A Foundation for Spatial Data Warehouses on the Semantic Web

Editor(s): Name Surname, University, Country
Solicited review(s): Name Surname, University, Country
Open review(s): Name Surname, University, Country

Nurefşan Gür\textsuperscript{a,b,}*\textsuperscript{,}, Torben Bach Pedersen\textsuperscript{a}, Esteban Zimányi\textsuperscript{b} and Katja Hose\textsuperscript{a}

\textsuperscript{a} Center for Data Intensive Systems, Aalborg University, Selma Lagerløfsvej 300, DK-9220 Aalborg Ø, Denmark
E-mail: \{nurefsan,tbp,khose\}@cs.aau.dk
\textsuperscript{b} Department of Computer and Decision Engineering, Université Libre de Bruxelles, Avenue F. D. Roosevelt 50, B-1050 Brussels, Belgium
E-mail: \{nurefsan.gur,ezimanyi\}@ulb.ac.be

Abstract. Large volumes of geospatial data is being published on the Semantic Web (SW), yielding a need for advanced analysis of such data. However, existing SW technologies only support advanced analytical concepts such as multidimensional (MD) data warehouses and Online Analytical Processing (OLAP) over non-spatial SW data. To remedy this need, this paper presents the QB4SOLAP vocabulary which supports spatially enhanced MD data cubes over RDF data. The paper also defines a number of Spatial OLAP (SOLAP) operators over QB4SOLAP cubes and provides algorithms for generating spatially extended SPARQL queries from the SOLAP operators. The proposals are validated by applying them to a realistic use case.

Keywords: Spatial OLAP, Spatial data, Multidimensional data, Data modelling, RDF, SPARQL

1. Introduction

The Semantic Web (SW) has evolved, from focusing mostly on data publishing to also support increasingly complex queries such as interactive analytical queries. Simultaneously, the data available on the SW has evolved from being simple, most alphanumeric data, to also include complex data such as spatial data. Indeed, geospatial data is now common on the SW, but it remains difficult to analyze it.

In a non-SW context, the main tools for interactive data analyses have been Data Warehouses (DWs) and Online Analytical Processing (OLAP) tools and queries. DWs store large volumes of data and are designed with a multidimensional (MD) modeling approach which has shown itself to be intuitive for interactive data analytics. Concretely, DWs consist of MD data cubes. The cells of the cube represent the topic of analysis, and associate observation facts with numerical measures that can be aggregated. For example, a sales fact cube has measures such as QuantitySold and SalesPrice. Facts are linked to dimensions which provide contextual information, e.g., sales date, product, and location. Dimensions are perspectives which are used to analyze data, and are organized into hierarchies with levels, e.g., Store, City, and Region, that allow users to analyze and aggregate measures at different levels of detail. Levels have a set of attributes that describe the characteristics of the level members.

In traditional DWs, the location dimension is widely used, but as a conventional dimension with alphanumeric data and thus only nominal reference to spatial concepts such as areas and places. This does not allow manipulating true spatial location data or deriving topological relations among the hierarchy levels of the location dimension. This yields a demand for

*Corresponding author. E-mail: nurefsan@cs.aau.dk.
truly spatial DWs for better analysis purposes. Including the geometric information of the location data significantly improves the analysis process (i.e., proximity analysis of the locations) with additional perspectives by revealing dynamic spatial hierarchy levels and new spatial members.

Similarly, providing deep spatial analytics support for spatial SW data is very valuable. Spatial data requires specific treatment techniques, in particular encoding, special functions and different manipulation methods, which should be considered in the modeling process and querying. The current state of the art for the geospatial Semantic Web focuses on techniques for publishing, linking and querying spatial data, but supports only “plain” spatial SW data (without support for spatial DW concepts such as spatial hierarchies, levels, and measures) and does not consider analytical queries over spatial RDF data. (see Section 6 for details).

Contributions. In order to address these issues, this paper makes a number of contributions. First, we propose QB4SOLAP, a generic and extensible vocabulary (metamodel) for spatial DWs on the SW. QB4SOLAP extends the most recent stable version of the QB4OLAP vocabulary with spatial concepts. We provide a full formalization of QB4SOLAP. The key concepts of spatial cube members, spatial hierarchies and levels, spatial measures, spatial aggregate functions (e.g., union, buffer, and convex–hull) and topological relations among spatial dimension and hierarchy level members (e.g., within, intersects, and overlaps), are defined. Second, we define a number of analytical Spatial OLAP (SOLAP) operators over the model including giving formal semantics of the operators. The operators support advanced analytical queries over MD geospatial SW data. Third, we provide algorithms for generation spatially extended SPARQL queries for individual and nested SOLAP operators, which allows writing SOLAP queries without knowledge of RDF/SPARQL. Fourth, we validate the vocabulary, operators, and query generation algorithms by applying them to a realistic use case. This paper extends a previous conference paper [10] very significantly by completely revising the QB4SOLAP formalization, adding the formal semantics of the SOLAP operators, and providing the SPARQL query generation algorithms. Section 6 discussed related work. Finally, Section 7 concludes the paper and points to future research.

2. Preliminary concepts

In this section, we describe the spatial objects and the spatial operations that manipulate them. Then, we introduce the data cubes and spatial enhancement on them as spatial data cubes. Finally, we show the traditional OLAP operations, which manipulate data cubes, and explain the Spatial OLAP (SOLAP) operators, which manipulate spatial data cubes.

2.1. Spatial objects

A spatial object represents a real-world object whose geographic features are important for an application. These geographic features are encoded using the geometry data type. Point, Line, and Polygon are the basic instantiable types of the geometry data type. Coordinates for geometry data type are generally given in 2-dimensions with X, Y values. Geometries are associated with a spatial reference system (SRS), which describes the coordinate space in which the geometry is defined. There are several SRSs and each of them are identified with a spatial reference system identifier (SRID). The World Geodetic System (WGS) is the most well-known SRS and the latest version is called WGS84, which is also used in our use case.

2.2. Spatial operations

There is a set of spatial operations that can be applied on spatial data. We grouped these operations into classes, based on the common functionality of the operators. These classes are defined next.

Definition 1. (Spatial aggregation) The operators in the spatial aggregation class $S_{agg}$ aggregate two or more spatial objects and return a new spatial object. Union, Intersection, Buffer, ConvexHull, and MinimumBoundingRectangle (MBR) are example operators of this class.

Definition 2. (Topological relations) The operators in the topological relation class $T_{rel}$ are commonly ex-
pressed in the RCC8¹ and DE-9DIM² models. Topological relations are Boolean predicates that specify how two spatial objects are related to each other. Examples of topological relations are Intersects, Disjoint, Equals, Overlaps, Contains, Within, Touches, Covers, CoveredBy, and Crosses.

**Definition 3.** (Numeric operations) The operators in the numeric operation class \( N_{\text{op}} \) take one or more spatial objects and return a numeric value. Perimeter, Area, NoOfInteriorRings, Distance, HaversineDistance, NearestNeighbor (NN), and NoOfGeometries are example operators of this class.

### 2.3. Data cubes

Data warehouses store large volumes of data for decision support. They are based on the multidimensional model, which views data in an \( n \)-dimensional space, usually called a data cube. The cells of the cube represent the observation facts for analysis with a set of attributes called measures (e.g., a sales fact cube with measures product quantity and price). Facts are linked to dimensions, which provide perspectives to analyze data (e.g., sales date, product, and customer location). Dimensions are organized into hierarchies, which allow users to aggregate measures at various levels of detail. Hierarchies are composed of levels and there is always a unique top level All with just one member all.

¹RCC8 (Region Connection Calculus) describes regions in Euclidean space or in a topological space by their possible relations to each other.
²DE-9DIM (Dimensionally Extended Nine-Intersection Model) is a topological model that describes spatial relations of two geometries in two dimensions.

Levels have a set of attributes that describe the characteristics of the level members.

An example of a data cube with three dimensions (Customer, Time, and Product) and one measure (Quantity) is given in Fig. 1. Each cell in the cube is an observation fact, which is characterized by dimension and measure values. The hierarchies of this cube are given Fig. 3a–c. Thus, in the cube shown in Fig. 1, the Product dimension is given at the Category level, the Time dimension at the Quarter level, and the Customer dimension at the City level. Measure values represent the measure Quantity of the sold products.

### 2.4. Spatial data cubes

A spatial data cube contains both conventional and spatial dimensions. A spatial dimension is a dimension which includes at least one spatial level in which the application should store the spatial characteristics of the members. Similarly, a hierarchy is a spatial hierarchy if it has at least one spatial level. Spatial characteristics of the levels are captured by their geometries and can be recorded in the spatial attributes of the level. A spatial fact is a fact that relates several dimensions in which, two or more are spatial.

For example, consider a Sales spatial fact, which has spatial dimensions Customer and Supplier, each with a spatial hierarchy Geography composed of spatial levels City → State → Country → All (Fig. 3c). These spatial levels record the spatial characteristics of its members with spatial attributes: Customer, Supplier, and City using a point spatial data type, whereas State and Country with a multi-polygon spatial data type. Spatial data cubes typically have spatial measures, which are also represented by a spatial data type.
An example is a SalesPoint measure that stores the location of sales. Fig. 6 shows the multidimensional schema of the GeoNorthwind data warehouse, which is used as running example in the paper.

2.5. **OLAP operators**

OLAP operators are used for expressing queries over data cubes. The traditional OLAP operators are given next.

The *slice* operator removes a dimension from a cube by selecting one instance in a dimension level. An example is “slice on City is equal to Odense”.

The *dice* operator selects the cells in a cube that satisfy a Boolean condition. An example is “dice on the first and last quarter of the year”.

The *roll-up* operator aggregates measures along a hierarchy to obtain data at a coarser granularity. An example is “roll-up to the Country level” (Fig. 2).

Finally, the *drill-down* operator disaggregates measures along a hierarchy to obtain data at a finer granularity. It is the inverse operation of roll-up. Starting from the cube in Fig. 2, an example is “drill-down to the City level”.

2.6. **Spatial OLAP operators**

Spatial OLAP (SOLAP) operates on spatial data cubes. SOLAP increases the analytical capabilities of OLAP by taking into account the spatial information in the cube. SOLAP operators involve *spatial conditions* or *spatial functions* by using the spatial operators defined in Sect. 2.1. Spatial conditions specify constraints on the geometries associated to cube members or measures, while spatial functions derive new data from the cube, which can be used, e.g., to derive dynamic spatial hierarchies or levels, as explained in the following example. Spatial extensions of the common OLAP operators are formally defined in Sect. 4.

### Table 1

<table>
<thead>
<tr>
<th>Customer</th>
<th>City</th>
<th>Supplier</th>
<th>Total Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>s1</td>
<td>s2</td>
<td>s3</td>
</tr>
<tr>
<td>Düsseldorf</td>
<td>c1</td>
<td>8</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>c2</td>
<td>10</td>
<td>–</td>
</tr>
<tr>
<td>Dortmund</td>
<td>c3</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>c4</td>
<td>–</td>
<td>20</td>
</tr>
<tr>
<td>Münster</td>
<td>c5</td>
<td>–</td>
<td>30</td>
</tr>
</tbody>
</table>

### Table 2

Roll-up of the Sales cube

<table>
<thead>
<tr>
<th>City</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Düsseldorf</td>
<td>21</td>
</tr>
<tr>
<td>Dortmund</td>
<td>34</td>
</tr>
<tr>
<td>Münster</td>
<td>30</td>
</tr>
</tbody>
</table>

### Table 3

S-Roll-up of the Sales cube

<table>
<thead>
<tr>
<th>Supplier</th>
<th>CityClosest</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Düsseldorf</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>Dortmund</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Münster</td>
<td>33</td>
</tr>
</tbody>
</table>

**Example 1.** Consider the summarized data for the Sales cube given in Table 1, where a ‘–’ is used if there are no sales to customers from the corresponding suppliers. The data in Table 1 is shown on the map...
Table 4
Customer to Supplier distance

<table>
<thead>
<tr>
<th>Customer City</th>
<th>Supplier City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Düsseldorf</td>
<td>Dortmund</td>
</tr>
<tr>
<td>s1</td>
<td>s2</td>
</tr>
<tr>
<td>c1</td>
<td>15 km</td>
</tr>
<tr>
<td>c2</td>
<td>15 km</td>
</tr>
<tr>
<td>Dortmünd</td>
<td></td>
</tr>
<tr>
<td>c3</td>
<td>15 km</td>
</tr>
<tr>
<td>c4</td>
<td>45 km</td>
</tr>
<tr>
<td>Münster</td>
<td></td>
</tr>
<tr>
<td>c5</td>
<td>60 km</td>
</tr>
</tbody>
</table>

in Fig. 4, where the arrows on the map between the supplier and customer locations represent the distance. The quantities of sold products are shown along these arrows.

The hierarchies in Fig. 3a–c can be used to perform classical roll-up operations, where measures are aggregated from a child to a parent level. An example of such a roll-up operator is expressed by the query “total sales to customers by city”, whose results is given in Table 2.

On the other hand, as shown in Table 4 and Fig. 4, some customers may be closer to suppliers from other cities. For example, customer c3 is related to its city Dortmund by using traditional Geography hierarchy, but the customer is closer to the city Düsseldorf of supplier s1. Similarly, customer c4 in city Dortmund is closer to the city Münster of supplier s3. Fig. 3d shows a new dynamic spatial hierarchy that can be obtained with a spatial roll-up (s-roll-up) operator that expresses the query “total sales to customers by city of the closest supplier”. Such queries are not possible to express on conventional hierarchies with traditional OLAP.

The hierarchy in Fig. 3d is created on the fly with the help of a spatial function computing the distance between customer and supplier locations. Therefore, using the s-roll-up operator, sales to customers are aggregated by city of the closest suppliers, where Dortmund has a significant drop off in the quantity of the sales from 34 (Table 2) to 20 (Table 3).

3. The QB4SOLAP vocabulary

In this section, we formally define how to represent (spatial) data cubes in RDF. We use as running example the GeoNortwhind data warehouse whose conceptual schema is given in Fig. 6.

The QB4OLAP [8] vocabulary allows to define cube schemas and cube instances as RDF triples. QB4OLAP is an extension of the RDF Data Cube Vocabulary (QB) [5] with multidimensional concepts in order to be able to support OLAP operations directly over RDF data with SPARQL queries. We extended QB4OLAP (v1.2)\(^3\) with spatial concepts to give QB4SOLAP [10]. We based our extension on GeoSPARQL [16], a standard from the Open Geospatial Consortium (OGC) for representing and querying geospatial linked data for the Semantic Web. Fig. 5 shows the QB4SOLAP vocabulary for representing a spatial cube schema and spatial cube members as RDF triples. A cube schema defines the structure of the cube in terms of dimension levels, measures, aggregation functions (e.g., SUM, AVG, COUNT) on measures, spatial aggregation functions (\(S_{\text{agg}}\) in Def. 1) on spatial measures, dimensions hierarchies, and parent–child relationships between levels (including their cardinality and topological relationships for spatial levels). These schema level metadata are used to define multidimensional data sets in RDF. Cube members are the instances of a cube schema that represent level members, facts, and measure values. As we will show in Sect. 5, we use the schema level metadata to produce SPARQL queries that implement SOLAP operators on cube members.

Capitalized terms with non-italic font in Fig. 5 represent RDF classes, capitalized terms with italic font represent RDF instances, and non-capitalized terms represent RDF properties. Classes in external vocabu-

---

\(^3\)QB4OLAP v1.2: https://github.com/lorenae/qb4olap/blob/master/rdf/qb4olap.1.2.ttl
Fig. 5. The QB4SOLAP vocabulary

In what follows, we first define formally RDF triples, and then discuss how to describe (spatial) multidimensional data using QB4OLAP and QB4SOLAP.

Definition 4. (RDF triple) An RDF triple $t = (s, p, o)$ consists of three components: $s$ is the subject, $p$ is the predicate, and $o$ is the object. RDF triples are defined over

$$T = (I \cup B) \times I \times (I \cup B \cup L)$$

where $I$ is the set of IRIs, $B$ is the set of blank nodes, and $L$ is the set of literals. □

A set of RDF triples is referred to as a graph. We denote a QB4SOLAP graph by $\mathcal{G}$, where $\mathcal{G} \subset T$. The cube schema and cube instances are subsets of this graph and are denoted, respectively, by $\mathcal{G}^S$ and $\mathcal{G}^I$, where $\mathcal{G}^S \subset \mathcal{G}$ and $\mathcal{G}^I \subset \mathcal{G}$.

Given an MD element $x \in (I \cup B)$ in a schema graph or instance graph $\mathcal{G}$, we define by $\mathcal{G}[x]$ the subgraph of $\mathcal{G}$ for $x$, where $\mathcal{G}[x] \subset \mathcal{G}$. We define the function $\text{id}(x) : \mathcal{G} \rightarrow I$, that given a MD element $x$ returns its identifier $I$ from the graph $\mathcal{G}$. We use superscript notation to indicate the type of the identifier from the cube schema graph ($\mathcal{G}^S$) and cube instance graph ($\mathcal{G}^I$), e.g., $\text{id}^S(x)$ for a cube schema identifier and $\text{id}^I(x)$ for a cube instance identifier.

---

4RDF cube: http://purl.org/linked-data/cube#
5QB4OLAP: http://purl.org/qb4olap/cubes#
6QB4SOLAP: http://w3id.org/qb4solap#
7GeoSPARQL: http://www.opengis.net/ont/geosparql#
3.1. Defining spatial data cube schemas with QB4SOLAP

An n-dimensional cube schema CS is a tuple \( CS = (D, M, F) \), with a set of dimensions \( D \), a set of measures \( M \), and a fact \( F \). A dimension \( d \in D \) has a set of hierarchies \( H(d) \). Each hierarchy \( h \in H(d) \) is organized into a set of levels \( L(h) \). Each level \( l \in L(h) \) has a set of attributes \( A(l) \). Each attribute \( a \in A(l) \) is defined over a domain. Each measure \( m \in M \) is also defined over a domain.

We define now how to represent a cube schema CS in RDF using QB4SOLAP. We denote the RDF graph of the cube schema \( G^S \). In the examples we prefix the elements of \( G^S \) with gnw:. The subgraph of \( G^S \) that refers to a specific schema element \( x \) is denoted by \( G^S_x \) and the unique identifier of \( x \) is denoted by \( id^S(x) \).

Definition 5. (Dimensions) An n-dimensional cube schema CS has a set of dimensions \( D = \{d_1, \ldots, d_n\} \) and each dimension \( d_i \) has a set of hierarchies \( H(d_i) \) (Def. 6). Each dimension \( d_i \in D \) is defined in the cube schema graph \( G^S \) with \( \text{qb:DimensionProperty} \). Each hierarchy \( h \in H(d_i) \) is linked to its dimension \( d_i \) with the \( \text{qb4o:hasHierarchy} \) property. The RDF graph formulation of the dimensions \( D \) is represented as

\[
G^S_D = \bigcup_{i=1}^{n} G^S_{d_i}
\]

where

\[
G^S_{d_i} = \{(id^S(d_i) \text{ rdf:type } \text{qb:DimensionProperty})\} \cup \bigcup_{l \in L(h_i)} \{(id^S(d_i) \text{ qb4o:hasHierarchy id^S(h_i)})\}
\]

Extension 5. (Spatial dimensions) A dimension is spatial if it has at least one spatial level. A spatial dimension \( d_i \), belongs to the set of spatial dimensions \( D_s \), which is a subset of the set of dimensions \( D \), such that \( d_i \in D_s \subseteq D \).

Example 2. The triples below show how some of the dimensions of the GeoNorthwind DW (Fig. 6) are represented in RDF using Def. 5 and Ext. 5. As we will see below, the Customer and Supplier dimensions are spatial as they both have a spatial hierarchy Geography.

```
# Dimensions
gnw:customerDim rdf:type qb:DimensionProperty ;
qb4o:hasHierarchy gnw:customerGeography .
gnw:productDim rdf:type qb:dimensionProperty ;
qb4o:hasHierarchy gnw:productGeography .
gnw:supplierDim rdf:type qb:DimensionProperty ;
qb4o:hasHierarchy gnw:supplierGeography .
gnw:supplierGeography rdf:type qb4o:Hierarchy .
gnw:employeeDim rdf:type qb:dimensionProperty .

# Hierarchies
gnw:customerGeography rdf:type qb4o:Hierarchy ;
qb4o:inDimension gnw:customerDim ;
qb4o:hasLevel gnw:customer, gnw:country ,
gnw:state, gnw:country .
gnw:supplierGeography rdf:type qb4o:Hierarchy ;
qb4o:inDimension gnw:supplierDim ;
qb4o:hasLevel gnw:supplier, gnw:country ,
gnw:state, gnw:country .
gnw:categories rdf:type qb4o:Hierarchy ;
```
Definition 7. (Levels) A hierarchy $h$ has a set of levels $L(h) = \{l_1, \ldots, l_k\}$ and each level $l_i$ has a set of attributes $A(l_i)$ (Def. 8). Each level $l_i \in L(h)$ is defined in the cube schema graph $G^S_l$ with the qb4o:LevelProperty predicate. Each attribute $a \in A(l_i)$ is linked to its level $l_i$ with the qb4o:hasAttribute property. The RDF graph formulation of the levels $L(h)$ is represented as

$$G^S_{L(h)} = \bigcup_{i=1}^{k} G^S_{l_i}$$

where

$$G^S_{l_i} = \{id^S(l_i) \text{ rdf:type qb4o:LevelProperty}\} \cup \bigcup_{a \in A(l_i)} \{id^S(l_i) \text{ qb4o:hasAttribute id}^S(a)\}$$

Extension 7. (Spatial levels) A level is spatial if it has an associated geometry. A spatial level $l_{is}$ belongs to the set of spatial levels $L_s(h)$, which is a subset of the set of levels $L(h)$, such that $l_{is} \in L_s(h) \subseteq L(h)$. The geometry of a spatial level is defined in the cube schema graph $G^S_l$ with the geo:hasGeometry property. The RDF graph formulation of the spatial levels $L_s(h)$ is represented as

$$G^S_{L_s(h)} = \bigcup_{i=1}^{k} G^S_{l_{is}}$$

where

$$G^S_{l_{is}} = \{id^S(l_{is}) \text{ rdf:type qb4o:LevelProperty}\} \cup \{id^S(l_{is}) \text{ geo:hasGeometry geo:Geometry}\} \cup \bigcup_{a \in A(l_{is})} \{id^S(l_{is}) \text{ qb4o:hasAttribute id}^S(a)\}$$

Example 4. The triples below show how some of the levels of the GeoNorthwind DW (Fig. 6) are repre-
sent in RDF using Def. 7 and Ext. 7. Note that the Customer and City levels are spatial as they have a geometry that is specified at the level definition.

Definition 8. (Attributes) A level \( l \) has a set of attributes \( A(l) = \{a_1, \ldots, a_p\} \), which defines the characteristics of the level members. One among these attributes, denoted as \( a_{ID} \), specifies a surrogate key for the level, i.e., the value of \( a_{ID} \) uniquely identifies the members of the level. For simplicity, we assume that it is the first attribute in the set of attributes \( A(l) \), i.e., \( a_1 = a_{ID} \). Each attribute \( a_i \in A(l) \) is defined in the cube schema graph \( G^S \) with the \( \text{rdfs:range} \) property. Each attribute \( a_i \) is defined as ranging over XSD literals \( \mathcal{L} \) using the \( \text{rdfs:range} \) property. The RDF graph formulation of the attributes \( A(l) \) is represented as

\[
G^S_{A(l)} = \bigcup_{i=1}^{p} G^S_{a_i}
\]

where

\[
G^S_{a_i} = \{(id^S(a_i) \text{rdf:type} \; \text{qb4o:LevelAttribute})\} \cup \{(id^S(a_i) \text{qb4o:inLevel} \; id^S(l))\} \cup \{(id^S(a_i) \text{rdfs:range} \; \mathcal{L})\}
\]

Extension 8. (Spatial attributes) An attribute is spatial if it is defined over a spatial domain. A spatial attribute \( a_{s_i} \) belongs to the set of spatial attributes \( A_s(l) \), which is a subset of the set of attributes \( A(l) \), such that \( a_{s_i} \in A_s(l) \subseteq A(l) \). The RDF graph formulation of the spatial attributes is similar as in Def. 8. However, the attribute must range over spatial literals \( \mathcal{L}_s \) i.e., a well-known text literal (WKT) from OGC schemas. Further, the domain of the attribute should be specified with the \( \text{rdfs:domain} \) property which must be a geometry. Finally, the attribute must be specified as a spatial object with the \( \text{rdfs:subClassOf} \) property.

The RDF graph formulation of the spatial attributes \( A_s(l) \) is represented as

\[
G^S_{A_s(l)} = \bigcup_{i=1}^{p} G^S_{a_{s_i}}
\]

where

\[
G^S_{a_{s_i}} = \{(id^S(a_{s_i}) \text{rdf:type} \; \text{qb4o:LevelAttribute})\} \cup \{(id^S(a_{s_i}) \text{qb4o:inLevel} \; id^S(l))\} \cup \{(id^S(a_{s_i}) \text{rdfs:range} \; \mathcal{L}_s)\}
\]

Example 5. The triples below show how some of the attributes of the GeoNorthwind DW (Fig. 6) are represented in RDF using Def. 8 and Ext. 8. Note that the Customer level has a spatial attribute (Customer geometry). It is represented as a WKT literal which defines a Point type from the Geometry class, which is a subclass of Spatial Object.

### Attributes

\[
\begin{align*}
\text{gnw:customerID} & \text{rdf:type} \; \text{qb4o:LevelAttribute} ; \\
\text{qb4o:inLevel} & \text{rdfs:range} \; \text{gnw:customer} ; \\
\text{rdfs:range} & \text{xsd:Integer} . \\
\text{gnw:customerName} & \text{rdf:type} \; \text{qb4o:LevelAttribute} ; \\
\text{qb4o:inLevel} & \text{rdfs:range} \; \text{gnw:customer} ; \\
\text{rdfs:range} & \text{xsd:Integer} . \\
\text{gnw:postalCode} & \text{rdf:range} \; \text{qb4o:LevelAttribute} ; \\
\text{qb4o:inLevel} & \text{rdfs:range} \; \text{gnw:customer} ; \\
\text{rdfs:range} & \text{xsd:Integer} . \\
\text{gnw:customerGeometry} & \text{rdf:type} \; \text{qb4o:LevelAttribute} ; \\
\text{rdfs:subPropertyOf} & \text{geo:Geometry} ; \\
\text{qb4o:inLevel} & \text{rdfs:range} \; \text{gnw:customer} ; \\
\text{rdfs:range} & \text{geo:wktLiteral} ; \\
\text{rdfs:domain} & \text{geo:Point} ; \\
\text{rdfs:subClassOf} & \text{geo: SpatialObject} .
\end{align*}
\]

Definition 9. (Hierarchy steps) A hierarchy \( h \) has a set of hierarchy steps \( H_S(h) = \{h_1, \ldots, h_n\} \) which define the structure of the hierarchy in relation with its corresponding levels. A hierarchy step \( h_i = (l_i, l_p, \text{card}) \in H_S(h) \) entails a roll-up relation between a lower (child) level \( l_c \) to an upper (par-
Each hierarchy step $h_{s_i}$ is defined in the cube schema graph $G^S$ as a blank node $_i:hs_i \in B$ with the qb4o:HierarchyStep predicate. Each hierarchy step is linked to its hierarchy with the qb4o:inHierarchy property. The child and parent levels are linked in a hierarchy step with the qb4o:inHierarchy property. The child and parent levels are linked in a hierarchy step with the qb4o:parentLevel property. The RDF graph formulation of the hierarchy steps $HS(h)$ is represented as

$$G^S_{HS(h)} = \bigcup_{i=1}^{\#} G^S_{h_{s_i}}$$

where

$$G^S_{h_{s_i}} = \{(_i:hs,\text{rdfs:range } \text{qb4o:HierarchyStep})\} \cup \{(_i:hs,\text{qb4o:inHierarchy id}^{\mathcal{S}}(h))\} \cup \{(_i:hs,\text{qb4o:parentLevel id}^{\mathcal{S}}(l_p))\} \cup \{(_i:hs,\text{qb4o:childLevel id}^{\mathcal{S}}(l_c))\} \cup \{(_i:hs,\text{qb4o:pcCardinality id}^{\mathcal{S}}(\text{card}))\}$$

**Extension 9.** (Spatial hierarchy steps) A hierarchy step is spatial if it relates a spatial child level $l_c$ and a spatial parent level $l_p$, in which case it entails a topological relationships between these spatial levels. A spatial hierarchy step is then a tuple $h_{s_i} = (l_c, l_p, \text{card}, \text{topoRel})$ where the topological relation $\text{topoRel}$ belongs to the $\mathcal{T}_{rel}$ class (Def. 2). The topological relation between parent-child levels of a spatial hierarchy step is defined by the qb4os:pcTopoRel property. The RDF graph formulation of the spatial hierarchy steps $HS_s(h)$ (w.r.t. Def. 9) is represented as

$$G^S_{HS_s(h)} = \bigcup_{i=1}^{\#} G^S_{h_{s_i}}$$

where

$$G^S_{h_{s_i}} = G^S_{h_{s_i}} \cup \{(_i:hs, \text{qb4os:pcTopoRel id}^{\mathcal{S}}(\text{topoRel}))\}$$

**Example 6.** The triples below show how the hierarchy steps of the Geography spatial hierarchy in the Customer dimension of the GeoNorthwind DW (Fig. 6) are represented in RDF using Def. 9 and Ext. 9. Note that all hierarchy steps are spatial and have an associated topological relation.

```rdfs
_:customerGeography_hsl a qb4o:HierarchyStep ;
    qb4o:inHierarchy gwncustomer ;
    qb4o:childLevel gwncustomer ;
    qb4o:parentLevel gwnstate ;
    qb4o:pcCardinality qb4o:ManyToOne ;
    qb4o:pcTopoRel qb4o:Within .

_:customerGeography_hs2 a qb4o:HierarchyStep ;
    qb4o:inHierarchy gwncustomer ;
    qb4o:childLevel gwnstate ;
    qb4o:parentLevel gwncycountry ;
    qb4o:pcCardinality qb4o:ManyToOne ;
    qb4o:pcTopoRel qb4o:Within .

#:Hierarchy steps
_:customerGeography_hs3 a qb4o:HierarchyStep ;
    qb4o:inHierarchy gwncustomer ;
    qb4o:childLevel gwncountry ;
    qb4o:parentLevel gwnstate ;
    qb4o:pcCardinality qb4o:ManyToOne ;
    qb4o:pcTopoRel qb4o:Within .
```

**Definition 10.** (Partial order on levels) The hierarchy steps $HS(h)$ of a hierarchy $h$ define a partial order on the levels $l \in L(h)$. The reflexive and transitive closure of the partial order is denoted as $\sqsubseteq$, with a unique base level ($l_b$) and a unique top level ($All$), where all levels $l$ are such that $l_b \sqsubseteq l$, and $l \subseteq All$.

**Definition 11.** (Measures) An $n$-dimensional cube schema has a set of measures $M = \{m_1, ..., m_r\}$, which record the values of a phenomena being observed. Each measure $m_i \in M$ is defined in the cube schema graph $G^S$ with the qb:MeasureProperty predicate. Similarly to attributes, each measure $m_i$ is defined as ranging over XSD literals $\mathcal{L}$ with the rdfs:range property. The RDF graph formulation of the measures $M$ is represented as

$$G^S_M = \bigcup_{i=1}^{r} G^S_{m_i}$$
where
\[ G^S_{m_i} = \{ (id^S(m_i) \text{ rdf:type qb4o:MeasureProperty}) \} \cup \{ (id^S(m_i) \text{ rdfs:range L}) \} \]

**Extension 11.** (Spatial measures) A measure is spatial if it is defined over a spatial domain as in spatial attributes (Ext. 8). A spatial measure \( m_i \) belongs to the set of spatial measures \( M_s \), which is a subset of the set of measures \( M \), such that \( m_i \in M_s \subseteq M \). RDF formulation of the spatial measures is similar as in Def. 11. However, the domain should range over spatial literals \( L_s \). The RDF graph formulation of the spatial measures \( M_s \) (w.r.t. Def. 11) is represented as
\[ G^S_{M_s} = \bigcup_{i=1}^r G^S_{m_{i_s}} \]
where
\[ G^S_{m_{i_s}} = \{ (id^S(m_{i_s}) \text{ rdf:type qb4o:MeasureProperty}) \} \cup \{ (id^S(m_{i_s}) \text{ rdfs:range L}_s) \} \cup \{ (id^S(m_{i_s}) \text{ rdfs:subClassOf geo:SpatialObject}) \} \]

**Example 7.** The triples below show how the measures of the GeoNorthwind DW (Fig. 6) are represented in RDF using Def. 11 and Ext. 11. Note that SalesPoint is a spatial measure which records the location of the stores in which the sales occurred. It is defined over Geometry domain as a Point type with WKT literal.

```sparql
# Measures
gnw:quantity rdf:type qb:MeasureProperty ;
   rdfs:range xsd:integer .
gnw:unitPrice rdf:type qb:MeasureProperty ;
   rdfs:range xsd:decimal .
gnw:discount rdf:type qb:MeasureProperty ;
   rdfs:range xsd:decimal .
gnw:salesAmount rdf:type qb:MeasureProperty ;
   rdfs:range xsd:decimal .
gnw:freight rdf:type qb:MeasureProperty ;
   rdfs:range xsd:decimal .
gnw:salesPoint rdf:type qb:MeasureProperty ;
   rdfs:domain geo:Point ;
   rdfs:range geo:wktLiteral ;
   rdfs:subClassOf geo:SpatialObject .
```

**Definition 12.** (Fact) In an \( n \)-dimensional cube schema \( CS = (D, M, F) \), the fact \( F \) defines the structure of a cube with the \text{qb:DataStructureDefinition} property. The dimensions are given as components of the fact and are defined with the \text{qb4o:level} property. We assume that the fact \( F \) links the dimensions at the lowest granularity level, therefore \text{qb4o:level} links the lowest (base) level \( l_b \) of each dimension \( d_i \), which is denoted as \( l_b(d_i) \). The cardinality \( \text{card} \) of the relationship between a dimension level and a fact is represented with the \text{qb4o:cardinality} property. Similarly, the measures are given as components of the fact and are defined with the \text{qb4o:measure} property. The aggregate function \text{aggr} associated to each measure is represented with the \text{qb4o:aggregateFunction} property. The RDF graph formulation of the fact \( F \) is given in the following equation.
\[ G^S_F = \{ (id^S(F) \text{ rdf:type qb:DataStructureDefinition}) \} \cup \bigcup_{d_i \in D} \{ (id^S(F) \text{ qb:component}
   \text{qb4o:level id}^S(l_b(d_i));
   \text{qb4o:cardinality id}^S(\text{card}))\} \cup \bigcup_{m_i \in M} \{ (id^S(F) \text{ qb:component}
   \text{qb:measure id}^S(m_i);
   \text{qb4o:aggregateFunction id}^S(\text{aggr}))\} \]

**Extension 12.** (Spatial fact) A fact is spatial if it relates several levels, where two or more are spatial. A spatial fact may also have a topological relation \text{topoRel} that must be satisfied by the related spatial levels, which is represented with \text{qb4so:topologicalRelation}. This object property allows to specify a topological relation in fact-level relationship of spatial facts. The RDF graph formulation of such a fact is simply by adding the property of fact-level topological relation consecutively to the cardinality property as given in the following equation.
3.2. Defining spatial data cube members with QB4SOLAP

We have explained in Sect. 3.1 how a data cube schema can be represented in RDF with QB4SOLAP. We show next how to use this schema to represent the instances of the GeoNorthwind DW (Fig. 6) in RDF.

We denote by $G^I$ the RDF graph of the data cube instances. In the examples, we prefix the elements of $G^I$ with gnwi:. The subgraph of $G^I$ that refers to a specific cube instance $x$ is denoted by $G^I_x$ and the unique identifier of $x$ is denoted by $id^I(x)$.

**Definition 13.** (Level members) A level $l$ has a set of level members $LM(l) = \{lm_1, \ldots, lm_y\}$. Each level member $lm_i$ has a unique IRI $id^I(lm_i) \in I$, which is linked in the cube instance graph $G^I$ with the qb4o:LevelMember predicate. A level member is related to its level by the qb4o:memberOf property. The RDF graph formulation of the level members $LM(l)$ is represented as

$$G^I_{LM(l)} = \bigcup_{i=1}^{y} G^I_{lm_i}$$

where

$$G^I_{lm_i} = \{ (id^I(lm_i) \ rdfs:type \ qb4o:LevelMember) \} \cup \{ (id^I(lm_i) \ rdfs:memberOf \ id^I(l)) \}$$

**Definition 14.** (Attributes of level members) A level member $lm$ has a set of attributes $A(lm) = \{a_1, \ldots, a_p\}$, which are used to describe the characteristics of the level member (Def. 8). Each attribute $a_i$ is linked to the level member with the identifier $id^S(lm)$. We denote by $lm \rightsquigarrow v_{a_i}$ the value $v_{a_i}$ that a level member $lm$ associates to attribute $a_i$. This value is given as a literal $L$ such that $v_{a_i} \in L$. The RDF graph formulation of the attributes $A(lm)$ is represented as

$$G^I_{A(lm)} = \bigcup_{i=1}^{p} G^I_{a_i}$$

where

$$G^I_{a_i} = \{ (id^I(lm) \ rdfs:Property \ v_{a_i}) \mid lm \rightsquigarrow v_{a_i} \}$$

**Definition 15.** (Partial order on level members) A hierarchy step $hs = \{l_r, l_p, \text{card} \}$ between a child level $l_r$ and a parent level $l_p$ defines a set of roll-up relations $RU(hs) = \{r_1, \ldots, r_k\}$ where each $r_i = lm_{r_i} \subseteq lm_{p_i}$ relates a child level member $lm_{r_i} \in LM(l_r)$ to a parent level member $lm_{p_i} \in LM(l_p)$. These roll-up relations define a partial order between level members with regard to Def. 10 and are expressed using the...
property skos:broader. The RDF graph formulation of the roll-up relations $RU(hs)$ is represented as
\[
G^I_{RU(hs)} = \bigcup_{i=1}^{k} G^I_{ri}
\]
where
\[
G^I_{ri} = \{(id(l_{ri}) skos:broader id(l_{pi})) | r_i = \text{id_{cri}} \subseteq \text{id_{pi}} \}
\]

Example 9. The triples below show how some level members of the GeoNorthwind DW (Fig. 6) are represented in RDF using Defs. 13–14.

```
gwni:customer_1 rdf:type qb4o:LevelMember ;
    qb4o:memberOf gnw:customer ;
    gnw:customerID "12209" ;
    gnw:cityName "Berlin" ;
    gnw:postalCode "12209" ;
    gnw:address "Obere Str. 57" ;
    gnw:customerName "Alfreds Futterkiste" ;
    gnw:customerID 1 ;
```

Definition 16. (Fact members) A fact $F$ has a set of fact members $FM(F) = \{f_1, \ldots, f_i\}$, which are the instances of the data cube. Each fact $f_i \in FM$ has a unique IRI $id^F(f_i) \in I$, which is linked in the cube instance graph $G^I$ with the qb:Observation predicate.

A fact member $f_i$ is related to a set of dimension levels $L(f_i) = \{l_1, \ldots, l_i\}$ and has a set of measures $M(f_i) = \{m_1, \ldots, m_s\}$. Each dimension level $l_j$ is linked to the level member with the identifier $id^S(l_j)$ and each measure $m_k$ is linked to the level member with the identifier $id^P(m_k)$. We denote by $f \Rightarrow v_1$ and $f \Rightarrow v_{mk}$, respectively, the dimension values and measure values associated with a fact $f$. The value $v_{1j}$ in $I$ is the identifier of a level member in $LM(l_j)$. Further, the value $v_{mk}$ for every measure $m_k$ is a literal such that $v_{mk} \in L$. The RDF graph formulation of the fact members $FM(F)$ is represented as
\[
G^I_{FM(F)} = \bigcup_{i=1}^{t} G^I_{ri}
\]
where
\[
G^I_{ri} = \{(id(l_{ri}) rdf:type qb:Observation) \cup
\bigcup_{l_j \in L(f_i)} \{(id^S(l_j) id^P(v_{1j}) | f_i \Rightarrow v_{1j})\} \cup
\bigcup_{m_k \in M(f_i)} \{(id^S(l_j) id^P(m_k) id^P(v_{mk}) | f_i \Rightarrow v_{mk})\}
\]

Example 10. The triples below show how a fact member of the GeoNorthwind DW (Fig. 6) is represented in RDF using Defs. 12–16. Note that the fact member and corresponding level members relating to dimensions are given with the prefix gnwi::id$\{a_{1D}\}$ is the surrogate key (Def. 8) that links the fact member to the corresponding dimensions’ base level members.

```
gwni:sale_10613_1 rdf:type qb:Observation ;
    gnw:customerID gnwi:customer_1 ;
    gnw:supplierID gnwi:supplier_6 ;
    gnw:productID gnwi:product_13 ;
    ...;
    gnw:quantity 80;
    gnw:unitPrice "6,00"\"xsd:decimal\";
    gnw:discount "0,10"\"xsd:decimal\";
    gnw:salesPoint "POINT(23.08 42.34)\"geo:wktLiteral\";
```

4. Semantics of SOLAP operators

This section defines a formal algebra for SOLAP operators. Examples of the operators are provided after their definitions. These operators can be applied on spatially enhanced multidimensional data cubes (Sect. 2.3). The presentation defines the semantics of a SOLAP operator by logically specifying the typical OLAP operators with spatial functions and conditions. Spatial functions and conditions can be selected from a range of operation classes, which can be applied on spatial data types (Sect. 2.1). Let $\mathcal{S}$ be the set of any spatial operators where $\mathcal{S} = (\mathcal{S}_{\text{agg}} \cup \mathcal{T}_{\text{rel}} \cup \mathcal{N}_{\text{op}})$, used to represent a spatial predicate $\phi^S \in \mathcal{S}$ or a spatial function $\mathcal{L}^S \in \mathcal{S}$, which is in a SOLAP operator. The following SOLAP operators are defined with a spatial extension to the well-known OLAP operators, which are given in the remarks.

Remark 15. (Slice) The slice operator removes a dimension from a cube $C$ by selecting one instance in a dimension level. For example, the query “slice on customers in the city of Odense” is a slice operation.
(Cube is the sales, dimension is the customer, level is the city and the value is Odense, which is sliced out from the cube).

**Definition 17.** (S-Slice) The s-slice operator removes a dimension from a cube \( C \) by choosing a single spatial attribute value \( v_s \in L_s \) (Ext. 8) in a spatial level \( l_s \) (Ext. 7).

As for the semantics, s-slice takes an \( n \)-dimensional cube \( C \) as an argument. We assume that the cube has the cube schema \( CS = (D, M, F) \), with the fact members \( f \in FM \) as given in Def. 16. As parameters, s-slice takes a spatial literal value \( v_s \), the base level \( l_b \) and the target (spatial) level \( l_s \) of a dimension \( d_i \). The base level \( l_b \) specifies the dimension \( d_i \) (Def. 16). The target spatial level \( l_s \) is the level, that the spatial literal value \( v_s \) is related.

The operator is defined as: \( SS(C)[l_b, l_s, v_s] = C' \), which returns a cube \( C' \) with \( n - 1 \) dimensions and the schema \( CS = (D', M', F') \), where \( D' = D \setminus \{d_i\} \), \( M' = M \), and \( F' = F \). The measures \( M \) and the fact type \( F \) remains the same though the new cube \( C' \) has one dimension less.

The s-slice operator selects a subset \( FM' \) from the set of fact members \( FM \) (\( FM' \subseteq FM \)), with respect to the given parameter \( v_s \). Assuming that the granularity of the fact members are at the (lowest) base level of the dimension \( l_b \in L(d_i) \) in the given cube, a partial order exists among the levels, from bottom level to the target spatial level \( l_s \) such that \( l_b \not\subseteq l_s \). The given parameter \( v_s \) is related to a level member of the level \( l_s \). We say that the fact members are characterized by dimension values, which is written as \( f \sim v_s \) where \( v_d_i \equiv v_{s(d_i)} \) (Def. 16). In other words, dimensions are associated to the fact members by the values of the dimensions’ base level members \( v_{s(d_i)} \). When the dimension \( d_i \) is clear in the context, we will use base level \( v_{d_i} \) for simplicity reasons.

To sum up, the subset \( FM' \) of facts is selected with regards to the partial order on levels from base level \( l_b \) to the target level \( l_s \). The value \( v_l \) in the target level \( l_s \) is specified with respect to the given spatial literal value \( v_s \). The value of \( v_l \) might be equal to a spatial attribute value in the target level \( l_s \), thus \( v_{s(d_i)} \) is characterized by the attribute value \( v_s \) and written as \( v_{s(d_i)} \sim v_s \) (Ex. 11). Or, \( v_{s(d_i)} \) is an arbitrary spatial literal that entails a topological relation \( T_{rel} \) (i.e., within) in a value of the target spatial level \( v_l \), which is written as \( \exists v_{s(d_i)} : \phi^{S}(v_s) \) where \( \phi^{S} \) is a spatial Boolean predicate that represents a topological relation (Ex. 12). After applying the s-slice operator on cube \( C \), the new (sub)set of fact members is defined for both cases respectively as follows; \( FM' = \{ f \in FM \mid \exists v_{s(d_i)} \in L_{s(d_i)} \land \forall v_{s(d_i)} \not\sim v_s \land \phi^{S}(v_s) \} \).
Example 12. With regards to the traditional slice query “slice on customers in the city of Odense”, in this example of s-slice, the user gives a point geometry (i.e., X,Y coordinates of a point as spatial literal) and filter at the given level (i.e., City level) that the given point is within. So the s-slice query would be: “slice on customers of the city, in which the given "POINT(10.43951 55.47006)" is within”.

The following SPARQL query shows an s-slice operator, which filters at the specified level with the given spatial literal by the user.

```
SELECT ?obs WHERE {
    ?obs rdf:type qb:Observation ;
    gnw:customerID ?cust .
    ?cust qb4o:memberOf gnw:customer ;
    skos:broaderof ?city .
    BIND(bif:st_area(?x) as ?area)}
FILTER {?cityGeo = ?x}
```

Note that the s-slice can be operated in different ways based on the geometry given to the query. In both Ex.s 11 and 12, slice level is given as City, however in Ex. 12 a random X, Y point is given that is falling into the target city. Therefore we need to use within from topological relationships (T_{rel}) class in order to verify and filter that city.

Remark 16. (Dice) The traditional dice operator takes a cube and a Boolean condition \(\phi\), which returns a new cube containing only the cells that satisfy the Boolean condition \(\phi\). Dice operation is analogous to relational algebra, \(R\) selection; \(\sigma_{\phi}(R)\), but the argument is a cube not a relation. For example, the query “sales to customers of type LLC (Limited Liability Company)” is a dice operation. (Cube is the sales, dimension is the customer, and Boolean condition is the customer type if they are LLC).

Definition 18. (S-Dice) Similarly, the s-dice operator takes an \(n\)-dimensional cube \(C\) as an argument, which has the cube schema \(CS = (D, M, F)\) with the fact members \(f \in FM\) as given in Def. 16. As a parameter s-dice takes a spatial Boolean predicate which is denoted by \(\phi^S\). The s-dice operator keeps the cells of the cube \(C\) that satisfies the spatial predicate over spatial dimension levels \(l_s\), attributes \(a_s\), and measures \(m\).

The semantics of the operator is defined as:

\[SD(C)[\phi^S] = C'\]

where spatial predicate \(\phi^S\) can be applied on spatial level member values \(\phi^S(v_m)\), spatial attribute values \(\phi^S(v_a)\), measure values \(\phi^S(v_m)\) and/or a combination of these.

\[SD\] operator returns a sub cube \(C' \subseteq C\) which has the schema \(CS = (D', M', F')\) where \(D' = D, M' = M, F' = F\). Unlike the s-slice operator, s-dice keeps all the dimensions \(D\) in the output cube \(C'\). The set of measures \(M\) and the fact type \(F\) also remains the same, though the new cube \(C'\) is a subset of the original cube \(C\) with filtered fact members \(f \in FM'\) which is explained in the following.

The s-dice operator selects a subset \(FM'\) of the fact members’ set \(FM' \subseteq FM\) with respect to the spatial predicate \(\phi^S\) on level members as follows:

1. Spatial predicate on level values: \(FM' = \{ f \in FM \mid \exists v_{l_h} \in LM(l_h), v_{a_s} \in LM(a_s) : f \bowtie v_{l_h} \land v_{a_s} \subseteq v_{l_h} \land \phi^S(v_{a_s}) \}\).

2. Spatial predicate on level attribute values:
\(FM' = \{ f \in FM \mid \exists v_{l_h} \in LM(l_h), v_{a_s} \in LM(a_s) : f \bowtie v_{l_h} \land v_{a_s} \subseteq v_{l_h} \land \phi^S(v_{a_s}) \}\).

Note that the filtering the facts through level members can be done by \(v_{l_h}\) (level values) or attribute values \(v_{a_s}\) by applying the spatial predicate \(\phi^S\). Finally filtering of the facts is on associated measure values is defined in the following:

3. Spatial predicate on measure values of \(m_s\): \(FM' = \{ f \in FM \mid \exists v_{m_s} \in Codomain(m_s) : f \bowtie v_{m_s} \land \phi^S(v_{m_s}) \}\).

For complex cases, i.e., combining these three types; the result set is also followed by combining the basic result sets.

Example 13. The s-dice operator can be implemented on level and attribute values by filtering level members in the cube or on measures by filtering the facts in the cube. In both cases the spatial predicate \(\phi^S\) is used.

The query for the s-dice operator could be “sales to customers which are located within 5 km distance from their city center” where the s-dice is on level members by filtering the customer level. The spatial predicate \(\phi^S\) can be interpreted in two different ways.

1. First method is assuming a buffer area of 5 km from the coordinates of city center and checking customers’ locations by within operator from topological relations \(\phi^S \in T_{rel}\) if it meets the condition. The following SPARQL query shows
the implementation of this method on level members.

SELECT ?obs WHERE {
  ?obs rdf:type qb:Observation ;
  gwn:customerID ?cust .
  ?cust qb4o:memberOf gwn:customer ;
  skos:broader ?city ;
  gwn:customerGeo ?custGeo .
FILTER (bif:st_within (?cityCentGeo, ?custGeo, 5))
}

2. Second method is checking if the distance from a customer location to the corresponding city center is less than 5 km, by using distance function from numeric operations $f^S \in N_{op}$. In this case the spatial predicate $\phi^S$ is a combination of a spatial function $f^S$ and a regular Boolean predicate $\phi$. Spatial function is $distance$ from numeric operations and the predicate is $less\ than\ (<)$. The following SPARQL query shows the implementation of this method for s-dice on level members.

SELECT ?obs WHERE {
  ?obs a qb:Observation ;
  gwn:customerID ?cust .
  ?cust qb4o:memberOf gwn:customer ;
  skos:broader ?city ;
  gwn:customerGeo ?custGeo .
BIND (bif:st_distance (?custGeo, ?cityCentGeo) AS ?distance) FILTER (?distance < 5 )
}

Remark 17. (Roll-up) The traditional roll-up operator aggregates measures according to a dimension hierarchy (by using an aggregate function), in order to obtain measures at a coarser granularity for a given dimension. For example, the query “total amount of sales to customers by city” is a classical roll-up operation. (Cube is the sales, dimension is the customer, level in dimension to roll-up is the city such that $customer \subseteq city$, measure is the sales amount and aggregate function is the sum in order to calculate the total sales.)

Definition 19. (S-Roll-up) Similarly to roll-up operator, s-roll-up aggregates measures $m \in M$ of a given cube $\mathcal{C}$, by using an aggregate function and a spatial function $f^S \in S$ (Sect. 2.1) along a spatial dimension’s hierarchy $h_s$ (Ext. 6), which should have spatial levels $l_s$ (Ext. 7). However, in s-roll-up the dimension hierarchy is created dynamically on levels by the spatial function $f^S$. We call this hierarchy a dynamic spatial hierarchy, conceptually from a base level $l_b$ to the dynamically created target level $l'_s$ such that $l_b \subseteq_d l'_s$. The instances of the target level $l'_s$ are obtained by the spatial function $f^S(l_s)$ that is applied on spatial dimension levels.

As for the semantics, s-roll-up takes an $n$-dimensional cube $\mathcal{C}$ as an argument, which has the cube schema $\mathcal{C}S = (D, M, F)$ with the fact members $f \in FM$ as given in Def. 16. As a parameter s-roll-up takes a spatial function $f^S \in S$ to operate on levels $L(d_i)$ and an aggregate function $agg$ to calculate a measure $m$ at the higher target level. For simplicity of explanation and without loss of generality, we initially assume that there is only one measure $m$. The extension of the operator on several measures $m \in M$ is explained in the last paragraph. S-Roll-up operator is formulated as: $SRU(C)[f^S(L(d)), agg(m)] = C'$, which returns a cube $C'$ with $n$-dimensions and has the schema $CS = (D', M', F')$ where $F' = F, M' = M$, and $D' = \{d_i \in D \mid \{d_1, \ldots, d_i, \ldots, d_n\} \wedge L'(d_i') = L(d_i) \setminus \{l_b \subseteq_d \ldots \subseteq_d l_a \cup \{l'_s\}\}$. After the s-roll-up operation, number of dimensions in $D$ remains the same, although the base levels and levels below the target level $(l_b \subseteq_d \ldots \subseteq_d l_a)$ of the corresponding dimension $d_i$ are left out and a new target level $l'_s$ is added to the set of dimension levels $L'(d'_i)$ of $d'_i$.

The set of level members of the level $l'_s$ is selected with respect to the spatial function on base level members of a spatial dimension such that $LM(l'_s) = \{f^S(v_h) \mid v_h \in LM(l_b)\}$ where $l_b \subseteq_d l'_s \iff f^S(v_h) = v'_l$ which means that the base level $l_b$ rolls up along the spatial dynamic hierarchy $(\subseteq_d)$ to the target new spatial level $l'_s$ if and only if spatial function on base level $f^S(v_h) = v'_l$ produces the new spatial level members $v'_l$. Even though the set of measures $M$ remains the same, the s-roll-up operator obtains the measure values associated with fact members $f'$ at a coarser granularity $l'_s$, which alters the set of facts $FM' \not\subseteq FM$. In order to create the new set of facts $FM'$ at the new granularity level $l'_s$, the Group operator [17] is used to group the facts characterized by the same level members $v_l \in LM(l'_s)$ such that $Group(v'_l) = \{f \in FM \mid \exists v_h \in LM(l_h) : f \sim v_l \wedge v_h \subseteq_d v'_l\}$. The output of the Group operator on level members is a new fact instance $f'$. In order to aggregate the measure values $v_m$, which are associated with the fact members $f$ we use an aggregate function $agg$ such that $agg(\{f_1, \ldots, f_k\}) = agg(v_m_1, \ldots, v_m_k)$ where $f_i \sim v_m, i = 1, \ldots, k$.

Finally, the set of the new facts $f' \in FM'$ is constructed, that is given with the associated new level members and aggregated measure values as: $FM' = \{f' = Group(v'_l) \mid \exists v'_l \in LM(l'_s) : f' \sim v'_l \wedge f' \sim agg(\text{Group}(v'_l))\}$. 
The extension to multiple measures is similar which is done by providing and using a separate aggregate function for each measure $m \in M$.

**Example 14.** The following SPARQL query shows the $s$-roll-up operator which is exemplified in Sect. 2.6. The query is “total amount of sales to customers by city of the closest suppliers”. Note that the measures are aggregated up to a new city from customer level of the customer dimension, which is specified as the Closest City. The hierarchy step from customer to city is defined dynamically by a spatial function $f_S$ (distance from numeric operations $N_{op} \subset S$) which is then used in a wrapper function to find the closest distance of the suppliers and customers. The levels and level members (of customer), which are below the newly defined level (Closest City) are left out in the result.

```
SELECT ?city (SUM(?sales) AS ?totalSales)
WHERE { ?obs a qb:Observation ;
  gw:customerID ?cust ;
?cust qb4o:memberOf gw:customer ;
  gw:customerGeo ?custGeo ;
  gw:customerName ?custName ;
  skos:broader ?city .
?city qb4o:memberOf gw:city .
?s sup gw:supplierGeo ?supGeo .
# Inner Select for the total sales to
# the closest supplier of the customer
{ SELECT ?cust1 (MIN(?distance) AS ?minDistance) AS ?closestSupplier
  gw:customerID ?cust1 ;
  gw:supplierID ?sup1 .
  ?sup1 gw:supplierGeo ?sup1Geo .
  BIND (bif:st_distance( ?cust1Geo, ?sup1Geo ) AS ?distance)
GROUP BY ?cust1 }
FILTER (?distance = ?minDistance)
GROUP BY ?city
```

**Remark 18.** (Drill-down) Drill-down is the inverse operator of roll-up, which disregards previously summarized data to a child level in order to obtain measures at a finer granularity of a given dimension. For example, the roll-up query given in Remark 17 (“total amount of sales to customers by city”) aggregates sales by summing up the sales amount, from customer level to city level along a hierarchy. As drill-down operator performs the operation opposite to the roll-up an example would be: “average amount of sales of each supplier, drilled down from the city level to the supplier level”. (Cube is the same as sales, and the hierarchy is the same but the dimension is the supplier, so child level in dimension to drill-down from city level is the supplier such that $C_{ity} \sqsubseteq Supplier$). Conceptually, a drill-down to level $l_i$ on a cube $C$ corresponds to a roll-up to the same level $l_i$ on the base cube of $C$, that is denoted as $BaseCube(C)$.

**Definition 20.** (S-Drill-down) Analogously to drill-down operator, $s$-drill-down disaggregates measures $m \in M$ of a given cube $C$, by using an aggregate function and a spatial function $f_S$ (Sect. 2.1) along a spatial dimension’s hierarchy $h_s$ (Ext. 6), which should have spatial levels (i.e., $l_s$) (Ext. 7).

Conceptually, in $s$-drill-down, the dimension hierarchy is created dynamically on levels by the spatial function $f_S$ as in $s$-roll-up. This is similar to the dynamic spatial hierarchy defined in Def. 19, that is from a spatial parent level $l_p$ to a dynamically created spatial child level $l'_c$, such that $l_p \sqsubseteq l'_c$. The target spatial child level $l'_c$ is the output of the spatial function $f_S$ on spatial levels $l_i \in L(d_i)$ of the spatial dimension. Applying $s$-drill-down to child level $l'_c$, from a parent level $l_p$, on a cube $C$ corresponds to applying $s$-roll-up to the same level $l'_c$, from the base level $l_0$ on the base cube of $C$. Therefore, the semantics of the $s$-drill-down is described as same as $s$-roll-up and the operator is formulated as $SDD(C)[f_S(L(d_i)), agg(m)] = SRU(BaseCube(C))[f_S(L(d_i)), agg(m)]$.

**Example 15.** In order to exemplify an $s$-drill-down, starting from the result cube graph of Ex. 14 (“total amount of sales to customers by city of the closest supplier”), which is at the granularity of City level, we drill down to child level Supplier with the query “average amount of sales of furthest suppliers to their city center, drilled down from City level to Supplier level”. The following SPARQL query shows the given example.

```
SELECT ?sup (AVG(?sales) AS ?averageSales)
(MIN(?distance) AS ?maxDistance)
WHERE { ?obs a qb:Observation ;
  gw:supplierID ?sup ;
  gw:salesAmount ?sales .
?sup qb4o:memberOf gw:supplier ;
  gw:supplierGeo ?supGeo ;
  gw:supplierName ?supName ;
  skos:broader ?city .
?city qb4o:memberOf gw:city .
FILTER (?distance = ?maxDistance)
GROUP BY ?sup
```

In this paper, we focus on direct querying of single data cubes with main SOLAP operators in SPARQL. The integration of several cubes through s-drill-across or set-oriented operations such as union, intersection, and difference [4] is out of scope and remained as future work.

5. Generating SOLAP queries in SPARQL via QB4SOLAP

After having defined the high-level SOLAP operators in Sect. 4, this section first describes how to generate SPARQL queries for each of these operators by using the QB4SOLAP metamodel (Sect. 3). Afterwards, this section describes how to create more complex SPARQL queries for nested SOLAP operations.

5.1. Generation algorithms

The generated SPARQL queries \( Q \) are of the form "\( Q = \text{SELECT} \ R \text{WHERE} \ GP \)" , where \( GP \) is a graph pattern containing triple patterns and \( R \) is the (set of) variable(s) that are returned in the result of the query. Triple patterns are based on triples of the form \( (s, p, o) \) (Def. 4), where triple components are replaced by variables. A set of triple patterns defines a graph pattern \( GP \). Given an RDF graph \( G \), a graph pattern \( GP \) is used to search for subgraphs \( G(R) \subseteq G \) matching the pattern. In our algorithm, the graph pattern is initially empty, \( GP = \emptyset \), and the triple patterns are added incrementally to the body of the \textit{WHERE} clause: \( GP = GP \cup (s \ p \ o) \).

RDF datasets published with the QB4SOLAP vocabulary use the \texttt{skos:broader} property to define the roll-up relation from child level to parent level (Defs. 13 and 15). As this is the case for all hierarchy levels in a dimension, every OLAP query contains such roll-up paths that we need to consider as part of \( GP \) in the \textit{WHERE} clause.

Thus, we define a helper function \textit{RUPath} (Algorithm 1) that we can use in the SOLAP query generation algorithms.

**Algorithm 1: \textit{RUPath}**

\[ \text{\textit{ RUPath}}(G_s^C, l_b, l_s, a_{ID}, ?a_s, ?f) \]

\begin{itemize}
  \item \textbf{Input:} \( G_s^C, l_b, l_s, a_{ID}, ?a_s, ?f \)
  \item \textbf{Output:} \( GP \)
\end{itemize}

\begin{algorithm}
\begin{algorithmic}
\State \( GP = (\texttt{\{ ?f \texttt{\{} rdf:type \texttt{\{} qb:Observation \texttt{\} \}}}) \)
\State \( GP = GP \cup (?\texttt{\{ fID } skos:broader ?a \texttt{\}}) \)
\ForEach \( (id^S(l_1), id^S(l_2)) \in G_s^C \mid l_1 \subseteq l_2 \)
\State \( GP = GP \cup (?l_1 skos:broader ?l_2 \) \)
\EndFor
\State \( \text{let } GP = GP \cup (?l_s id^S(a_s) ?a_s) \)
\State \( \text{return } GP \)
\end{algorithmic}
\end{algorithm}

For each \( l_b \rightarrow l_s \) in order to represent the level member variables at the target level. The roll-up path starts at the fact instances \( f \) (Def. 16). Afterwards, the partial order on level members (Def. 15) from base level \( l_b \) to target level \( l_s \) is applied. Algorithm 1 sketches the helper function for building the roll-up path for dimensions; from facts to dimension levels with predicates and cube member IRIs defined in the cube schema.

In order to represent such varying parameters at the instance level such as fact members, level members, or parameter values given by the user, and to distinguish these parameters from other parameters in the algorithm, we represent such parameters using variable names with question marks.

We use a \texttt{FILTER} expression to restrict the output data by using a (spatial) Boolean predicate \( \phi^S \). A \texttt{FILTER} expression is part of the \textit{WHERE} clause in a SPARQL query. Therefore, it is added to the body of the \textit{WHERE} clause in the graph pattern \( GP \) as \( GP = GP \cup (\texttt{FILTER} \phi^S) \). In the cases where there is a spatial function \( f^S(x) \) in the SOLAP operator, it is given in the \texttt{BIND} clause, which is technically a part of the \textit{WHERE} clause and therefore added to the body of the \textit{WHERE} clause in a graph pattern \( GP \) as \( GP = GP \cup (\texttt{BIND} f^S(x)) \). SPARQL 1.1 defines
aggregate expressions, such as SUM, MIN, MAX, AVG, etc.

We apply them on measure values or use them as wrappers in spatial functions. In the following, we often write AGG to represent them.

In the following, we present the SPARQL query generation algorithms for the SOLAP operators defined in Sect. 4. The algorithms take the input parameters and arguments of the SOLAP operator and return the a SPARQL query $Q$ that can be executed.

### S-Slice generator
To generate a SPARQL query for the s-slice operator $SS(C)[l_b,l_s,v_s]$ (Def. 17), we use Algorithm 2. Parameter $v_s$ is a spatial literal value $v_s \in \mathcal{L}_s$ (i.e., POINT or POLYGON) that should be related to a spatial level $l_s$ (Ext. 7). This means that $v_s$ is defined as a polygon geometry that corresponds to a spatial attribute value in the target level $l_s$ (Ex. 11) or $v_s$ is defined as a point geometry that is spatially contained in a spatial attribute value of the target level $l_s$ (Ex. 12).

Note that in Ex. 11.1, the given spatial literal has the geometry data type polygon, which corresponds to a spatial level attribute $s_a$ (Ext. 8) at a spatial level $l_s$. Similarly, the spatial function call $f^S(x)$ in Ex. 11.2 returns a polygon that corresponds to a spatial level attribute $s_a$.

On the other hand the given spatial literal in Ex. 12 has the geometry data type point which corresponds to the spatial level $l_s$ via topological relations ($T_{rel}$). We consider all these possibilities in the s-slice generator algorithm. We explained these in the following, where the steps are referencing the line numbers in Algorithm 2.

#### Algorithm 2: S-SliceGenerator ($G^l_s(C), v_s, l_b, l_s$)

**Input:** $G^l_s(C), v_s, l_b, l_s$

**Output:** $Q$

1. **begin**
2. $Q = \emptyset$; $GP = RUPath (G^l_s(C), l_b, l_s, a_{ID}, ?a_s, ?f)$
3. if $v_s$ is a POINT then
4. $Q = GP \cup (\text{FILTER} (\text{st\_within} \ v_s, \ ?a_s))$
5. else if $v_s = f^S(x)$ then
6. $Q' = GP \cup (\text{FILTER} (\text{st\_within} \ v_s, \ ?a_s))$
7. $GP' = \text{FILTER} (\text{WHERE} \ GP')$
8. $Q = GP \cup Q' \cup (\text{FILTER} ?x = ?a_s)$
9. else
10. $GP = GP \cup (\text{FILTER} \ v_s = ?a_s)$
11. **return** $Q = \text{SELECT} ?f$ WHERE $GP$

Line 3. Check if the spatial literal $v_s$ is a point geometry type. If true, create a FILTER statement with a spatial Boolean predicate (Line 4) and go to the result (Line 12).

Line 4. Build the FILTER statement based on the spatial literal $v_s$ and the spatial attribute $a_s$ (Ex. 12). As a result the following lines might be added to the $GP$:

```
FILTER (bif:st\_within("POINT(10.43951 55.47006)", ?cityGeo))
```

Line 5. Check if $v_s$ is a function call $f^S(x)$. If true (Ex. 11.2), construct an inner select query to compute the spatial function $f^S(x)$, then go to the result (Line 12).

Line 6. Call the RUPath function in order to link $a_s$ variables with the fact instances (this time for inner select query $Q'$). This step creates a graph pattern $GP'$ for inner select query $Q'$, for example:

```
(?obs rdf:type qb:Observation ;
  gnw:customerID ?cust .
  ?cust qb4o:memberOf gnw:customer ;
  skos:broader ?city .
  ?city gnw:cityGeo ?cityGeo .)
```

Line 7. Build a bind statement on $a_s$ variables for calculating spatial functions (e.g., compute areas). For example, the following lines might be added to graph pattern $GP'$:
Line 8. Generate the inner select query \( Q' \) based on \( GP' \) generated in Lines 6 and 7. For example (\( Q' \) finds the geometry of the largest city):

\[
Q' = \{ \text{SELECT ?x (MAX(?area) as ?maxArea) WHERE {?obs rdf:type qb:Observation ;
\quad gnw:customerID ?cust .
\quad ?cust qb4o:memberOf gnw:customer ;
\quad skos:broader ?city .
\quad ?city gnw:cityGeo ?x .
\quad BIND (bif:st_area(?x) as ?area) } }
\]

Line 9. Build the filter statement with the output of the spatial function \( f_S(x) \), construct \( GP \) (includes \( Q' \)) for the outer query, and go to the result (Line 12). At this stage \( GP \) is constructed in Lines 2 and 8. The following for the filter statement is added to the \( GP \):

\[
\text{FILTER (?cityGeo = ?x) }
\]

Line 11. If a spatial literal \( v_s \in L \) is given as the parameter instead, build a filter statement that checks if \( v_s \) is equal to the spatial attribute \( a \) values, and go to the result (Line 12). For example, the following filter condition might be added to graph pattern \( GP \):

\[
\text{FILTER (?cityGeo = "POLYGON((10.43951 55.47006, 10.43947 55.47003, 10.43924 (...))")}
\]

S-Slice operator with a spatial function call: The following listing corresponds to the SPARQL output of the running example where the spatial value is returned from a function call (Ex. 11.2). The graph pattern \( GP' \) for the spatial function call is created in Lines 8 to 13. The graph pattern \( GP \) for the whole query is created in Lines 2 to 12.

\[
Q = \{ \text{SELECT ?obs WHERE}
\quad (?obs rdf:type qb:Observation ;
\quad gnw:customerID ?cust .
\quad ?cust qb4o:memberOf gnw:customer ;
\quad skos:broader ?city .
\quad ?city gnw:cityGeo ?x .
\quad BIND (bif:st_area(?x) as ?area) }
\quad \text{FILTER (?cityGeo = ?x) }
\}
\]

S-Slice operator with a given spatial value as point data type: The following listing corresponds to the SPARQL output of the running example where the spatial value is given as a \texttt{POINT} data type (Ex. 11.1) corresponding to a level attribute. The graph pattern \( GP \) for the query is created in Lines 2 to 7.

\[
Q = \{ \text{SELECT ?obs WHERE}
\quad (?obs rdf:type qb:Observation ;
\quad gnw:customerID ?cust .
\quad ?cust qb4o:memberOf gnw:customer ;
\quad skos:broader ?city .
\quad FILTER (?cityGeo = "POLYGON((10.43951 55.47006, 10.43947 55.47003, 10.43924 (...))")}
\}
\]

S-Dice generator. To generate a SPARQL query for the s-dice operator, \( SD(C)(\phi^S) \) (Def. 18 - parameter \( \phi^S \) represents a spatial predicate), we follow the steps sketched in Algorithm 3. The algorithm takes parameter \( \phi^S \) as input, which corresponds to a spatial predicate that could represent a topological relation from the \( T_{rel} \) set or a combination of a spatial function (a numeric operation from the \( N_{op} \) set) and a regular predicate \( \phi \). For illustration, we use the example query that we have introduced in Sect. 4 for s-dice (Ex. 13):

"sales to customers, which are located 5 km distance from their city center". In the following, we discuss the
main steps of Algorithm 3 with the running example, where the steps are referencing the line numbers in Algorithm 3.

Line 3. The algorithm runs through the levels from base level \( l_b \) to the target spatial level \( l_s \), which are both given in the spatial Boolean predicate \( \phi^S \).

Line 4. Build the roll-up path for those levels using the helper function \( \textsc{RUPath} \). Note that, when we apply the roll-up path to the target level, we can also link the level attributes for the target (spatial) level – as, for example, in the last line of the following listing. The output of function \( \textsc{RUPath} \) is added to graph pattern \( GP \):

```scheme
{\text{?obs a qb:Observation ;
  gwn:customerID ?cust .
  ?cust qb4o:memberOf gwn:customer ;
  skos:broader ?city ;
  gwn:customerGeo ?custGeo ,

Q = \emptyset ; GP = \emptyset
for \( l_b, l_s \in \phi^S \) do
  \text{GP} = \text{GP} \cup \text{RUPath}(G^S_{(C)}, l_b, l_s, aID, ?a_s, ?f)
if \( \phi^S \) is a spatial predicate then
  \text{GP} = \text{GP} \cup (\text{FILTER} \phi^S(?a_s))
else if \( \phi^S \) uses a spatial function \( f^S(x) \) and a regular Boolean predicate \( \phi \) then
  \text{GP} = \text{GP} \cup (\text{FILTER} \phi(?x))
return \text{Q} = \text{SELECT ?f WHERE GP}
```

Line 5. Check if \( \phi^S \) is to be implemented as a spatial predicate from topological relations \( T_{rel} \) as interpreted in Ex. 13.1.

Line 6. Create a filter statement with a spatial predicate and the spatial level attribute \( a_s \), which is referenced in the roll-up path (Line 4). For our running example, the filter statement is applied on customers that are located within a buffer area of 5 km from their city centers. The spatial predicate \( \text{st}_\text{within} \) is used from the topological relations. The following lines are added to graph pattern \( GP \):

```scheme
\text{FILTER (bif:st_within(}?custGeo, ?cityCentGeo, 5))\text{)}
```

Line 7. Check if \( \phi^S \) is to be implemented as a combination of a spatial function \( f^S(x) \) and a regular predicate \( \phi \) as interpreted in Ex. 13.2.

Lines 8, 9. Create a bind statement based on a spatial function (i.e., calculate \( \text{st}_\text{distance} \) between customers and city center) and a filter statement based on the assigned values with a regular predicate (i.e., less than 5 km). The following lines are added to graph pattern \( GP \):

```scheme
\text{BIND (bif:st_distance(}?custGeo, ?cityCentGeo AS ?distance) \text{FILTER ( ?distance < 5 ))}
```

Line 10. Generate query \( Q \) for selecting the facts \( f \in FM' \) matching the incrementally created graph pattern \( GP \) in the previous steps. In our running examples we obtain the following cases for the generated \( s\)-dice query \( Q \).

\begin{algorithm}[h]
\caption{S–DiceGenerator \( (G^I_{(C)}, \phi^S) : Q \)}
\begin{algorithmic}
\State \textbf{Input:} \( G^I_{(C)}, \phi^S \)
\State \textbf{Output:} \( Q \)
\State \textbf{begin}
\State \( Q = \emptyset ; GP = \emptyset \)
\For {\( l_b, l_s \in \phi^S \)}
\State \( GP = GP \cup \text{RUPath}(G^S_{(C)}, l_b, l_s, aID, ?a_s, ?f) \)
\If {\( \phi^S \) is a spatial predicate}
\State \( GP = GP \cup (\text{FILTER} \phi^S(?a_s)) \)
\ElseIf {\( \phi^S \) uses a spatial function \( f^S(x) \) and a regular Boolean predicate \( \phi \)}
\State \( GP = GP \cup (\text{FILTER} \phi(?x)) \)
\EndIf
\EndFor
\State \textbf{return} \( \text{Q} = \text{SELECT ?f WHERE GP} \)
\end{algorithmic}
\end{algorithm}

\textbf{S-Dice operator with} \( \phi^S \): The following listing is the SPARQL query generated for the running example (Ex. 13.1), where the spatial predicate is interpreted as a topological relation. The graph pattern \( GP \) for the query is created in Lines 2 to 8.

```scheme
Q = \text{SELECT ?obs WHERE}
\{ ?obs rdf:type qb:Observation ;
  gwn:customerID ?cust .
  ?cust qb4o:memberOf gwn:customer ;
  skos:broader ?city ;
  gwn:customerGeo ?custGeo ,
\text{FILTER (bif:st_within(}?cityCentGeo, ?custGeo, 5))\text{)}
```

\textbf{S-Dice operator with} \( f^S(x) \) and a regular Boolean predicate \( \phi \): The following listing is the SPARQL query generated for the running example (Ex. 13.2), where the spatial predicate is interpreted as a combination of a spatial function and a regular predicate. The graph pattern \( GP \) for the query is created in Lines 2 to 9.

```scheme
Q = \text{SELECT ?obs WHERE}
\{ ?obs rdf:type qb:Observation ;
  gwn:customerID ?cust .
  ?cust qb4o:memberOf gwn:customer ;
  skos:broader ?city ;
  gwn:customerGeo ?custGeo ,
\text{FILTER (bif:st_distance(}?custGeo, ?cityCentGeo AS ?distance) \text{FILTER ( ?distance < 5 ))}
```

\textbf{S-Roll-up Generator}. To generate a SPARQL query for the \( s\)-roll-up operator from a high-level SOLAP
expression, \( SRU(C)[f^S(L(d_i)), agg(m)] \) (Def. 19), where parameter \( f^S(L(d_i)) \) denotes a spatial function on spatial level members and \( agg(m) \) is an aggregate function on measures. For illustration, we use the query example for s-rollup given in Sect. 4 for s-rollup (Ex. 14): “total amount of sales to customers by city of the closest suppliers”. We follow the main steps sketched in Algorithm 4 in the following.

Lines 2, 3. Build the roll-up path using helper function \( RUPath \). In addition to the variables given in the \( RUPath \) function, we also need to consider measures and measure value variables (Line 3) since we aggregate the measures. A measure is specified in the following listing of the running example as \( gnw: salesAmount \). The following lines are added to the graph pattern \( GP \):

```
{?obs a qb:Observation ;
  gnw:customerID ?cust ;
  ?cust gb4:memberOf gnw:customer ;
  skos:broader ?city .
}
```

Line 4. Build inner select subquery to apply the spatial function \( f^S \) on the spatial level members \( L(d_i) \) (i.e., Customer, Supplier). In the example, we will use this information to create a dynamic spatial hierarchy from the Customer to the City level.

Line 5. Call \( RUPath \) for the inner select subquery to link the geometry attributes of base level members with different variables and create a graph pattern \( GP' \) for the inner select. The following lines are added to the graph pattern \( GP' \):

```
{?obs a qb:Observation;
  gnw:customerID ?cust1 ;
  gnw:supplierID ?sup1 ;
  ?cust1 gb4:memberOf gnw:customer Geo ;
  ?cust1 gb4:memberOf gnw:city Geo .
}
```

Line 6. Build the bind statement in order to calculate the spatial function \( f^S(L(d_i)) \) on spatial level members. For the running example the spatial function is \( st\_distance \). The following lines are added to the graph pattern \( GP' \):

```
BIND (bif:st_distance(?cust1Geo, ?sup1Geo) AS ?distance)
```

Algorithm 4: \( SRUGenerator(G(C)[f^S(L(d_i)), agg(m)]) \)

**Input:** \( G(C)[f^S(L(d_i)), agg(m)] \)

**Output:** \( Q \)

1. \( Q = \emptyset \):
   2. \( GP = RUPath(G(C)[f^S(L(d_i)), agg(m)]) \)
   3. \( GP = GP \cup \{ (??ID, ??) \} \)
   4. for \( f^S(L(d_i)) \) do
   5. \( GP' = RUPath(G(C)[f^S(L(d_i))]) \)
   6. \( Q' = \emptyset \)
   7. \( Q' = SELECT ?x (AGG(?y) AS ??) \) WHERE \( GP' \) GROUP BY ?x
   8. \( GP = GP \cup Q' \) (FILTER ?x = ??)
   9. let \( l_s = l_s' \)
10. return \( Q = SELECT ?f ??' AGG(?m) \) WHERE \( GP' \) GROUP BY ?f ??'

Line 7. Generate the inner select query \( Q' \) using graph pattern \( GP' \) (Lines 5 and 6). Select the corresponding level members (Customer level for the running example) and group them in a group by statement on the selected level members. Note that this is where the spatial function \( f^S(L(d_i)) \) is called with a wrapper expression (e.g., MIN, MAX, etc.) to find the closest distance. The following lines illustrate the inner select query \( Q' \):

```
Q' = SELECT ?cust1 (MIN(?distance) AS ?minDistance) WHERE GP GROUP BY ?cust1
```

Lines 8, 9. Build the filter statement for the whole query based on the output of the spatial function, which is calculated in the inner select subquery. Then, add the filter and inner select subquery to the main graph pattern \( GP' \) (Line 8). The filter statement for the running example is:

```
FILTER (?cust = ?cust1 && bif:st_distance (?custGeo, ?supGeo) = ?minDistance)
```

Note that in Line 9, the spatial target level \( l_s \) (City) is altered to a dynamic spatial level \( l_s' \) since applying the spatial function creates a dynamic hierarchy.

Line 10. Generate query \( Q \) for computing the facts \( f \in FM' \) based on graph pattern \( GP \) created in the
previous steps. The measures are also aggregated at the spatial target level (closest City, which is dynamically selected). The group by statement is applied on the fact members and target level members. In our running example we obtain the following case for the generated s-roll-up query \( Q \).

**S-Roll-up operator:** The following listing shows the generated SPARQL query. Graph pattern \( GP \) for the inner select subquery is created in Lines 13 to 20 and the graph pattern \( GP \) for the whole query is created in Lines 2 to 22.

```sql
Q = SELECT ?obs ?city (SUM(?sales) AS
?totalSales) WHERE { ?obs a qb:Observation ;
  gnw:customerID ?cust ;
  gnw:supplierID ?sup ;
  gnw:customerAmount ?sales .
  ?cust qb4o:memberOf gnw:customer ;
  gnw:customerGeo ?custGeo ;
  gnw:customerName ?custName ;
  skos:broader ?city .
  ?city qb4o:memberOf gnw:city .
  ?sup qb4o:memberOf gnw:supplier .
} GROUP BY ?city
```

**S-Drill-down Generator:** The semantics of the s-drill-down operator are defined in the same way as for the s-roll-up operator with the condition that the input cube \( C \) for s-roll-up is obtained using a function BaseCube such that \( SDD(C)[f^S(L(d_i)), \text{agg}(m)] = SRU(BaseCube(C))[f^S(L(d_i)), \text{agg}(m)] \) (Def. 20).

Therefore, no generator algorithm and steps are specified since an s-drill-down operator corresponds to a rewriting of an s-roll-up operator, which is obtained with a Base function that calls the base cube graph in SRUGenerator as: \( SDDGenerator = SRUGenerator(Base(G^1(C)), f^S(L(d_i)), \text{agg}(m)) \).

**5.2. Nested SOLAP operations to SPARQL**

We now show how a SPARQL query can be generated for a nested SOLAP expression. In general, a nested set of SOLAP operators can be rewritten into an expression with an additional s-dice, on top of a series of s-roll-ups, on top of one or more s-slices, on top of an s-dice, i.e., \( (s-dice_2(s-roll-up_2(\ldots s-roll-up_1(s-slice_1(\ldots s-slice_0(s-dice(C))))))) \).

Let us begin with a simpler nested form that shows the most typical pattern, namely \( (s-roll-up (s-slice (s-dice(C)))) \), where initially a subcube graph is selected by s-dice. Afterwards, an s-slice is performed on a higher level of a dimension. Then, an s-roll-up is applied, which aggregates the measures in the sliced cube from a lower level to a higher level. Finally, we could also perform another s-dice for filtering the measures. There may be several s-slices and s-roll-ups in between.

We formulate the nested SOLAP query as \( 3(s-roll-up 2(s-slice 1(s-dice(C)))) \) and apply our running examples such that the enumeration of operators can be interpreted as follows: 1 Get the subcube graph of customers that are located within a 5 km distance from their city center, 2 slice on the customers of the largest country (which drops the dimension and leaves out all the other countries), and 3 get the total amount of sales for customers by the city of their closest suppliers (aggregates the measure Sales amount from Customer to Closest City level). Finally, we may also perform another (s-)dice on measures, e.g., filtering the total amount of sales greater than 10500. To perform nested SOLAP operators, we identify a set of principles to be considered by the algorithm.

**Principle 1:** Perform s-dice in the beginning or at the end.

**Principle 2:** If there are several s-roll-up or s-slice operations call their generator algorithms repeatedly.

**Principle 3:** Always separate \( FILTER \) clauses when a SOLAP generator algorithm is used. Enumerate separated \( FILTER \) clauses. If a SOLAP operator is the final function added to the graph pattern, do not separate the \( FILTER \) clause.

**Principle 4:** Build the final graph pattern with the separated and enumerated \( FILTER \) clauses with respect to Principle 3.

**Principle 5:** Drop the main \( SELECT \) clause from each SOLAP generator algorithms and build only one \( SELECT \) that is added to the query at the end.

**Principle 6:** Separate the \( GROUP BY \) clause and \( AGG \) functions from the s-roll-up generator algorithms (and enumerate them), and build add them to the main (outer) \( SELECT \) clause at the end.
Algorithm 5: WriteSPARQL((SRIU(C)\(\mathbf{f}^S(L(d_{i})),\ agg(m))|(SS(C)[l_{i},l_{s},v_{a},]|(SD(C)[\phi^S]))):  

Input: \((SRIU(C)\mathbf{f}^S(L(d_{i})),\ agg(m))|(SS(C)[l_{i},l_{s},v_{a},]|(SD(C)[\phi^S]))\)  

Output: \(Q\)  

1. \(Q = \emptyset \); \(GP = \)  
2. \(\text{RUPath}(\mathcal{G}^C_{\phi}[,l_{i},l_{s},aID,?a,?f])\)  
3. \(GP = GP \cup (\forall id^S(m) ?m)\)  
4. \(GP^{1} = \text{S-DiceGenerator}(\mathcal{G}^{1}_{C},\phi^{S})\) \(\setminus \text{FILTER}^1 \setminus \text{SELECT}\)  
5. \(GP = GP \cup GP^{1}\)  
6. \(GP^{2} = \text{S-SliceGenerator}(\mathcal{G}^{1}_{C},v_{s},l_{b},l_{s})\) \(\setminus \text{FILTER}^2 \setminus \text{SELECT}\)  
7. \(GP = GP \cup GP^{2}\)  
8. \(GP = GP \cup \text{FILTER}^1 \cup \text{FILTER}^2 \cup \)  
9. \(\text{SRUGGenerator}(\mathcal{G}^{1}_{C},\mathbf{f}^{S}(L(d_{i})),\ agg(m))\)  
10. \(\text{SELECT} \ \text{GROUP BY} v_{1} \ \text{AGG}^{1}(?m)\) \(\text{WHERE} GP\)  

return \(Q = \text{SELECT} ?t_{s}, \text{AGG}^{1}(?m)\) \(\text{WHERE} GP\)  

To separate the \(\text{FILTER}\) clauses, we call SOLAP generator algorithms without their \(\text{FILTER}\) clause and enumerate each \(\text{FILTER}\) clause for each SOLAP generator algorithm that is used, i.e., \(\text{S-SliceGenerator}(\mathcal{G}^{1}_{C},\phi^{S}) \setminus \text{FILTER}^1\) (Algorithm 5, Line 4). Then, we build the final graph pattern with these separated \(\text{FILTER}\) clauses i.e., \(GP = GP \cup \text{FILTER}^1 \cup \text{FILTER}^2\) (Line 8). When the last SOLAP generator algorithm is called, the output is directly added to the graph pattern without separating its \(\text{FILTER}\) clause (Line 9). Throughout the algorithm, all the \(\text{SELECT}\) clauses are omitted and combined into one \(\text{SELECT}\) in the output on Line 10. According to Principle 6, if there are any \(\text{GROUP BY}\) clauses and \(\text{AGG}\) functions (on measures) in inner selects, we eliminate them with "\(\setminus\)" from the inner selects (Line 9) and finally build the main (outer) select query with (Line 10). Note that in the algorithm, the general graph pattern \(GP\) is initially created by the \(\text{RUPath}\) function (Line 2) and incremented with triple patterns for selected measures (Line 3).

**Example 16.** \((\mathbf{f}^S\setminus \text{ROLL-UP} (\mathbf{f}^S\setminus \text{SLICE} (\mathbf{f}^S\setminus \text{DICE}(C))))\):  
1. Get the subcube graph of customer that are located within a 5 km distance from their city center.  
2. \(\text{slice}\) on the customers of the largest country (which drops the dimension and leave out all the other countries), and  
3. \(\text{get the total amount of sales by customers by the city of their closest suppliers (aggregates the measure Sales amount from Customer to Closest City level). The query is written starting from the innermost operator \(\text{s-dice}\) to the outermost operator \(\text{s-roll-up}\).} \)

The graph pattern \(GP\) is initially created with the \(\text{RUPath}\) function for the corresponding levels and level attributes (Lines 3 to 18 of the generated SPARQL query in the listing above). The first opera-
tor is called by function $S$-DiceGenerator, where the first \texttt{FILTER} clause of the outer selected is added to the query (Line 37). The second operator is called by the $S$-SliceGenerator function excluding its \texttt{FILTER} clause (Lines 19 to 27), which is followed by the SRUGenerator function without \texttt{GROUP BY} and \texttt{AGG} statements (Lines 28 to 36). Note that in Line 36, \texttt{GROUP BY} is applied on the lower level Customer, and the actual \texttt{GROUP BY} for the target City level is applied in the last line (Line 40). Separated \texttt{FILTER} clauses for the $S$-DiceGenerator and $S$-SliceGenerator functions are later added to the graph pattern (Algorithm 5, Line 8) corresponding to Lines 37 and 38 in the above example. The main outer select query is defined in the first line by specifying the target level (City) and aggregate function on measures (sum of the total Sales).

6. Related work

DW and OLAP technologies have been successful for analyzing large volumes of data [1]. Combining DW/OLAP technologies with RDF data makes RDF data sources more easily available for interactive analysis. The following work concerns the integration of DW/OLAP with the SW.

**DW/OLAP and Semantic Web** Using OLAP to analyze SW data is considered in several approaches, e.g., MD modeling from ontologies is studied in [7,15]. However, these approaches do not support multidimensional querying of RDF data in SPARQL but instead require data to reside in a MD or relational database query engine, which limits the applicability for frequently updated RDF data. Kämpgen et al. propose an extended model [12] on top of the RDF Data Cube Vocabulary (QB) [5] for interacting with statistical linked data via OLAP operations directly in SPARQL. However, it has the inherent limitations of QB and thus cannot support OLAP dimensions with hierarchies and levels, and built-in aggregate functions. Etcheverry et al. introduce QB4OLAP [8] as an extended vocabulary based on QB, with a full MD metamodel, supporting OLAP operations directly over RDF data with SPARQL queries. Nath et al. considers creating an Extract–Transform–Load (ETL) framework for semantic data warehouses [6]. However, none of these vocabularies and approaches support spatial DWs, unlike our proposed QB4SOLAP.

**Spatial DW and OLAP** The constraint representation of spatial data has been the focus in many fields from databases to AI [18]. Extending OLAP with spatial features has also attracted the attention of the data warehousing community. Several conceptual models are proposed for representing spatial data in data warehouses. Stefanovic et al. [11] considers constructing and materializing spatial cubes in their proposed model. The MultiDim conceptual model, introduced by Malinowski and Zimányi [22], copes with spatial features and is extended in [21], to include complex geometric features (continuous fields), with a set of operations and an MD calculus supporting spatial data types. Gómez et al. [9] propose an algebra and a general framework for OLAP cube analysis on discrete and continuous spatial data. Even though spatial data warehousing is thus widely studied, none of this work has considered SW data, unlike QB4SOLAP.

**Geospatial Semantic Web** The Open Geospatial Consortium (OGC) has proposed GeoSPARQL [3] as a vocabulary to represent and query spatial data in RDF using an extension to SPARQL. Kyzirakos et al. present a comprehensive survey of data models and query languages for linked geospatial data in [14], and propose a semantic geospatial data store called Strabon in [13]. Strabon has an extensive query language called stSPARQL , which is however limited to the specific environment. LinkedGeoData is a significant contribution on interactively transforming OpenStreetMap data to RDF data [20]. GeoKnow [19] is a more recent project with focus on linking geospatial data from heterogeneous sources. Andersen et al. considers publishing/converting open spatial data as Linked Open Data [2]. However, none of this work considers the MD aspects of geospatial data, unlike QB4SOLAP.

In summary, none of the above work provides a substantial foundation for modeling and querying spatial data warehouses on the Semantic Web, unlike the QB4SOLAP vocabulary, SOLAP operators, and SPARQL generation algorithms presented in this paper.

7. Conclusions and research directions

Motivated by the need for a formal foundation for spatial data warehouses on the Semantic Web, this paper made a number of contributions. First, it proposed

---

9http://www.openstreetmap.org
the QB4SOLAP vocabulary (metamodel) which supports spatially enhanced multidimensional (MD) data cubes over RDF data. This allows users to publish MD spatial data in RDF format. Second, the paper defines a number of spatial OLAP (SOLAP) operators over the defined QB4SOLAP cubes, allowing spatial analytical queries over RDF data, and gives their formal semantics. Third, the paper provides algorithms for generating spatially extended SPARQL queries from individual and nested SOLAP operators, allowing users to write their spatial analytical queries in our high-level SOLAP language instead of the lower-level and more complex SPARQL. Fourth, the vocabulary, operators, and query generation algorithms are validated by applying them to a realistic use case.

Several directions are interesting for future research: extending the formal techniques and algorithms for generating SOLAP queries in SPARQL to work over multiple RDF cubes, i.e., to support s-drill-across; supporting spatial aggregation (s-aggregation) with user-defined functions over spatial measures; developing a query system with a GUI for users to perform SOLAP operations, generate the SPARQL queries and return the results for further analysis.

Acknowledgment. This research was partially funded by the Erasmus Mundus Joint Doctorate in Information Technologies for Business Intelligence - Doctoral College (IT4BI-DC).

References


