:boolean
Spatial Geometry (WKT)	POINT(1 1)	geo:wktLiteral	"POINT(1 1)"^^geo:wktLiteral
Spatial Geometry (JSON)	{"type": "Point", "coordinates":[1,1]}	geo:geoJSONLiteral	"{"type": "Point", "coordinates":[1,1]}"^^geo:geoJSONLiteral
Temporal Literal	1969-07-20 1969-07-20T20:17:40Z	xsd:date xsd:dateTime	"1969-07-20"^^xsd:date "1969-07-20T20:17:40Z"^^xsd:dateTime

C2.3 Property references

CQL2 allows to reference properties in the Feature Collection it is targeting for filtering. A property reference is converted to a triple pattern as shown in the following example. A SPARQL variable ?item is assumed to represent the Feature Collection.

Property Reference	Triple pattern
name="OGC"	?item my:name "OGC"^^xsd:string
number=5	?item my:number "5"^^xsd:integer
number>5	?item my:number ?number . FILTER(?number>5)

C2.4 Comparison Predicates

CQL2 defines comparison predicates to compare two scalar expressions. A comparison predicate is converted to a triple pattern as shown in the following example. A SPARQL variable ?item is assumed to represent the Feature Collection.

Comparison predicate	Triple pattern	Comment
name="OGC"	?item my:name "OGC"^^xsd:string	Equality statements can be converted to a triple pattern
number=5	?item my:number "5"^^xsd:integer	
number>5	?item my:number ?number . FILTER(?number>5)	Arithmetic comparisons (<,>,>=,⇐) are converted to filter expressions
number BETWEEN 5 AND 10	?item my:number ?number . FILTER(?number>=5 && ?number⇐10)	BETWEEN statements are converted to arithmetic expressions

name IN ("OGC","W3C")	VALUES ?namevalues {"OGC" "W3C"} ?item my:name ?namevalues .	IN statements are expressed by VALUES statements
name IS NOT NULL	EXISTS {?item my:name ?name }	NOT NULL statements are converted to EXIST statements
name LIKE "OGC."	?item my:name ?name . FILTER(regex(?name, "OGC.", "i"))	LIKE statements are converted to SPARQL regex filters
INTERSECTS(geometry1, geometry2)	FILTER(geof:sfIntersects(?geometry1,?geometry2))	The INTERSECTS filter statement is converted to a GeoSPARQL FILTER statement

- Filter CRS parameter equivalent does not yet exist in GeoSPARQL?

C2.5 Spatial Operators

GeoSPARQL includes equivalents of many CQL2 filter functions as can be seen in the table below.

CQL2 Filter Expression	GeoSPARQL Filter Function
CONTAINS(geometry1,geometry2)	FILTER(geof:sfContains(?geometry1,?geometry2))
CROSSES(geometry1,geometry2)	FILTER(geof:sfCrosses(?geometry1,?geometry2))
DISJOINT(geometry1,geometry2)	FILTER(geof:sfDisjoint(?geometry1,?geometry2))
EQUALS(geometry1,geometry2)	FILTER(geof:sfEquals(?geometry1,?geometry2))
INTERSECTS(geometry1,geometry2)	FILTER(geof:sfIntersects(?geometry1,?geometry2))
OVERLAPS(geometry1,geometry2)	FILTER(geof:sfOverlaps(?geometry1,?geometry2))
TOUCHES(geometry1,geometry2)	FILTER(geof:sfTouches(?geometry1,?geometry2))
WITHIN(geometry1,geometry2)	FILTER(geof:sfWithin(?geometry1,?geometry2))

C2.6 Temporal Operators

Temporal operators are not part of the GeoSPARQL standard.

CQL2 Filter Expression	GeoSPARQL Filter Function
beginTime AFTER 1969-07-16T13:32:00Z	N/A
beginTime BEFORE 1969-07-16T13:32:00Z	N/A
beginTime BEGINS 1969-07-16T13:32:00Z	N/A
beginTime BEGUNBY 1969-07-16T13:32:00Z	N/A
beginTime DURING 1969-07-16T13:32:00Z	N/A

beginTime ENDEDBY 1969-07-16T13:32:00Z	N/A
beginTime ENDS 1969-07-16T13:32:00Z	N/A
beginTime MEETS 1969-07-16T13:32:00Z	N/A
beginTime METBY 1969-07-16T13:32:00Z	N/A
beginTime OVERLAPPEDBY 1969-07-16T13:32:00Z	N/A
beginTime TCONTAINS 1969-07-16T13:32:00Z	N/A
beginTime TEQUALS 1969-07-16T13:32:00Z	N/A
beginTime TOVERLAPS 1969-07-16T13:32:00Z	N/A

C.3 Mappings from Simple Features for SQL

The following table maps the functions and properties from Simple Features for SQL [ISO19125-1] to GeoSPARQL.

Simple Features for SQL	GeoSPARQL Equivalent	Since GeoSPARQL	Related Property Available	Since GeoSPARQL
2.1.1.1 Basic Methods on Geometry				
Dimension(): Double	geof:dimension	-	geo:dimension	1.0
GeometryType(): Integer	Class of geometry instance	1.0	N/A	-
SRID(): Integer	geof:getSRID	1.0	N/A	-
Envelope(): Geometry	geof:envelope	1.0	geo:hasBoundingBox	1.1
AsText(): String	geof:asWKT	1.1	geo:asWKT	1.0
AsBinary(): Binary	N/A	-	N/A	-
IsEmpty(): Integer	geof:isEmpty	-	geo:IsEmpty	1.0
IsSimple(): Integer	geof:isSimple	-	geo:IsSimple	1.0
Boundary(): Geometry	geof:boundary	1.0	N/A	-
2.1.1.2 Spatial Relations				
Equals(anotherGeometry: Geometry): Integer	geof:sfEquals	1.0	geo:sfEquals	1.0

Disjoint(anotherGeometry: Geometry): Integer	geof:sfDisjoint	1.0	geo:sfDisjoint	1.0
Intersects(anotherGeometry: Geometry): Integer	geof:sfIntersects	1.0	geo:sfIntersects	1.0
Touches(anotherGeometry: Geometry): Integer	geof:sfTouches	1.0	geo:sfTouches	1.0
Crosses(anotherGeometry: Geometry): Integer	geof:sfCrosses	1.0	geo:sfCrosses	1.0
Within(anotherGeometry: Geometry): Integer	geof:sfWithin	1.0	geo:sfWithin	1.0
Contains(anotherGeometry: Geometry): Integer	geof:sfContains	1.0	geo:sfContains	1.0
Overlaps(anotherGeometry: Geometry): Integer	geof:sfOverlaps	1.0	geo:sfOverlaps	1.0
Relate(anotherGeometry: Geometry, IntersectionPatternMatrix: String): Integer	geof:relate	1.0	N/A	-
2.1.1.3 Spatial Analysis				
Buffer(distance: Double): Geometry	geof:buffer	1.0	N/A	-
ConvexHull(): Geometry	geof:convexHull	1.0	N/A	-
Intersection(anotherGeometry: Geometry): Geometry	geof:intersection	1.0	N/A	-

Union(anotherGeometry: Geometry): Geometry	geof:union	1.0	N/A	-
Difference(anotherGeometry: Geometry): Geometry	geof:difference	1.0	N/A	-
SymDifference(anotherGeometry: Geometry): Geometry	geof:symDifference	1.0	N/A	-
2.1.2.1 GeometryCollection				
NumGeometries(): Integer	geof:numGeometries	-	N/A	-
GeometryN(N: Integer): Geometry	geof:geometryN	-	N/A	-
2.1.3.1 Point				
X(): Double	N/A	-	N/A	-
Y(): Double	N/A	-	N/A	-
Z(): Double (not in the SQL spec, but a logical extension)	N/A	-	N/A	-
M(): Double (not in the SQL spec, but a logical extension)	N/A	-	N/A	-
2.1.5.1 Curve				
Length(): Double	geof:length	-	geo:hasLength	1.1
StartPoint(): Point	N/A	-	N/A	-
EndPoint(): Point	N/A	-	N/A	-
IsClosed(): Integer	N/A	-	N/A	-
IsRing(): Integer	N/A	-	N/A	-
2.1.6.1 LineString				
NumPoints(): Integer	N/A	-	N/A	-
PointN(N: Integer): Point	N/A	-	N/A	-
2.1.7.1 MultiCurve				

IsClosed(): Integer	N/A	-	N/A	-
Length(): Double	geof:length	-	geo:hasLength	1.1
2.1.9.1 Surface				
Area(): Double	geof:area	-	geo:hasArea	1.1
Centroid(): Point	geof:centroid	1.1	geo:hasCentroid	1.1
PointOnSurface(): Point	N/A	-	N/A	-
2.1.10.1 Polygon				
ExteriorRing(): LineString	N/A	-	N/A	-
NumInteriorRing(): Integer	N/A	-	N/A	-
InteriorRingN(N: Integer): LineString	N/A	-	N/A	-
2.1.11.1 MultiSurface				
Area(): Double	geof:area	-	geo:hasArea	1.1
Centroid(): Point	geof:centroid	1.1	geo:hasCentroid	1.1
PointOnSurface(): Point	N/A	-	N/A	-

References

- OGC API - Features - Part 3: Filtering and the Common Query Language (CQL2) https://docs.ogc.org/DRAFTS/19-079r1.html
- Clementini, Eliseo; Di Felice, Paolino and van Oosterom, Peter, *A small set of formal topological relationships suitable for end-user interaction* (1993). In Abel, David; Ooi, Beng Chin (eds.). Advances in Spatial Databases: Third International Symposium, SSD '93 Singapore, June 23–25, 1993 Proceedings. Lecture Notes in Computer Science. 692/1993. Springer. pp. 277–295. doi:10.1007/3-540-56869-7_16.
- International Organization for Standardization, *ISO 19125-1: Geographic information — Simple feature access — Part 1: Common architecture*
- Open Geospatial Consortium, *OGC API - Features - Part 1: Core*. OGC Implementation Standard (14 October 2019). http://www.opengis.net/doc/IS/ogcapi-features-1/1.0
- World Wide Web Consortium, *SPARQL 1.1 Protocol*, W3C Recommendation (21 March 2013)> http://www.w3.org/TR/sparql11-protocol/