Aalborg Universitet

AALBORG UNIVERSITY
DENMARK

A Foundation for Spatial Data Warehouses on the Semantic Web

Gur, Nurefsan; Pedersen, Torben Bach; Zimányi, Esteban; Hose, Katja

Published in:
Semantic Web

DOI (link to publication from Publisher):
10.3233/SW-170281

Publication date:
2018

Document Version
Accepted author manuscript, peer reviewed version

Link to publication from Aalborg University

Citation for published version (APA):
Gur, N., Pedersen, T. B., Zimányi, E., & Hose, K. (2018). A Foundation for Spatial Data Warehouses on the Semantic Web. *Semantic Web*, *9*(5), 557-587. Advance online publication. https://doi.org/10.3233/SW-170281

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal -

Take down policy
If you believe that this document breaches copyright please contact us at vbn@aub.aau.dk providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from vbn.aau.dk on: March 29, 2024

Aalborg Universitet

AALBORG UNIVERSITY
DENMARK

A Foundation for Spatial Data Warehouses on the Semantic Web

Gur, Nurefsan; Pedersen, Torben Bach; Zimányi, Esteban; Hose, Katja

Published in:
Semantic Web

DOI (link to publication from Publisher):
10.3233/SW-170281

Publication date:
2018

Document Version
Accepted author manuscript, peer reviewed version

Link to publication from Aalborg University

Citation for published version (APA):
Gur, N., Pedersen, T. B., Zimányi, E., & Hose, K. (2018). A Foundation for Spatial Data Warehouses on the Semantic Web. Semantic Web, 9(5), 557-587. https://doi.org/10.3233/SW-170281

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

? Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
? You may not further distribute the material or use it for any profit-making activity or commercial gain
? You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us at vbn@aub.aau.dk providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from vbn.aau.dk on: september 04, 2019

The final publication is available at IOS Press through http://dx.doi.org/10.3233/SW-170281

Undefined 1 (2009) 1–5
IOS Press

A Foundation for Spatial Data Warehouses on the Semantic Web

Editor(s): Mark Gahegan, The University of Auckland, New Zealand
Solicited review(s): Grant McKenzie, University of Maryland, United States. Benjamin Adams, University of California, United States. Kristin Stock, University of New Zealand, New Zealand.

Nurefşan Gür [a,b,*], Torben Bach Pedersen [a], Esteban Zimányi [b] and Katja Hose [a]
[a] *Center for Data Intensive Systems, Aalborg University, Selma Lagerlöfsvej 300, DK-9220 Aalborg Ø, Denmark*
E-mail: {nurefsan,tbp,khose}@cs.aau.dk
[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 are 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 through spatial location data or deriving topological relations among the hierarchy levels of the location dimension. This yields a demand for truly spatial DWs for better analysis purposes. Including the geometric information of the location data, sig-

*Corresponding author. E-mail: nurefsan@cs.aau.dk.

0000-0000/09/$00.00 © 2009 – IOS Press and the authors. All rights reserved

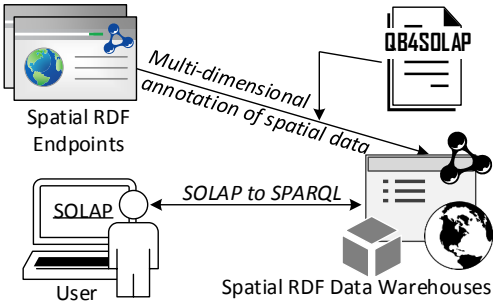


Fig. 1. QB4SOLAP approach to SOLAP on the SW

nificantly 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 2 for details).

Problem Definition. The proliferation of open geospatial data on the SW creates possibilities for advanced analysis of such data. Many examples exist of spatial Linked Open Data (LOD) published on the SW as RDF[1,2,3,4]. These datasets have observations and measures that are well suited for analytical queries (e.g., water/air quality measurements, immigration rates, EU subsidies in agriculture, crop revenue, etc.). However, such datasets are typically not modeled with spatial dimension levels and hierarchies. Thus, they cannot be queried with interactive spatial analytical queries (a.k.a. SOLAP) on the SW. In the current state of the SW, if a (spatial) DW user would like to query the existing spatial RDF data from the SW with SOLAP operations, the user needs to download the RDF data, map it to a relational data model (i.e., with a snowflake schema), and then import it into a traditional spatial data warehouse in order to query with SOLAP, which is slow, labor-intensive, and stores the data in a non-open format.

Our Approach. On the contrary, annotating spatial RDF datasets with QB4SOLAP [17,16] allows users to define spatial multidimensional concepts on top of existing RDF data. Hence, the user can create and publish spatial data warehouses on the Semantic Web, which can be easily queried with SOLAP operations. Fig. 1 depicts the general workflow scenario, where the spatial RDF datasets from endpoints can be annotated with QB4SOLAP. This makes it possible for end users to use SOLAP queries. However, writing a SOLAP query in SPARQL can be very complicated for users inexperienced with SPARQL (e.g., traditional DW users). Due to the lack of MD semantics of spatial RDF data and the lack of translation techniques from high-level SOLAP expressions to SPARQL, there is a considerable entry barrier for advanced spatial data analysis on the SW for data warehouse users.

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 generating 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.

Paper structure. The remainder of the paper is structured as follows. Section 2 discusses related work. Section 3 defines preliminary spatial and OLAP concepts. Section 4 defines the QB4SOLAP vocabulary, while Section 5 defines the SOLAP operators. Section 6 provides the SPARQL query generation algorithms. Finally, Section 7 concludes the paper and points to future research.

[1]EuroStat: `http://ec.europa.eu/eurostat`

[2]UK Environmental Data: `http://environment.data.gov.uk`

[3]Danish Agricultural Data: `https://datahub.io/dataset/govagribus-denmark`

[4]Australian Climate Observations: `https://datahub.io/dataset/acorn-sat`

2. 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. Kämpgen et al. propose an extended model [23] on top of the RDF Data Cube Vocabulary (QB) [6] 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 [11] 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 [8]. Varga et al. presents a comprehensive methodology for dimensional enrichment of statistical LOD by using QB4OLAP and provide a SW-based OLAP engine for traditional DW users [38]. However, these approaches and vocabularies support neither spatial DWs nor provide SOLAP operators for the SW.

Spatial DW and OLAP. The constraint representation of spatial data has been the focus in many fields from databases to AI [32]. Extending OLAP with spatial features has also attracted the attention of the data warehousing community. Bédard et al. first introduced the term SOLAP [4] in 1997. SOLAP systems [33,9] since then, have significantly been improved. Respectively, various papers improve the spatial aggregation functions and techniques [29,26,7,15].

Several conceptual models are proposed for representing spatial data in data warehouses. Stefanovic et al. [19] considers constructing and materializing spatial cubes in their proposed model. The MultiDim conceptual model, introduced by Malinowski and Zimányi [27], copes with spatial features and is extended in [37], to include complex geometric features (continuous fields), with a set of operations and an MD calculus supporting spatial data types. Gómez et al. [14] 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, those studies are limited to traditional *non-semantic* spatial data warehouses and SOLAP techniques. The work above neither considered semantic web data nor spatial analytical querying in SPARQL.

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 [25], and propose a semantic geospatial data store called Strabon in [24]. Strabon has an extensive query language called stSPARQL , which is however limited to the specific environment. LinkedGeoData is a significant contribution on interactive transformation of OpenStreetMap[5] data to RDF data [36]. GeoKnow [34] 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 these works consider the MD aspects of geospatial data or allow querying with SOLAP on the SW, unlike QB4SOLAP. The QB4SOLAP vocabulary is validated with both the running example use case, the GeoNorthwind data cube, as well as a substantial real-world use case, the *GeoFarmHerdState* data cube [16]. GeoFarmHerdState is a spatial data cube about livestock holdings in Denmark, which integrates environmental and geographical open data from several sources, thus enabling a range of interesting SOLAP queries.

In summary, none of the related work, which is surveyed in the fields of “DW/OLAP and the SW", “Spatial DW and OLAP", and “Geospatial Semantic Web" 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.

3. 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.

[5] http://www.openstreetmap.org

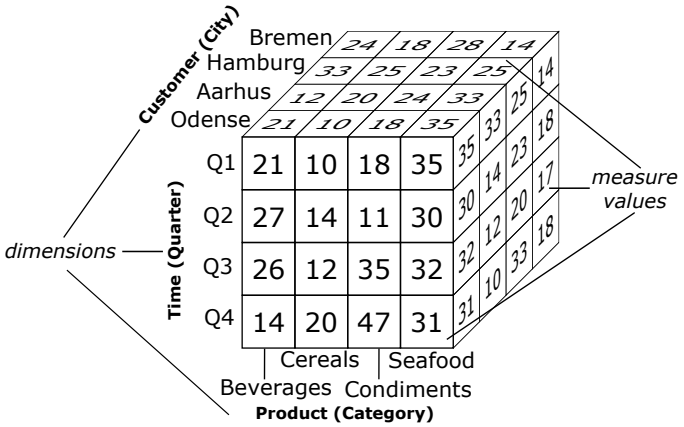


Fig. 2. A three-dimensional cube for Sales data

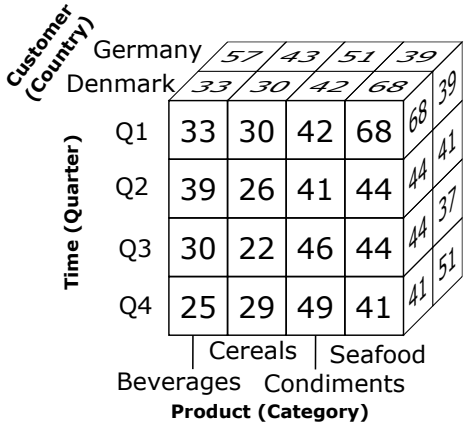


Fig. 3. Roll-up to the Country level

3.1. 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.

3.2. 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 $\mathcal{S}_{agg}$ aggregate two or more spatial objects and return a new spatial object. Union, Intersection, ConvexHull, and MinimumBoundingRectangle (MBR) are example operators of this class. Some spatial functions such as ConvexHull or MBR can also be interpreted as unary spatial functions with a single parameter, but here we only consider the aggregate versions of the functions. In order to make this clear, the aggregate versions of those functions are given with a prefix "*Aggr*" in the QB4SOLAP vocabulary (Fig. 6). For our purpose, it is enough to group all spatial aggregate functions into a single group, although more fine-grained classification proposals for spatial aggregate functions exist [7].

Definition 2. (Topological relations) The operators in the topological relation class $\mathcal{T}_{rel}$ are commonly expressed in the RCC8[6] and DE-9DIM[7] models [31,10]. 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.

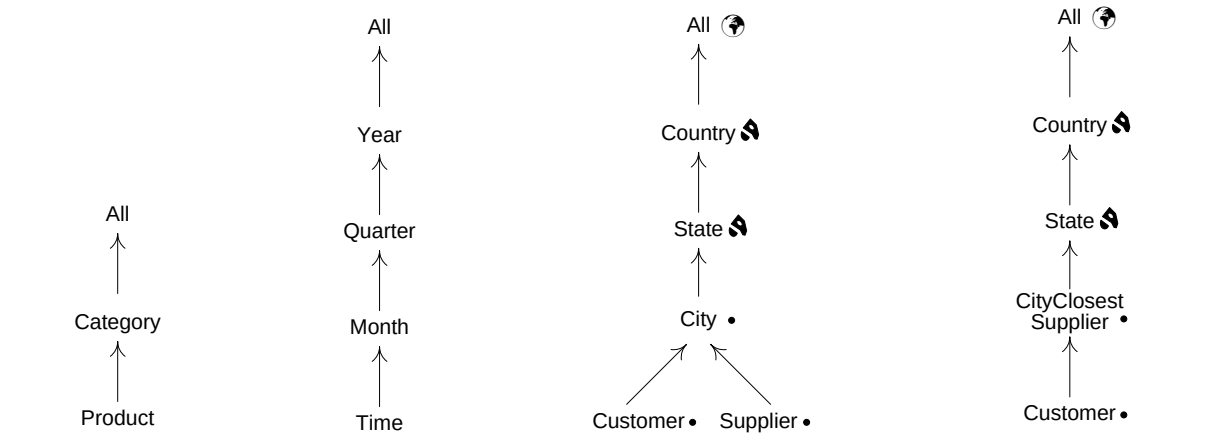
Definition 3. (Numeric operations) The operators in the numeric operation class $\mathcal{N}_{op}$ take one or more spatial objects and return a numeric value. Perimeter, Area, NoOfInteriorRings, Distance, HaversineDistance, and NoOfGeometries are example operators of this class.

3.3. 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

[6]RCC8 (Region Connection Calculus) describes regions in Euclidean space or in a topological space by their possible relations to each other.

[7]DE-9DIM (Dimensionally Extended Nine-Intersection Model) is a topological model that describes spatial relations of two geometries in two dimensions.



(a) Categories hierarchy in the Product dimension

(b) Calendar hierarchy in the Time dimension

(c) Spatial Geography hierarchies in the Customer and Supplier dimensions

(d) Dynamic spatial Geography hierarchy in the Customer dimension

Fig. 4. Dimension hierarchies

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*. 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. 2. Each cell in the cube is an observation fact, which is characterized by dimension and measure values. The hierarchies of this cube are given in Fig. 4a–c. Thus, in the cube shown in Fig. 2, 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.

3.4. 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 $\rightarrow$ State $\rightarrow$ Country $\rightarrow$ *All* (Fig. 4c). 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.

Following the philosophy of spatially extended MultiDim conceptual model [37], MD concepts such as levels are considered to be spatial, only if they record the spatial characteristics of the concepts as geometries. For instance, “continent" might be considered as a spatial object, in theory or in other vocabularies. However, if there is no information about the geometry of the continents in the schema and in the instance data, continent does not become a spatial level (Ext. 7), although continent might still be a traditional level (Def. 7) of the spatial hierarchy Geography with alphanumeric attributes (i.e., continent name, code, and etc.).

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. 7 shows the multidimensional schema of the GeoNorthwind data warehouse, which is used as running example in the paper.

3.5. 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. 3).

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. 3, an example is "drill-down to the City level".

3.6. 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. 3.1. 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. 5.

Table 1
Sample (instance) data for the Sales cube

Customer City		**Supplier**			**Total**
Customer		s1	s2	s3	**Sales**
Düsseldorf	c1	8	–	3	11
	c2	10	–	–	10
Dortmund	c3	7	4	–	11
	c4	–	20	3	23
Münster	c5	–	–	30	30

Table 2
Roll-up of the Sales cube

Customer City	**Sales**
Düsseldorf	21
Dortmund	34
Münster	30

Table 3
S-Roll-up of the Sales cube

CityClosest Supplier	**Sales**
Düsseldorf	25
Dortmund	20
Münster	33

Table 4
Customer to Supplier distance

		Supplier City		
		Supplier		
Customer City		Düsseldorf	Dortmund	Münster
Customer		s1	s2	s3
Düsseldorf	c1	15 km	45 km	30 km
	c2	15 km	60 km	60 km
Dortmund	c3	15 km	30 km	45 km
	c4	45 km	15 km	15 km
Münster	c5	60 km	45 km	15 km

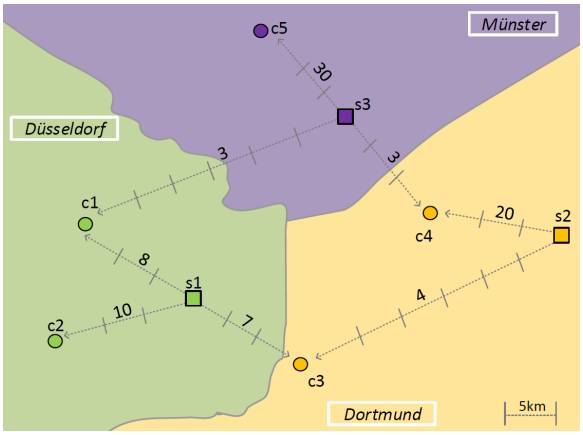


Fig. 5. Example map of Sales (instance) data

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 in Fig. 5, 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. 4a–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. 5, 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. 4d 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. 4d 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).

4. The QB4SOLAP vocabulary

In this section, we formally define how to represent (spatial) data cubes in RDF. We use as running example the GeoNorthwind data warehouse whose conceptual schema is given in Fig. 7.

The QB4OLAP [11] vocabulary allows to define *cube schemas* and *cube instances* as RDF triples. QB4OLAP is an extension of the RDF Data Cube Vocabulary (QB) [6] with multidimensional concepts in order to be able to support OLAP operations directly over RDF data with SPARQL queries. We extended QB4OLAP (v1.2)[8] with spatial concepts to give QB4SOLAP [17]. We based our extension on GeoSPARQL [30], a standard from the Open Geospatial Consortium (OGC) for representing and querying geospatial linked data for the Semantic Web. Since our base vocabulary QB4OLAP uses the MultiDim conceptual model to describe the multidimensional concepts, we base our definitions on a spatially extended version of MultiDim conceptual model [37] for spatial extension of the MD concepts. Fig. 6 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 ($\mathcal{S}_{agg}$ 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 datasets 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. 6, we use the schema level metadata to produce SPARQL queries that implement SOLAP operators on cube members.

Terms with capitalized initials and non-italic font in Fig. 6 represent RDF classes, terms with capitalized initials and italic font represent RDF instances, and terms with non-capitalized initials represent RDF properties. Classes in external vocabularies are depicted in light gray background and font. RDF Cube (QB), QB4OLAP, and QB4SOLAP classes are shown, respectively, with white, light gray, dark gray backgrounds. Original QB terms are prefixed with `qb:`[9]. QB4OLAP and QB4SOLAP terms are prefixed, respectively, with `qb4o:`[10] and `qb4so:`[11]. Spatial classes and properties are prefixed with `geo:`[12].

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

$$\mathcal{T} = (\mathcal{I} \cup \mathcal{B}) \times \mathcal{I} \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$$

where $\mathcal{I}$ is the set of *IRIs*, $\mathcal{B}$ is the set of *blank nodes*, and $\mathcal{L}$ 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 $\mathcal{G} \subset \mathcal{T}$.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 $x \in (\mathcal{I} \cup \mathcal{B})$ 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 $id(x) : \mathcal{G} \rightarrow \mathcal{I}$, that given a MD element x returns its identifier $\mathcal{I}$ 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., $id^S(x)$ for a cube schema identifier and $id^I(x)$ for a cube instance identifier.

[8] QB4OLAP v1.2: `https://github.com/lorenae/qb4olap/blob/master/rdf/qb4olap.1.2.ttl`

[9] RDF cube: `http://purl.org/linked-data/cube#`

[10] QB4OLAP: `http://purl.org/qb4olap/cubes#`

[11] QB4SOLAP: `http://w3id.org/qb4solap#`

[12] GeoSPARQL:`http://www.opengis.net/ont/geosparql#`

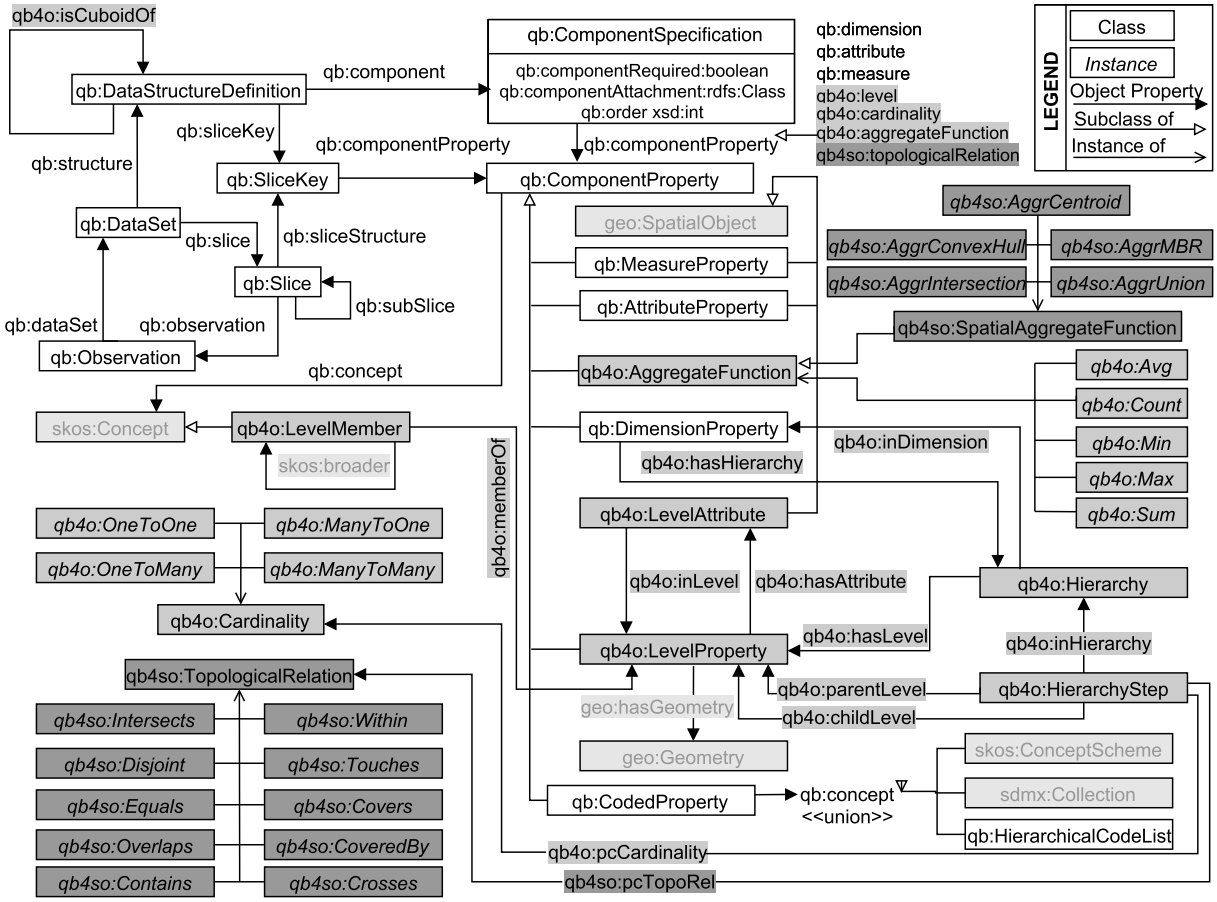


Fig. 6. The QB4SOLAP vocabulary

4.1. Defining spatial data cube schemas with QB4SOLAP

An *n-dimensional cube schema* $\mathcal{CS}$ is a tuple $\mathcal{CS} = (D, M, F)$, 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 next how to represent a cube schema $\mathcal{CS}$ in RDF using QB4SOLAP. We denote the RDF graph of the cube schema $\mathcal{G}^S$. In the examples we prefix the elements of $\mathcal{G}^S$ with `gnw:`. We follow a similar naming convention for schema elements as in QB4OLAP. If there is a possibility of confusion for different MD concepts with same schema name, i.e., customer dimension and customer level, we suffix the dimensions with `Dim` (e.g., `gnw:customerDim` for dimension, and `gnw:customer` for level). The subgraph of $\mathcal{G}^S$ that refers to a specific schema element x is denoted by $\mathcal{G}^S_x$ and the unique identifier of x is denoted by $id^S(x)$.

Definition 5. (Dimensions) An n-dimensional cube schema $\mathcal{CS}$ 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 $\mathcal{G}^S$ with `qb:DimensionProperty`. Each hierarchy $h \in H(d_i)$ is linked to its dimension d_i with the `qb4o:hasHierarchy` property. The RDF graph formulation of the dimensions D is represented as

$$\mathcal{G}^S_D = \bigcup_{i=1}^{n} \mathcal{G}^S_{d_i}$$

where

$$\mathcal{G}^S_{d_i} = \{(id^S(d_i)\ \texttt{rdf:type}\ \texttt{qb:DimensionProperty})\} \cup \bigcup_{h \in H(d_i)} \{(id^S(d_i)\ \texttt{qb4o:hasHierarchy}\ id^S(h))\}$$

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

Example 2. The triples below show how some of the dimensions of the GeoNorthwind DW (Fig. 7) 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:supplierDim rdf:type qb:DimensionProperty ;
   qb4o:hasHierarchy gnw:supplierGeography .
gnw:productDim rdf:type qb:DimensionProperty ;
   qb4o:hasHierarchy gnw:categories .
gnw:employeeDim rdf:type qb:DimensionProperty .
```

Definition 6. (Hierarchies) A dimension d has a set of hierarchies $H(d) = \{h_1, \ldots, h_m\}$, where each hierarchy h_i has a set of levels $L(h_i)$ (Def. 7). Each hierarchy $h_i \in H(d)$ is defined in the cube schema graph $\mathcal{G}^S$ with the `qb4o:Hierarchy` predicate and is linked with its dimension d by the `qb4o:inDimension` property. Each level $l \in L(h_i)$ that belongs to a hierarchy h_i is defined with the `qb4o:hasLevel` property. The RDF graph formulation of the hierarchies $H(d)$ is represented as

$$\mathcal{G}^S_{H(d)} = \bigcup_{i=1}^{m} \mathcal{G}^S_{h_i}$$

where

$$\mathcal{G}^S_{h_i} = \{(id^S(h_i)\ \texttt{rdf:type}\ \texttt{qb4o:Hierarchy})\} \cup \{(id^S(h_i)\texttt{qb4o:inDimension}\ id^S(d))\} \cup \bigcup_{l \in L(h_i)} \{(id^S(h_i)\ \texttt{qb4o:hasLevel}\ id^S(l))\}$$

Extension 6. (Spatial hierarchies) A hierarchy is spatial if it contains at least one spatial level. A spatial hierarchy h_{i_s} belongs to the set of spatial hierarchies $H_s(d)$, which is a subset of the set of hierarchies $H(d)$, such that $h_{i_s} \in H_s(d) \subseteq H(d)$.

Example 3. The triples below show how some of the hierarchies of the GeoNorthwind DW (Fig. 7) are represented in RDF using Def. 6 and Ext. 6. As we will see below, the Geography hierarchies in the Customer and Supplier dimensions are spatial since they have spatial levels (City, State, etc.)

```
# Hierarchies
gnw:customerGeography rdf:type qb4o:Hierarchy ;
   qb4o:inDimension gnw:customerDim ;
   qb4o:hasLevel gnw:customer, gnw:city,
      gnw:state, gnw:country .
gnw:supplierGeography rdf:type qb4o:Hierarchy ;
   qb4o:inDimension gnw:supplierDim ;
   qb4o:hasLevel gnw:supplier, gnw:city,
      gnw:state, gnw:country .
gnw:categories rdf:type qb4o:Hierarchy ;
   qb4o:inDimension gnw:productDim ;
   qb4o:hasLevel gnw:product, gnw:category .
```

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 $\mathcal{G}^S$ 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

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

where

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

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

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

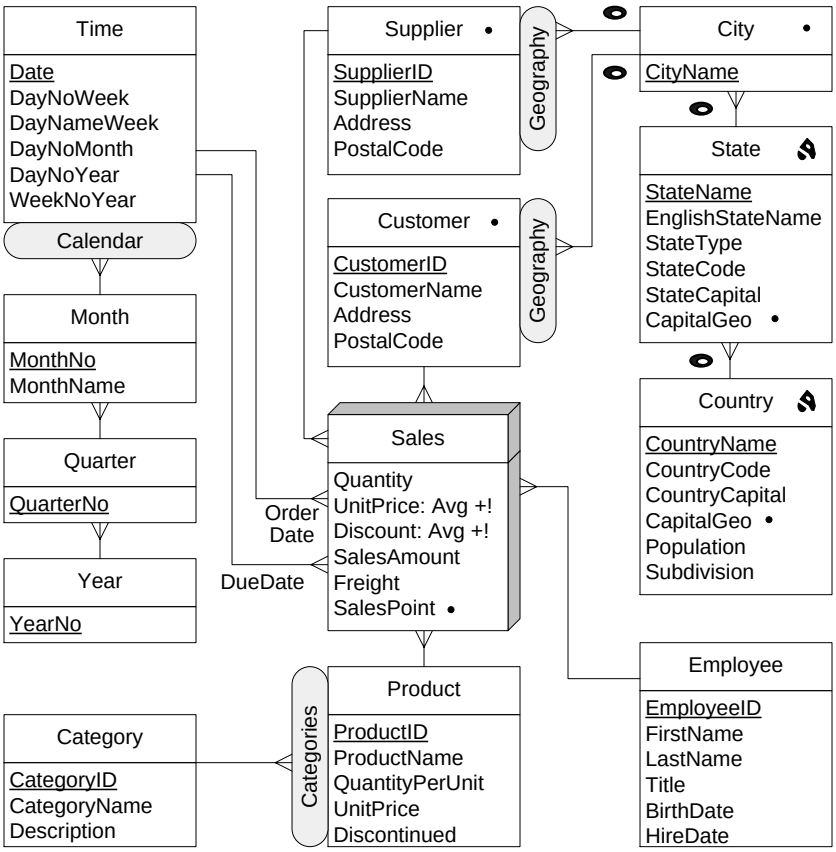


Fig. 7. Conceptual multidimensional schema of the GeoNorthwind data warehouse

where

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

Example 4. The triples below show how some of the levels of the GeoNorthwind DW (Fig. 7) are represented 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.

```
# Levels
gnw:customer rdf:type qb4o:LevelProperty ;
   qb4o:hasAttribute gnw:customerID ;
   qb4o:hasAttribute gnw:customerName ;
   qb4o:hasAttribute gnw:address ;
   qb4o:hasAttribute gnw:postalCode ;
   geo:hasGeometry gnw:customerGeometry .
gnw:city rdf:type qb4o:LevelProperty ;
   qb4o:hasAttribute gnw:cityName ;
   geo:hasGeometry gnw:cityGeometry .
```

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 attribute, 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 $\mathcal{G}^S$ with the `qb4o:LevelAttribute` predicate and is linked to its level l with the `qb4o:inLevel` property. Each attribute a_i is defined as ranging over XSD literals $\mathcal{L}$ using the `rdfs:range` property. The RDF graph formulation of the attributes $A(l)$ is represented as

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

where

$\mathcal{G}^S_{a_i} =$

$\{(id^S(a_i)\ \texttt{rdf:type qb4o:LevelAttribute})\} \cup$

$\{(id^S(a_i)\ \texttt{qb4o:inLevel}\ id^S(l))\} \cup$

$\{(id^S(a_i)\ \texttt{rdfs:range}\ \mathcal{L})\}$

Extension 8. (Spatial attributes) An attribute is spatial if it is defined over a spatial domain. A spatial attribute a_{i_s} belongs to the set of spatial attributes $A_s(l)$, which is a subset of the set of attributes $A(l)$, such that $a_{i_s} \in A_s(l) \subseteq A(l)$. 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 `rdfs:domain` property, which must be a geometry. Finally, the attribute must be specified as spatial object with the `rdfs:subclassOf` property. The RDF graph formulation of the spatial attributes $A_s(l)$ is represented as

$$\mathcal{G}^S_{A_s(l)} = \bigcup_{i=1}^{p} \mathcal{G}^S_{a_{i_s}}$$

where

$\mathcal{G}^S_{a_{i_s}} =$

$\{(id^S(a_{i_s})\ \texttt{rdf:type qb4o:LevelAttribute})\} \cup$

$\{(id^S(a_{i_s})\ \texttt{qb4o:inLevel}\ id^S(l))\} \cup$

$\{(id^S(a_{i_s})\ \texttt{rdfs:range}\ \mathcal{L}_s)\} \cup$

$\{(id^S(a_{i_s})\ \texttt{rdfs:subPropertyOf geo:Geometry})\} \cup$

$\{(id^S(a_{i_s})\ \texttt{rdfs:subClassOf geo:SpatialObject})\}$

Example 5. The triples below show how some of the attributes of the GeoNorthwind DW (Fig. 7) 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 that defines a Point type from the Geometry class, which is a subclass of Spatial Object.

```
# Attributes
gnw:customerID rdf:type qb4o:LevelAttribute ;
   qb4o:inLevel gnw:customer;
   rdfs:range xsd:Integer .
gnw:customerName rdf:type qb4o:LevelAttribute ;
   qb4o:inLevel gnw:customer;
   rdfs:range xsd:String .
gnw:address rdf:type qb4o:LevelAttribute ;
   qb4o:inLevel gnw:customer;
   rdfs:range xsd:String .
gnw:postalCode rdf:type qb4o:LevelAttribute ;
   qb4o:inLevel gnw:customer;
   rdfs:range xsd:String .
gnw:customerGeometry rdf:type
qb4o:LevelAttribute ;
   rdfs:subPropertyOf geo:Geometry ;
   qb4o:inLevel gnw:customer ;
   rdfs:range geo:wktLiteral;
   rdfs:domain geo:Point ;
   rdfs:subClassOf geo:SpatialObject .
```

Definition 9. (Hierarchy steps) A hierarchy h has a set of hierarchy steps $HS(h) = \{hs_1, \ldots, hs_q\}$, which define the structure of the hierarchy in relation with its corresponding levels. A hierarchy step $hs_i = (l_c, l_p, card) \in HS(h)$ entails a roll-up relation between a lower (child) level l_c to an upper (parent) level l_p with a cardinality $card$. The cardinality $card \in \{1\text{-}1, 1\text{-}n, n\text{-}1, n\text{-}n\}$ describes the number of members in one level that can be related to a member in the other level for both the child and the parent levels.

Each hierarchy step hs_i is defined in the cube schema graph $\mathcal{G}^S$ as a blank node $_:\texttt{hs}_i \in \mathcal{B}$ 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:childLevel` and `qb4o:parentLevel` properties, respectively. The cardinality $card$ of a hierarchy step is defined by the `qb4o:pcCardinality` property. The RDF graph formulation of the hierarchy steps $HS(h)$ is represented as

$$\mathcal{G}^S_{HS(h)} = \bigcup_{i=1}^{q} \mathcal{G}^S_{hs_i}$$

where

$\mathcal{G}^S_{hs_i} =$

$\{(_:\texttt{hs}_i\ \texttt{rdf:type qb4o:HierarchyStep})\} \cup$

$\{(_:\texttt{hs}_i\ \texttt{qb4o:inHierarchy}\ id^S(h))\} \cup$

$\{(_:\texttt{hs}_i\ \texttt{qb4o:parentLevel}\ id^S(l_p))\} \cup$

$\{(_:\texttt{hs}_i\ \texttt{qb4o:childLevel}\ id^S(l_c))\} \cup$

$\{(_:\texttt{hs}_i\ \texttt{qb4o:pcCardinality}\ id^S(card))\}$

Extension 9. (Spatial hierarchy steps) A hierarchy step is spatial if it relates a spatial child level l_{c_s} and a spatial parent level l_{p_s}, in which case it entails a topological relationships between these spatial levels. A spatial hierarchy step is then a tuple $hs_{i_s} = (l_{c_s}, l_{p_s}, card, topoRel)$ where the topological relation $topoRel$ 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 `qb4so:pcTopoRel` property. The RDF graph formulation of the spatial hierarchy steps $HS_s(h)$ (w.r.t. Def. 9) is represented as

$$\mathcal{G}^S_{HS_s(h)} = \bigcup_{i=1}^{q} \mathcal{G}^S_{hs_{i_s}}$$

where

$$\mathcal{G}^S_{hs_{i_s}} = \mathcal{G}^S_{hs_i} \cup \{(_{:}\texttt{hs}_i\ \texttt{qb4so:pcTopoRel}\ id^S(topoRel))\}$$

Example 6. The triples below show how the hierarchy steps of the Geography spatial hierarchy in the Customer dimension of the GeoNorthwind DW (Fig. 7) are represented in RDF using Def. 9 and Ext. 9. Note that all hierarchy steps are spatial and have an associated topological relation.

```
# Hierarchy steps
_:customerGeography_hs1 a qb4o:HierarchyStep ;
   qb4o:inHierarchy gnw:customerGeography ;
   qb4o:childLevel gnw:customer ;
   qb4o:parentLevel gnw:city ;
   qb4o:pcCardinality qb4o:ManyToOne ;
   qb4so:pcTopoRel qb4so:Within .
_:customerGeography_hs2 a qb4o:HierarchyStep ;
   qb4o:inHierarchy gnw:customerGeography ;
   qb4o:childLevel gnw:city ;
   qb4o:parentLevel gnw:state ;
   qb4o:pcCardinality qb4o:ManyToOne ;
   qb4so:pcTopoRel qb4so:Within .
_:customerGeography_hs3 a qb4o:HierarchyStep ;
   qb4o:inHierarchy gnw:customerGeography ;
   qb4o:childLevel gnw:state ;
   qb4o:parentLevel gnw:country ;
   qb4o:pcCardinality qb4o:ManyToOne ;
   qb4so:pcTopoRel qb4so: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 $l \sqsubseteq All$.

Definition 11. (Measures) An n-dimensional cube schema has a set of measures $M = \{m_1, \ldots, m_r\}$, which record the values of a phenomena being observed. Each measure $m_i \in M$ is defined in the cube schema graph $\mathcal{G}^S$ 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

$$\mathcal{G}^S_M = \bigcup_{i=1}^{r} \mathcal{G}^S_{m_i}$$

where

$$\mathcal{G}^S_{m_i} = \{(id^S(m_i)\ \texttt{rdf:type qb:MeasureProperty})\} \cup \{(id^S(m_i)\ \texttt{rdfs:range}\ \mathcal{L})\}$$

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_s} belongs to the set of spatial measures M_s, which is a subset of the set of measures M, such that $m_{i_s} \in M_s \subseteq M$. The RDF formulation of the spatial measures is similar as in Def. 11. However, the domain should range over spatial literals $\mathcal{L}_s$. The RDF graph formulation of the spatial measures M_s (w.r.t. Def. 11) is represented as

$$\mathcal{G}^S_{M_s} = \bigcup_{i=1}^{r} \mathcal{G}^S_{m_{i_s}}$$

where

$$\mathcal{G}^S_{m_{i_s}} = \{(id^S(m_{i_s})\ \texttt{rdf:type qb:MeasureProperty})\} \cup \{(id^S(m_{i_s})\ \texttt{rdfs:range}\ \mathcal{L}_s)\} \cup \{(id^S(m_{i_s})\ \texttt{rdfs:subClassOf geo:SpatialObject})\}$$

Example 7. The triples below show how the measures of the GeoNorthwind DW (Fig. 7) 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.

```
# 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 $\mathcal{CS} = (D, M, F)$, the fact F defines the structure of a cube with the `qb:DataStructureDefinition` property. The dimensions are given as `components` of the fact and are defined with the `qb4o:level` property. We assume that the fact F links the dimensions at the lowest granularity level, therefore `qb4o:level` links the lowest (base) level l_b of each dimension d_i, which is denoted as $l_b(d_i)$. The cardinality $card$ of the relationship between a dimension level and a fact is represented with the `qb4o:cardinality` property. Similarly, the measures are given as `components` of the fact and are defined with the `qb:measure` property. The aggregate function $aggr$ associated to each measure is represented with the `qb4o:aggregateFunction` property. The RDF graph formulation of the fact F is given in the following equation.

$$\mathcal{G}_F^S = \{(id^S(F)\ \texttt{rdf:type qb:DataStructureDefinition})\} \cup \bigcup_{d_i \in D} \{(id^S(F)\ \texttt{qb:component}\ [\texttt{qb4o:level}\ id^S(l_b(d_i)); \texttt{qb4o:cardinality}\ id^S(card)])\} \cup \bigcup_{m_i \in M} \{(id^S(F)\ \texttt{qb:component}\ [\texttt{qb:measure}\ id^S(m_i); \texttt{qb4o:aggregateFunction}\ id^S(aggr)])\}$$

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 $topoRel$ that must be satisfied by the related spatial levels, which is represented with `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.

$$\mathcal{G}_{F_s}^S = \{(id^S(F_s)\ \texttt{rdf:type qb:DataStructureDefinition})\} \cup \bigcup_{d_i \in D} \{(id^S(F_s)\ \texttt{qb:component}\ [\texttt{qb4o:level}\ id^S(l_b(d_i)); \texttt{qb4o:cardinality}\ id^S(card); \texttt{qb4so:topologicalRelation}\ id^S(topoRel)]\} \cup \bigcup_{m_i \in M} \{(id^S(F)\ \texttt{qb:component}\ [\texttt{qb:measure}\ id^S(m_i); \texttt{qb4o:aggregateFunction}\ id^S(aggr)])\}$$

Example 8. The triples below show how the fact of the GeoNorthwind DW (Fig. 7) is represented in RDF using Def. 12. Sales fact does not impose any topological relation between its spatial dimensions Supplier and Customer. SalesPoint is a spatial measure, which has a spatial aggregate function (AggrConvexHull).

```
# Cube definition
gnw:GeoNorthwind rdf:type
qb:DataStructureDefinition ;
   # Lowest level for each dimension
   qb:component [qb4o:level gnw:customer ;
   qb4o:cardinality qb4o:ManyToOne] ;
   qb:component [qb4o:level gnw:supplier ;
   qb4o:cardinality qb4o:ManyToOne] ;
   qb:component [qb4o:level gnw:product ;
   qb4o:cardinality qb4o:ManyToOne] ;
   ...
   # Cube measures
   qb:component [qb:measure gnw:quantity ;
      qb4o:aggregateFunction qb4o:Sum] ;
   qb:component [qb:measure gnw:unitPrice ;
      qb4o:aggregateFunction qb4o:Avg] ;
   qb:component [qb:measure gnw:discount ;
      qb4o:aggregateFunction qb4o:Avg] ;
   ...
   qb:component [qb:measure gnw:salesPoint ;
      qb4o:aggregateFunction qb4so:AggrConvexHull] .
```

4.2. Defining spatial data cube members with QB4SOLAP

We have explained in Sect. 4.1 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. 7) in RDF. We denote by $\mathcal{G}^I$ the RDF graph of the data cube instances. In the examples, we prefix the elements of $\mathcal{G}^I$ with `gnwi:`. The subgraph of $\mathcal{G}^I$ that refers to a specific cube *instance* x is denoted by $\mathcal{G}^I_x$ 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 $id^I(lm_i) \in \mathcal{I}$, which is linked in the cube instance graph $\mathcal{G}^I$ 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

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

where

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

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 $id^S(a_i)$. 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 $\mathcal{L}$ such that $v_{a_i} \in \mathcal{L}$. The RDF graph formulation of the attributes $A(lm)$ is represented as

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

where

$$\mathcal{G}^I_{a_i} = \{(id^I(lm)\ \ id^S(a_i)\ \ v_{a_i}) \mid lm \rightsquigarrow v_{a_i}\}$$

Definition 15. (Partial order on level members) A hierarchy step $hs = (l_c, l_p, card)$ between a child level l_c and a parent level l_p defines a set of roll-up relations $RU(hs) = \{r_1, \ldots, r_k\}$ where each $r_i = lm_{c_i} \sqsubseteq lm_{p_i}$ relates a child level member $lm_{c_i} \in LM(l_c)$ 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

$$\mathcal{G}^I_{RU(hs)} = \bigcup_{i=1}^{k} \mathcal{G}^I_{r_i}$$

where

$$\mathcal{G}^I_{r_i} = \{(id^I(lm_c)\ \texttt{skos:broader}\ id^I(lm_p)) \mid r_i = lm_{c_i} \sqsubseteq lm_{p_i}\}$$

Example 9. The triples below show how some level members of the GeoNorthwind DW (Fig. 7) are represented in RDF using Defs. 13–14.

```
gnwi:customer_1 rdf:type qb4o:LevelMember ;
   qb4o:memberOf gnw:customer ;
   gnw:customerID 1 ;
   gnw:customerName "Alfreds Futterkiste" ;
   gnw:address "Obere Str. 57" ;
   gnw:postalCode "12209" ;
   gnw:customerGeo
   "POINT(13.099 52.401)"^^geo:wktLiteral ;
   skos:broader gnwi:city_6 .
gnwi:city_6 rdf:type qb4o:LevelMember ;
   qb4o:memberOf gnw:city ;
   gnw:cityName "Berlin" ;
   gnw:cityGeo
   "POINT(13.4060 52.519)"^^geo:wktLiteral ;
   skos:broader gnwi:state_224 .
```

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

A fact member f_i is related to a set of dimension levels $L(f_i) = \{l_1, \ldots, l_r\}$ 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 $id^S(m_k)$. We denote by $f \rightsquigarrow v_{l_j}$ and $f \rightsquigarrow v_{m_k}$, respectively, the dimension values and

measure values associated with a fact f. The value $v_{l_j} \in \mathcal{I}$ is the identifier of a level member in $LM(l_j)$. Further, the value v_{m_k} for every measure m_k is a literal such that $v_{m_k} \in \mathcal{L}$. The RDF graph formulation of the fact members $FM(F)$ is represented as

$$\mathcal{G}^I_{FM(F)} = \bigcup_{i=1}^{t} \mathcal{G}^I_{f_i}$$

where

$$\mathcal{G}^I_{f_i} = \{(id^I(f_i)\ \texttt{rdf:type}\ \texttt{qb:Observation})\} \cup \bigcup_{l_j \in L(f_i)} \{(id^I(f_i)\ \ id^S(l_j)\ \ id^I(v_{l_j}) \mid f_i \rightsquigarrow v_{l_j})\} \cup \bigcup_{m_k \in M(f_i)} \{(id^I(f_i)\ \ id^S(m_k)\ \ id^I(v_{m_k}) \mid f_i \rightsquigarrow v_{m_k})\}$$

Example 10. The triples below show how a fact member of the GeoNorthwind DW (Fig. 7) 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:`. $id^S(a_{ID})$ is the surrogate key (Def. 8) that links the fact member to the corresponding dimensions' base level members.

```
gnwi:sale_10613_1 rdf:type qb:Observation ;
   gnw:customer gnwi:customer_1 ;
   gnw:supplier gnwi:supplier_6 ;
   gnw:product gnwi:product_13 ;
   ...
   gnw:quantity 8 ;
   gnw:unitPrice "6,00"^^xsd:decimal ;
   gnw:discount "0,10"^^xsd:decimal ;
   gnw:salesPoint
   "POINT(23.08 42.34)"^^geo:wktLiteral.
```

5. Semantics of SOLAP operators

This section defines a formal algebra for SOLAP operators. Examples of the operators are provided after their definitions. The complete SPARQL query examples are given at our website[13] and can be tested at our public endpoint[14]. The query runtimes for each SOLAP operator are given in Appendix A1, Table 5 for the use case dataset GeoNorthwind (Sect. 4.1, Fig. 7). These operators can be applied on spatially enhanced multidimensional data cubes (Sect. 3.3). 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. 3.1). Let $\mathcal{S}$ be the set of any spatial operators where $\mathcal{S} = (\mathcal{S}_{agg} \cup \mathcal{T}_{rel} \cup \mathcal{N}_{op})$, used to represent a *spatial predicate* $\phi^{\mathcal{S}} \in \mathcal{S}$ or a *spatial function* $\mathtt{f}^{\mathcal{S}} \in \mathcal{S}$, 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 17. (Slice) The *slice* operator removes a dimension from a cube $\mathcal{C}$ 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 in dimension 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 $\mathcal{C}$ by choosing a single spatial attribute value $v_s \in \mathcal{L}_s$ (Ext. 8) in a spatial level l_s (Ext. 7).

As for the semantics, s-slice takes an n-dimensional cube $\mathcal{C}$ as an argument. We assume that the cube has the cube schema $\mathcal{CS} = (D, M, F)$, 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: $\mathcal{SS}(\mathcal{C})[l_b, l_s, v_s] = \mathcal{C}'$, which returns a cube $\mathcal{C}'$ with $n-1$ dimensions and the schema $\mathcal{CS} = (D', M', F')$, 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 $\mathcal{C}'$ 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 $l_b \sqsubseteq l_s$. 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 $f \rightsquigarrow v_{d_i}$ where $v_{d_i} \equiv v_{l_{b(d_i)}}$ (Def. 16). In other words, dimensions

[13] http://extbi.cs.aau.dk/SOLAP4SW/queries
[14] http://extbi.lab.aau.dk/sparql

are associated to the fact members by the values of the dimensions' base level members $v_{l_{b(d_i)}}$. When the dimension d_i is clear in the context, we will use base level v_{l_b} 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_s} in the target level l_s is specified with respect to the given spatial literal value v_s. The value of v_s might be equal to a spatial attribute value in the target level l_s, thus v_{l_s} is characterized by the attribute value v_s and written as $v_{l_s} \rightsquigarrow v_s$ (Ex. 11). Or, v_s is an arbitrary spatial literal that entails a topological relation $\mathcal{T}_{rel}$ (i.e., within) in a value of the target spatial level v_{l_s}, which is written as $\exists v_{l_s} : \phi^{\mathcal{S}}(v_s)$ where $\phi^{\mathcal{S}}$ is a *spatial Boolean predicate* that represents a topological relation (Ex. 12). After applying the s-slice operator on cube $\mathcal{C}$, the new (sub)set of fact members is defined for both cases respectively as follows; $FM' = \{f \in FM \mid \exists\, v_{l_b} \in LM(l_b), v_{l_s} \in LM(l_s) : f \rightsquigarrow v_{l_b} \wedge v_{l_b} \sqsubseteq v_{l_s} \wedge v_{l_s} \rightsquigarrow v_s\}$, $FM' = \{f \in FM \mid \exists\, v_{l_b} \in LM(l_b), v_{l_s} \in LM(l_s) : f \rightsquigarrow v_{l_b} \wedge v_{l_b} \sqsubseteq v_{l_s} : \phi^{\mathcal{S}}(v_s)\}$.

Example 11. With regards to the traditional slice query "slice on customers in the city of Odense", in *s-slice*, the user could specify a geometry extent (e.g., polygon coordinates of the city of Odense) as spatial literal for slicing instead of giving a text literal (e.g., "Odense"). So the s-slice query would be; "slice on customers of the city, which has the geometry `"POLYGON((10.43951 55.47006, 10.439472 55.470036, 10.439240(...))"`". More intuitively, instead of the specified spatial literal $v_s \in \mathcal{L}_s$, the user can pass a function call as parameter to s-slice, e.g., by querying "slice on customers in the largest city of (southern) Denmark by land area". The function call should calculate the area of the cities by their geometries where the largest city is selected as a requirement of the s-slice operator. Both cases are given in the following.

1. The following SPARQL query shows an s-slice operator, which filters 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:broader ?city .
   ?city gnw:cityGeo ?cityGeo .
FILTER (?cityGeo = "POLYGON((10.439517 55.470064,
10.4394729 55.4700361, 10.4392403 (...))") }
```

2. The following SPARQL query shows the s-slice operator, which filters with the function call (largest city) returned from inner select. Given the current limitations of SPARQL, there is not an area calculation function from the geometries of the spatial objects during query run time, however we give the query with a notional built-in `bif:st_area` function.

```
SELECT ?obs WHERE {
   ?obs rdf:type qb:Observation ;
      gnw:customerID ?cust .
   ?cust qb4o:memberOf gnw:customer ;
      skos:broader ?city .
   ?city gnw:cityGeo ?cityGeo .
# Inner select for finding the largest city
   {SELECT ?x (MAX(?area) as ?maxArea)
   WHERE {
      ?obs rdf:type qb:Observation ;
         gnw:customerID ?cust .
      ?cust qb4o:memberOf gnw:customer ;
         skos:broader ?city .
      ?city gnw:cityGeo ?x .
   BIND(bif:st_area(?x) as ?area)}
FILTER (?cityGeo = ?x) }
```

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:broader ?city .
   ?city gnw:cityGeo ?cityGeo .
FILTER (bif:st_within("POINT(10.43951 55.47006)",
?cityGeo)) }
```

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 ($\mathcal{T}_{rel}$) class in order to verify and filter that city.

Remark 18. (Dice) The traditional *dice* operator takes a cube and a Boolean condition ϕ, which returns a new cube containing only the cells that satisfy the Boolean condition ϕ. 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 $\mathcal{C}$ as an argument, which has the cube schema $\mathcal{CS} = (D, M, F)$ 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 $\phi^{\mathcal{S}}$. The s-dice operator keeps the cells of the cube $\mathcal{C}$ 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:
$\mathcal{SD}(\mathcal{C})[\phi^{\mathcal{S}}] = \mathcal{C}'$ where spatial predicate $\phi^{\mathcal{S}}$ can be applied on spatial level member values $\phi^{\mathcal{S}}(v_{l_s})$, spatial attribute values $\phi^{\mathcal{S}}(v_{a_s})$, measure values $\phi^{\mathcal{S}}(v_m)$ and/or a combination of these.

$\mathcal{SD}$ operator returns a sub cube $\mathcal{C}' \subseteq \mathcal{C}$, which has the schema $\mathcal{CS} = (D', M', F')$ where $D' = D$, $M' = M$, and $F' = F$. Unlike the s-slice operator, s-dice keeps all the dimensions D in the output cube $\mathcal{C}'$. The set of measures M and the fact type F also remains the same, though the new cube $\mathcal{C}'$ is a subset of the original cube $\mathcal{C}$ 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 $\phi^{\mathcal{S}}$ on level members as follows;

1. Spatial predicate on level values: $FM' = \{f \in FM \mid \exists\, v_{l_b} \in LM(l_b), v_{l_s} \in LM(l_s) : f \rightsquigarrow v_{l_b} \wedge v_{l_b} \sqsubseteq v_{l_s} \wedge \phi^{\mathcal{S}}(v_{l_s})\}$.
2. Spatial predicate on level attribute values: $FM' = \{f \in FM \mid \exists\, v_{l_b} \in LM(l_b), v_{l_s} \in LM(l_s) \wedge v_{l_s} \rightsquigarrow v_{a_s} : f \rightsquigarrow v_{l_b} \wedge v_{l_b} \sqsubseteq v_{l_s} \wedge v_{l_s} \rightsquigarrow v_{a_s} \wedge \phi^{\mathcal{S}}(v_{a_s})\}$.
   Note that the filtering the facts through level members can be done by v_{l_s} (level values) or attribute values v_{a_s} by applying the spatial predicate $\phi^{\mathcal{S}}$. Finally filtering of the facts is on associated measure values is defined in the following;
3. Spatial predicate on measure values of m_s: $FM' = \{f \in FM \mid \exists\, v_{m_s} \in Codomain(m_s) : f \rightsquigarrow v_{m_s} \wedge \phi^{\mathcal{S}}(v_{m_s})\}$.

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 $\phi^{\mathcal{S}}$ 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 $\phi^{\mathcal{S}}$ can be interpreted in two different ways (See Appendix A1 for comparison of their query run times).

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 $\phi^{\mathcal{S}} \in \mathcal{T}_{rel}$ 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 ;
      gnw:customerID ?cust .
   ?cust qb4o:memberOf gnw:customer ;
      skos:broader ?city ;
      gnw:customerGeo ?custGeo .
   ?city gnw:cityGeo ?cityCentGeo .
FILTER (bif:st_within (?custGeo, ?cityCentGeo,
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 $\mathtt{f}^{\mathcal{S}} \in \mathcal{N}_{op}$. In this case the spatial predicate $\phi^{\mathcal{S}}$ is a combination of a spatial function $\mathtt{f}^{\mathcal{S}}$ and a regular Boolean predicate ϕ. 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 rdf:type qb:Observation ;
      gnw:customerID ?cust .
   ?cust qb4o:memberOf gnw:customer ;
      skos:broader ?city ;
      gnw:customerGeo ?custGeo .
   ?city gnw:cityGeo ?cityCentGeo .
BIND (bif:st_distance (?custGeo, ?cityCentGeo)
AS ?distance) FILTER ( ?distance < 5 ) }
```

Remark 19. (Roll-up) The traditional *roll-up* operator aggregates measures according to a dimension hi-

erarchy (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 $customer \sqsubseteq city$, 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 $\mathtt{f}^{\mathcal{S}} \in \mathcal{S}$ (Sect. 3.1) 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 $\mathtt{f}^{\mathcal{S}}$. 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 $l_b \sqsubseteq_d l'_s$. The instances of the target level l'_s are obtained by the spatial function $\mathtt{f}^{\mathcal{S}}(l_s)$ 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 $\mathcal{CS} = (D, M, F)$ with the fact members $f \in FM$ as given in Def. 16. As a parameter s-roll-up takes a spatial function $\mathtt{f}^{\mathcal{S}} \in \mathcal{S}$ 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; $\mathcal{SRU}(\mathcal{C})[\mathtt{f}^{\mathcal{S}}(L(d_i)), agg(m)] = \mathcal{C}'$, which returns a cube $\mathcal{C}'$ with n-dimensions and has the schema $\mathcal{CS} = (D', M', F')$ where $F' = F$, $M' = M$, and $D' = \{d_i \in D \mid \{d_1, \ldots, d'_i, \ldots, d_n\} \wedge L'(d'_i) = L(d_i) \setminus (l_b \sqsubseteq_d \ldots \sqsubset_d l_s) \cup \{l'_s\}\}$. After the s-roll-up operation, number of dimensions in D remains the same, although the base levels and levels below the target level $(l_b \sqsubseteq_d \ldots \sqsubset_d l_s)$ 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 $LM(l'_s) = \{\mathtt{f}^{\mathcal{S}}(v_{l_b}) \mid v_{l_b} \in LM(l_b)\}$ where $l_b \sqsubseteq_d l'_s \iff \mathtt{f}^{\mathcal{S}}(v_{l_b}) = v_{l'_s}$, which means that the base level l_b rolls up along the spatial dynamic hierarchy ($\sqsubseteq_d$) to the target new spatial level l'_s if and only if spatial function on base level $\mathtt{f}^{\mathcal{S}}(v_{l_b}) = v_{l'_s}$ produces the new spatial level members $v_{l'_s}$. 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 [28] is used to group the facts characterized by the same level members $v_{l'_s} \in LM(l'_s)$ such that $Group(v_{l'_s}) = \{f \in FM \mid \exists\, v_{l_b} \in LM(l_b) : f \rightsquigarrow v_{l_b} \wedge v_{l_b} \sqsubseteq_d v_{l'_s}\}$. 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 $f_i \rightsquigarrow v_{m_i},\ i = 1, \ldots, k$. 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; $FM' = \{f' = Group(v_{l'_s}) \mid \exists\, v_{l'_s} \in LM(l'_s) : f' \rightsquigarrow v_{l'_s} \wedge f' \rightsquigarrow agg(Group(v_{l'_s}))\}$.

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. 3.6. 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 $\mathtt{f}^{\mathcal{S}}$ (*distance* from numeric operations $\mathcal{N}_{op} \subset \mathcal{S}$), 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 rdf:type qb:Observation ;
   gnw:customerID ?cust ;
   gnw:supplierID ?sup;gnw:salesAmount ?sales .
?cust qb4o:memberOf gnw:customer ;
   gnw:customerGeo ?custGeo ;
   gnw:customerName ?custName ;
   skos:broader ?city .
?city qb4o:memberOf gnw:city .
?sup gnw:supplierGeo ?supGeo .
# Inner Select for the total sales to
# the closest supplier of the customer
   { SELECT ?cust1 (MIN(?distance) AS
   ?minDistance) WHERE
   { ?obs rdf:type qb:Observation ;
```

```
      gnw:customerID ?cust1 ;
      gnw:supplierID ?sup1 .
   ?sup1 gnw:supplierGeo ?sup1Geo .
   ?cust1 gnw:customerGeo ?cust1Geo .
   BIND (bif:st_distance( ?cust1Geo, ?sup1Geo )
   AS ?distance)}
   GROUP BY ?cust1 }
   FILTER (?cust = ?cust1 && bif:st_distance
   (?custGeo, ?supGeo) = ?minDistance)}
GROUP BY ?city
```

Remark 20. (Drill-down) Drill-down is the inverse operator of roll-up, which disaggregates 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 19 ("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 $City \sqsupseteq Supplier$). Conceptually, a drill-down to level l_i on a cube $\mathcal{C}$ corresponds to a roll-up to the same level l_i on the *base cube* of $\mathcal{C}$, that is denoted as $BaseCube(\mathcal{C})$.

Definition 20. (S-Drill-down) Analogously to drill-down operator, *s-drill-down* disaggregates measures $m \in M$ of a given cube $\mathcal{C}$, by using an aggregate function and a spatial function $\mathtt{f}^{\mathcal{S}}$ (Sect. 3.1) 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 $\mathtt{f}^{\mathcal{S}}$ 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_s} to a dynamically created spatial child level l'_{c_s} such that $l_{p_s} \sqsupseteq_d l'_{c_s}$. The target spatial child level l'_{c_s} is the output of the spatial function $\mathtt{f}^{\mathcal{S}}$ on spatial levels $l_{i_s} \in L(d_i)$ of the spatial dimension. Applying s-drill-down to child level l'_{c_s} from a parent level l_{p_s} on a cube $\mathcal{C}$ corresponds to applying s-roll-up to the same level l'_{c_s} from the base level l_b on the *base cube* of $\mathcal{C}$. Therefore, the semantics of the s-drill-down is described same as s-roll-up and the operator is formulated as $SDD(\mathcal{C})[\mathtt{f}^{\mathcal{S}}(L(d_i)), agg(m)] = SRU(BaseCube(\mathcal{C}))[\mathtt{f}^{\mathcal{S}}(L(d_i)), agg(m)]$.

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 the from City level to Supplier level". The following SPARQL query shows the given example.

```
SELECT ?sup (AVG(?sales) AS ?averageSales)
   (MAX(?distance) AS ?maxDistance)
WHERE { ?obs rdf:type qb:Observation ;
   gnw:supplierID ?sup ;
   gnw:salesAmount ?sales .
?sup qb4o:memberOf gnw:supplier ;
   gnw:supplierGeo ?supGeo ;
   gnw:supplierName ?supName ;
   skos:broader ?city .
?city qb4o:memberOf gnw:city ;
   gnw:cityGeo ?cityCentGeo .
BIND (bif:st_distance(?supGeo, ?cityCentGeo)
   AS ?distance)}
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* [5] is out of scope and remained as future work.

6. Generating SOLAP queries in SPARQL via QB4SOLAP

After having defined the high-level SOLAP operators in Sect. 5, this section first describes how to generate SPARQL queries for each of these operators by using the QB4SOLAP metamodel (Sect. 4). Afterwards, this section describes how to create more complex SPARQL queries for nested SOLAP operations.

6.1. Generation algorithms

The generated SPARQL queries Q are of the form "$Q = \texttt{SELECT}\ R\ \texttt{WHERE}\ GP$", 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 $\mathcal{G}$, a graph pattern GP is used to search for subgraphs $\mathcal{G}_{(R)} \subseteq \mathcal{G}$ 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 `WHERE` clause: $GP = GP \cup (s\ p\ o)$.

RDF datasets published with the QB4SOLAP vocabulary use the `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 `WHERE` clause.

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

Algorithm 1: $RUPath(\mathcal{G}^S_{(\mathcal{C})}, l_b, l_s, a_{ID}, ?\mathrm{a}_s, ?\mathrm{f}) : GP$

Input: $\mathcal{G}^S_{(\mathcal{C})}, l_b, l_s, a_{ID}, ?\mathrm{a}_s, ?\mathrm{f}$
Output: GP

```
begin
    GP = (?f rdf:type qb:Observation)
    GP = GP ∪ (?f id^S(a_ID) ?l_b ∧ ?l_b qb4o:memberOf id^S(l_b))
    foreach (id^S(l_c), id^S(l_p)) ∈ G^S_(C) | l_p ⊑ l_s do
        GP = GP ∪ (?l_b skos:broader ?l_p ∧ ?l_p skos:broader ?l_s)
    let GP = GP ∪ (?l_s id^S(a_s) ?a_s)
return GP
```

Build roll-up path ($RUPath$). The helper function $RUPath$ returns a graph pattern that we can use in the body of the `WHERE` clause. The roll-up path pattern is created as a path-shaped join of triples with partial order ($\sqsubseteq$, `skos:broader`) (Def.s 10 and 15). The triple pattern is of the form $\{(s_1\ p_1\ o_1), (o_1\ p_2\ o_2), (o_2\ p_2\ o_3), \ldots (o_{n-1}\ p_2\ o_n)\}$ where s_1 is the root of the graph pattern and corresponds to fact members f from the QB4SOLAP schema (Def. 16), p_1 is the predicate $id^S(a_{ID})$ that associates facts with level members v_{l_i} ($f \rightsquigarrow v_{l_i}$, Def. 16), o_1 is the variable for the first level member that rolls up to its parent level o_2 such that $o_1 \sqsubseteq o_2$ and so on, and the p_2 predicate corresponds to the `skos:broader` property. The last variable in the path o_n corresponds to the target level 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 `FILTER` expression to restrict the output data by using a (spatial) Boolean predicate $\phi^{\mathcal{S}}$. A `FILTER` expression is part of the `WHERE` clause in a SPARQL query. Therefore, it is added to the body of the `WHERE` clause in the graph pattern GP as $GP = GP \cup (\texttt{FILTER}\ \phi^{\mathcal{S}})$. In the cases where there is a spatial function $\mathtt{f}^{\mathcal{S}}(x)$ in the SOLAP operator, it is given in the `BIND` clause, which is technically a part of the `WHERE` clause and therefore added to the body of the `WHERE` clause in a graph pattern GP as $GP = GP \cup (\texttt{BIND}\ \mathtt{f}^{\mathcal{S}}(x))$. SPARQL 1.1 defines aggregate expressions[15], 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. 5. 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 $\mathcal{SS}(\mathcal{C})[l_b, l_s, v_s]$ (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 a_s (Ext. 8) at a spatial level l_s. Similarly, the spatial function call $\mathtt{f}^{\mathcal{S}}(x)$ in Ex. 11.2 returns a polygon that corresponds to a spatial level attribute a_s.

[15] `https://www.w3.org/TR/sparql11-query/#aggregates`

Algorithm 2: *S–SliceGenerator* $(\mathcal{G}^I_{(\mathcal{C})}, v_s, l_b, l_s) : Q$

Input: $\mathcal{G}^I_{(\mathcal{C})}$, v_s, l_b, l_s
Output: Q

1 **begin**
2 $Q = \emptyset$; GP = RUPath ($\mathcal{G}^S_{(\mathcal{C})}, l_b, l_s, a_{ID}$,?a$_s$,?f)
3 **if** v_s *is a* POINT **then**
4 $GP = GP \cup$ (FILTER(st_within v_s, ?a$_s$))
5 **else if** $v_s = \mathtt{f}^{\mathcal{S}}(x)$ **then**
6 $Q' = \emptyset$; GP' =RUPath ($\mathcal{G}^S_{(\mathcal{C})}, l_b, l_s, a_{ID}$,?a$_s$,?f)
7 GP' = $GP' \cup$ (BIND $\mathtt{f}^{\mathcal{S}}(x)$ AS ?v$_x$)
8 Q' = SELECT ?x AGG(?v$_x$) WHERE GP'
9 $GP = GP \cup Q' \cup$ (FILTER ?x = ?a$_s$)
10 **else**
11 $GP = GP \cup$ (FILTER v_s = ?a$_s$)
12 **return** Q = SELECT ?f WHERE GP

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 ($\mathcal{T}_{rel}$). 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.

Line 2. Get the path for dimension d_s (e.g., Customer) from the observation facts f to the base level l_b, and build path-shaped triple pattern paths from the dimension's base level l_b to the target spatial level l_s (e.g., City level). Finally, get level attribute IRIs and variables for the spatial attributes a_s (e.g., City geometry). All this is done by the RUPath function (Algorithm 1) that is used by the s-slice generator. The following shows an example result of this step that is added to GP:

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

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 $\mathtt{f}^{\mathcal{S}}(x)$. If true (Ex. 11.2), construct an inner select query to compute the spatial function $\mathtt{f}^{\mathcal{S}}(x)$, 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 ?x .
```

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':

```
BIND (bif:st_area(?x) as ?area) }
```

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

Line 9. Build the filter statement with the output of the spatial function $\mathtt{f}^{\mathcal{S}}(x)$, 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:

```
FILTER (?cityGeo = ?x) }
```

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

```
    FILTER (?cityGeo = "POLYGON((10.43951
    55.47006, 10.439472 55.470036,
    10.439240 (...))")}
```

Line 12. Finally, the algorithm generates query Q, which can be executed over the fact members FM'. In our running examples we obtain the following cases for the generated s-slice query Q.

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 `POINT` data type (Ex. 12) and filters the level attributes with a spatial predicate within a given level. The graph pattern GP for the query is created in Lines 2 to 7.

```
1 Q = SELECT ?obs WHERE
2     {?obs rdf:type qb:Observation ;
3        gnw:customerID ?cust .
4     ?cust qb4o:memberOf gnw:customer ;
5        skos:broader ?city .
6     ?city gnw:cityGeo ?cityGeo .
7 FILTER (bif:st_within("POINT(10.43951
  55.47006)", ?cityGeo)) }
```

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.

```
1 Q = SELECT ?obs WHERE
2     { ?obs rdf:type qb:Observation ;
3        gnw:customerID ?cust .
4     ?cust qb4o:memberOf gnw:customer ;
5        skos:broader ?city .
6     ?city gnw:cityGeo ?cityGeo .
  # When there is spatial function call we apply
  # inner select for finding the largest city
7     {SELECT ?x (MAX(?area) as ?maxArea) WHERE
8        { ?obs rdf:type qb:Observation ;
9           gnw:customerID ?cust .
10       ?cust qb4o:memberOf gnw:customer ;
11          skos:broader ?city .
12       ?city gnw:cityGeo ?x .
13    BIND(bif:st_area(?x) as ?area)}
  # Then we apply the filter on the output
  # coming from the spatial function
14 FILTER (?cityGeo = ?x) }
```

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

```
1 Q = SELECT ?obs WHERE
2     {?obs rdf:type qb:Observation ;
3        gnw:customerID ?cust .
4     ?cust qb4o:memberOf gnw:customer ;
5        skos:broader ?city .
6     ?city gnw:cityGeo ?cityGeo .
7 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, $\mathcal{SD}(\mathcal{C})[\phi^{\mathcal{S}}]$ (Def. 18 – parameter $\phi^{\mathcal{S}}$ represents a spatial predicate), we follow the steps sketched in Algorithm 3. The algorithm takes parameter $\phi^{\mathcal{S}}$ as input, which corresponds to a spatial predicate that could represent a topological relation from the $\mathcal{T}_{rel}$ set or a combination of a spatial function (a numeric operation from the $\mathcal{N}_{op}$ set) and a regular predicate ϕ. For illustration, we use the example query that we have introduced in Sect. 5 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 $\phi^{\mathcal{S}}$

Line 4. Build the roll-up path for those levels using the helper function `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 `RUPath` is added to graph pattern GP:

```
{?obs rdf:type qb:Observation ;
    gnw:customerID ?cust .
?cust qb4o:memberOf gnw:customer ;
    skos:broader ?city ;
    gnw:customerGeo ?custGeo .
?city gnw:cityGeo ?cityCentGeo .
```

Line 5. Check if $\phi^{\mathcal{S}}$ is to be implemented as a spatial predicate from topological relations $\mathcal{T}_{rel}$ 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 `st_within` is used from the topological relations. The following lines are added to graph pattern GP:

```
FILTER (bif:st_within(?custGeo,
?cityCentGeo, 5))}
```

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

Algorithm 3: *S–DiceGenerator* $(\mathcal{G}^{I}_{(\mathcal{C})}, \phi^{\mathcal{S}}) : Q$

Input: $\mathcal{G}^{I}_{(\mathcal{C})}$, $\phi^{\mathcal{S}}$
Output: Q

```
1  begin
2      Q = ∅ ; GP = ∅
3      for l_b, l_s ∈ φ^S do
4          GP = GP ∪
           RUPath (G^S_(C), l_b, l_s, a_ID, ?a_s, ?f)
5      if φ^S is a spatial predicate then
6          GP = GP ∪ (FILTER φ^S(?a_s))
7      else if φ^S uses a spatial function f^S(x) and a
       regular Boolean predicate φ then
8          GP = GP ∪ (BIND f^S(x) AS ?v_x)
9          GP = GP ∪ (FILTER φ(?v_x))
10 return Q = SELECT ?f WHERE GP
```

Lines 8, 9. Create a bind statement based on a spatial function (i.e., calculate `st_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:

```
BIND (bif:st_distance (?custGeo, ?cityCentGeo)
   AS ?distance) 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.

S-Dice operator with $\phi^{\mathcal{S}}$: 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.

```
1 Q = SELECT ?obs WHERE
2     { ?obs rdf:type qb:Observation ;
3        gnw:customerID ?cust .
4     ?cust qb4o:memberOf gnw:customer ;
5        skos:broader ?city ;
6        gnw:customerGeo ?custGeo .
7     ?city gnw:cityGeo ?cityCentGeo .
8 FILTER (bif:st_within (?cityCentGeo, ?custGeo, 5))}
```

S-Dice operator with $\mathtt{f}^{\mathcal{S}}(x)$ and a regular Boolean predicate ϕ: 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.

```
1 Q = SELECT ?obs WHERE
2     { ?obs rdf:type qb:Observation ;
3        gnw:customerID ?cust .
4     ?cust qb4o:memberOf gnw:customer ;
5        skos:broader ?city ;
6        gnw:customerGeo ?custGeo .
7     ?city gnw:cityGeo ?cityCentGeo .
8 BIND (bif:st_distance (?custGeo, ?cityCentGeo) AS
9     ?distance) FILTER ( ?distance < 5 ) }
```

S-Roll-up Generator. To generate a SPARQL query for the s-roll-up operator from a high-level SOLAP expression, $\mathcal{SRU}(\mathcal{C})[\mathtt{f}^{\mathcal{S}}(L(d_i)), agg(m)]$ (Def. 19), where parameter $\mathtt{f}^{\mathcal{S}}(L(d_i))$ 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-roll-up given in Sect. 5 for s-roll-up (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 rdf:type qb:Observation ;
    gnw:customerID ?cust ;
    gnw:supplierID ?sup ;
    gnw:salesAmount ?sales .
?cust qb4o:memberOf gnw:customer ;
    gnw:customerGeo ?custGeo ;
    skos:broader ?city .
?sup qb4o:memberOf gnw:supplier ;
    gnw:supplierGeo ?supGeo ;
    skos:broader ?city .
?city qb4o:memberOf gnw:city .
```

Line 4. Build inner select subquery to apply the spatial function $\mathtt{f}^{\mathcal{S}}$ 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.

Algorithm 4: $SRUGenerator(\mathcal{G}^I_{(\mathcal{C})}, \mathtt{f}^{\mathcal{S}}(L(d_s)), agg(m)) : Q$

Input: $\mathcal{G}^I_{(\mathcal{C})}, \mathtt{f}^{\mathcal{S}}(L(d_i)), agg(m)$
Output: Q

1 **begin**
2 $Q = \emptyset$; GP = RUPath ($\mathcal{G}^S_{(\mathcal{C})}, l_b, l_s, a_{ID}$,?a$_s$,?f)
3 $GP = GP \cup$ (?f $id^S(m)$?m)
4 **for** $\mathtt{f}^{\mathcal{S}}(L(d_s))$ **do**
5 $GP' =$ RUPath ($\mathcal{G}^S_{(\mathcal{C})}, l_b, l_s, a_{ID}$,?a$_s$,?f) ; $Q' = \emptyset$
6 $GP' = GP' \cup$ (BIND $\mathtt{f}^{\mathcal{S}}(x)$ AS ?v$_x$)
7 Q' = SELECT ?x (AGG(?v$_x$) AS ?v$_y$) WHERE GP' GROUP BY ?x
8 $GP = GP \cup Q' \cup$ (FILTER ?x = ?a$_s$ && $\mathtt{f}^{\mathcal{S}}(x)$ = ?v$_y$)
9 let $l_s = l'_s$
10 **return** Q = SELECT ?f ?l'$_s$ AGG(?m) WHERE GP GROUP BY ?f ?l'$_s$

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 rdf:type qb:Observation ;
    gnw:customerID ?cust1 ;
    gnw:supplierID ?sup1 .
?sup1 gnw:supplierGeo ?sup1Geo .
?cust1 gnw:customerGeo ?cust1Geo .
```

Line 6. Build the bind statement in order to calculate the spatial function $\mathtt{f}^{\mathcal{S}}(L(d_s))$ 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)}
```

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 $\mathtt{f}^{\mathcal{S}}(L(d_i))$ 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 15 to 22 and the graph pattern GP for the whole query is created in Lines 3 to 24.

```
1  Q = SELECT ?obs ?city (SUM(?sales) AS
2  ?totalSales) WHERE
3  { ?obs rdf:type qb:Observation ;
4      gnw:customerID ?cust ;
5      gnw:supplierID ?sup ;
6      gnw:salesAmount ?sales .
7  ?cust qb4o:memberOf gnw:customer ;
8      gnw:customerGeo ?custGeo ;
9      gnw:customerName ?custName ;
10     skos:broader ?city .
11 ?city qb4o:memberOf gnw:city .
12 ?sup gnw:supplierGeo ?supGeo .
   # Inner Select for the distance function
13     { SELECT ?cust1 (MIN(?distance) AS
14     ?minDistance) WHERE
15     { ?obs rdf:type qb:Observation ;
16        gnw:customerID ?cust1 ;
```

```
17        gnw:supplierID ?sup1 .
18     ?sup1 gnw:supplierGeo ?sup1Geo .
19     ?cust1 gnw:customerGeo ?cust1Geo .
20     BIND (bif:st_distance( ?cust1Geo, ?sup1Geo )
21     AS ?distance)}
22     GROUP BY ?cust1 }
23     FILTER (?cust = ?cust1 && bif:st_distance
24     (?custGeo, ?supGeo) = ?minDistance)}
25 GROUP BY ?city ?obs
```

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 $\mathcal{C}$ for s-roll-up is obtained using a function $BaseCube$ such that $SDD(\mathcal{C})[\mathtt{f}^{\mathcal{S}}(L(d_i)), agg(m)] = SRU(BaseCube(\mathcal{C}))[\mathtt{f}^{\mathcal{S}}(L(d_i)), agg(m)]$ (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` $(Base(\mathcal{G}^I_{(\mathcal{C})}), \mathtt{f}^{\mathcal{S}}(L(d_i)), agg(m))$.

6.2. 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$_1$(... s-roll-up$_k$(s-slice$_1$(... s-slice$_n$(s-dice$_1$(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*$((\mathcal{SRU}(\mathcal{C})[\mathtt{f}^{\mathcal{S}}(L(d_i)), agg\ (m)]\ (\mathcal{SS}(\mathcal{C})[l_b, l_s, v_{a_s}](\mathcal{SD}(\mathcal{C})[\phi^{\mathcal{S}}]))))$: Q

Input: $(\mathcal{SRU}(\mathcal{C})[\mathtt{f}^{\mathcal{S}}(L(d_s)), agg(m)](\mathcal{SS}(\mathcal{C})[l_b, l_s, v_{a_s}](\mathcal{SD}(\mathcal{C})[\phi^{\mathcal{S}}])))$
Output: Q

```
1  begin
2      Q = ∅ ; GP = RUPath (G^S_(C), l_b, l_s, a_ID, ?a_s, ?f)
3      GP = GP ∪ (?f id^S(m) ?m )
4      GP^1 = S-DiceGenerator(G^I_(C), φ^S) \FILTER^1 \ SELECT
5      GP = GP ∪ GP^1
6      GP^2 = S-SliceGenerator(G^I_(C), v_s, l_b, l_s) \FILTER^2 \ SELECT
7      GP = GP ∪ GP^2
8      GP = GP ∪ FILTER^1 ∪ FILTER^2 ∪
9      SRUGenerator(G^I_(C), f^S(L(d_s)), agg(m)) \
       SELECT \ GROUP BY^1 \ AGG^1
10 return Q = SELECT ?l'_s AGG^1(?m) WHERE GP
   GROUP BY^1 ?l'_s
```

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

Example 16. *((^{3}s-roll-up (^{2}s-slice (^{1}s-dice($\mathcal{C}$)))))*: ^{1}Get the subcube graph of customer 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 leave out all the other countries) and ^{3}get the total amount of sales of 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 s-dice to the outermost operator s-roll-up.

```
1  SELECT ?city (SUM(?sales) AS ?totalSales)
2  WHERE {
3      ?obs rdf:type qb:Observation ;
4         gnw:customerID ?cust ;
5         gnw:supplierID ?sup ;
6         gnw:salesAmount ?sales .
7      ?cust qb4o:memberOf gnw:customer ;
8         gnw:customerGeo ?custGeo ;
9         skos:broader ?city .
10     ?sup qb4o:memberOf gnw:supplier ;
11        gnw:supplierGeo ?supGeo ;
12        skos:broader ?city .
13     ?city qb4o:memberOf gnw:city ;
14        gnw:cityGeo ?cityCentGeo ;
15        skos:broader ?country .
16     ?country qb4o:memberOf gnw:country ;
17        gnw:countryGeo ?countGeo .
18     ?city gnw:cityGeo ?cityGeo .
   ### 1.Inner select for (S-SLICE)
   ### Find the largest country
19     {SELECT ?x (MAX(?area) as ?maxArea)
20     WHERE {
21        ?obs rdf:type qb:Observation ;
22           gnw:customerID ?cust .
23        ?cust qb4o:memberOf gnw:customer ;
24           skos:broader ?city .
25        ?city skos:broader ?country .
26        ?country gnw:countryGeo ?x .
27     BIND(bif:st_area(?x) as ?area)}}
   ### 2.Inner select for (S-ROLL-UP)
   ### Find the closest suppliers to customers
28     { SELECT ?cust1 (MIN(?distance) AS
29     ?minDistance) WHERE {
30        ?obs rdf:type qb:Observation ;
31           gnw:customerID ?cust1 ;
32           gnw:supplierID ?sup1 .
33        ?sup1 gnw:supplierGeo ?sup1Geo .
34        ?cust1 gnw:customerGeo ?cust1Geo .
35     BIND (bif:st_distance(?cust1Geo, ?sup1Geo)
       AS ?distance)}
36     GROUP BY ?cust1 }
   ### FILTER for S-DICE, to get a subcube
37 FILTER (bif:st_within (?custGeo, ?cityCentGeo, 5))
   ### FILTER for S-SLICE, the 1st inner SELECT
38 FILTER (?countGeo = ?x)
   ### FILTER for S-ROLL-UP, the 2nd inner SELECT
39 FILTER (?cust = ?cust1 && bif:st_distance
   (?custGeo, ?supGeo) = ?minDistance)}
40 GROUP BY ?city
```

The graph pattern GP is initially created with the RUPath function for the corresponding levels and level attributes (Lines 3 to 18 of the generated SPARQL query in the listing above). The first operator is called by function S-DiceGenerator, where the first 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 FILTER clause (Lines 19 to 27), which is followed by the SRUGenerator function without GROUP BY and AGG statements (Lines 28 to 36). Note that in Line 36, GROUP BY is applied on the lower level Customer, and the actual GROUP BY for the target City level is applied in the last line (Line 40). Separated 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).

7. Conclusions and future work

Motivated by the need for a formal foundation for spatial data warehouses on the Semantic Web, this pa-

per made a number of contributions. First, it proposed 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.

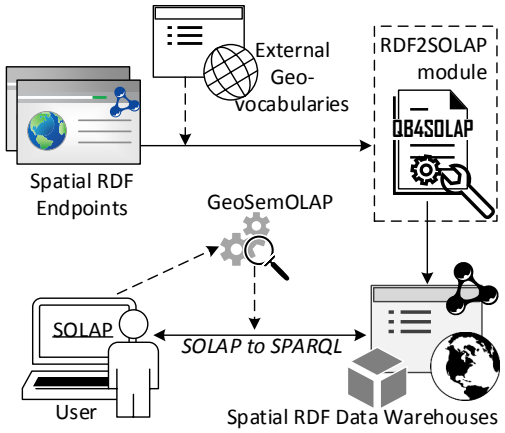


Fig. 8. Our future vision of SOLAP on the SW

Fig. 8 presents our future vision of SOLAP on the Semantic Web with regards to the current state of the art and ongoing work. In order to verify the algorithms, which are defined in this paper, we developed *GeoSemOLAP* [18], a SPARQL query generation tool. GeoSemOLAP allows users to perform SOLAP operations and generate SPARQL queries interactively through a GUI. We refer to our screencast[16] for a detailed demonstration of GeoSemOLAP.

Publishing spatial DWs on the SW allows users to also exploit the existing external geo-vocabularies (e.g., GeoNames, etc.)[17] by defining spatial levels and hierarchies from external open data sources. Our main ongoing work thus focuses on developing an RDF2SOLAP enrichment module that performs the multidimensional annotation of existing spatial RDF datasets with QB4SOLAP in a (semi-)automatic fashion.

Additional interesting aspects of future work would be, for instance, 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*, and supporting spatial aggregation (*s-aggregation*) with user-defined functions over spatial measures. It would be also interesting to increase efficiency by extending our spatial data warehouse with techniques that been developed in the context of RDF data cubes and SPARQL analytical queries in general, e.g., materialization and optimizing the physical layout [13,21,22], and to enable efficient support of a broad range of external sources by considering aspects such as federated processing of analytical queries [20] and schema heterogeneity [35]. We will also consider more efficient representations of the data, e.g., by removing redundancies. Furthermore, it would be interesting to extend QB4SOLAP and GeoSemOLAP [18] to handle highly dynamic spatio-temporal data and queries, as for instance, found in large-scale transport analytics [12].

Acknowledgment. This research is partially funded by the European Commission through the Erasmus Mundus Joint Doctorate Information Technologies for Business Intelligence (EM IT4BI-DC) and the Danish Council for Independent Research (DFF) under grant agreement no. DFF-4093-00301.

References

[1] A. Abelló, O. Romero, T. Pedersen, R. Berlanga Llavori, V. Nebot, M. Aramburu, and A. Simitsis. Using Semantic Web Technologies for Exploratory OLAP: A Survey. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 27(2):571–588, 2014. https://doi.org/10.1109/TKDE.2014.2330822.

[2] A. B. Andersen, N. Gür, K. Hose, K. A. Jakobsen, and T. B. Pedersen. Publishing Danish Agricultural Government Data as Semantic Web Data. In *Semantic Technology: 4th Joint International Semantic Technology Conference (JIST'14)*, volume 8943, pages 178–186. Springer, 2014. https://dx.doi.org/10.1007/978-3-319-15615-6_13.

[3] R. Battle and D. Kolas. Enabling the Geospatial Semantic Web with Parliament and GeoSPARQL. *Semantic Web Journal (SWJ)*, 3(4):355–370, 2012. https://dx.doi.org/10.3233/SW-2012-0065.

[4] Y. Bédard, E. Bernier, S. Larrivée, M. Nadeau, M. Proulx, and S. Rivest. Spatial OLAP. In *Forum annuel sur la RD, Géomatique VI: Un monde accessible*, pages 13–14, 1997.

[16] https://youtu.be/Pc3RBPPgBhA

[17] GeoNames: http://www.geonames.org/
Global Administrative Areas: http://gadm.geovocab.org/
NUTS – EU's Nomenclature of Territorial Units for Statistics: http://nuts.geovocab.org/

[5] C. Ciferri, L. Gómez, M. Schneider, A. Vaisman, and E. Zimányi. Cube algebra: A Generic User-centric Model and Query Language for OLAP Cubes. *International Journal of Data Warehousing and Mining (IJDWM)*, 9(2):39–65, 2013. http://dx.doi.org/10.4018/jdwm.2013040103.

[6] R. Cyganiak, D. Reynolds, and J. Tennison. The RDF Data Cube Vocabulary. *W3C Recommendation*, 2014. https://www.w3.org/TR/vocab-data-cube/.

[7] J. da Silva, V. C. Times, A. C. Salgado, C. Souza, R. d. N. Fidalgo, and A. G. de Oliveira. A set of aggregation functions for spatial measures. In *Proceedings of the 11th International Workshop on Data Warehousing and OLAP (DOLAP'08)*, pages 25–32. ACM, 2008. https://doi.acm.org/10.1145/1458432.1458438.

[8] R. P. Deb Nath, K. Hose, and T. B. Pedersen. Towards a Programmable Semantic Extract-Transform-Load Framework for Semantic Data Warehouses. In *Proceedings of the 18th International Workshop on Data Warehousing and OLAP (DOLAP'15)*, pages 15–24. ACM, 2015. https://doi.org/10.1145/2811222.2811229.

[9] E. Edoh-Alove, S. Bimonte, and F. Pinet. An UML Profile and SOLAP Datacubes Multidimensional Schemas Transformation Process for Datacubes Risk-Aware Design. *International Journal of Data Warehousing and Mining (IJDWM)*, 11(4):64–83, 2015. https://dx.doi.org/10.4018/ijdwm.2015100104.

[10] M. J. Egenhofer and J. Herring. A mathematical framework for the definition of topological relationships. In *Fourth international symposium on spatial data handling*, pages 803–813. Zurich, Switzerland, 1990.

[11] L. Etcheverry, A. Vaisman, and E. Zimányi. Modeling and Querying Data Warehouses on the Semantic Web using QB4OLAP. In *Data Warehousing and Knowledge Discovery (DaWaK'14)*, volume 8646, pages 45–56. Springer, 2014. https://dx.doi.org/10.1007/978-3-319-10160-6_5.

[12] G. Gidofalvi, T. B. Pedersen, T. Risch, and E. Zeitler. Highly scalable trip grouping for large-scale collective transportation systems. In *Proceedings of the 11th International Conference on Extending Database Technology (EDBT'08)*, pages 678–689. ACM, 2008. https://doi.acm.org/10.1145/1353343.1353425.

[13] F. Goasdoué, K. Karanasos, J. Leblay, and I. Manolescu. View Selection in Semantic Web Databases. *VLDB Endowment*, 5(2):97–108, 2011. https://dx.doi.org/10.14778/2078324.2078326.

[14] L. I. Gómez, S. A. Gómez, and A. A. Vaisman. A Generic Data Model and Query Language for Spatiotemporal OLAP Cube Analysis. In *Proceedings of the 15th International Conference on Extending Database Technology (EDBT'12)*, pages 300–311. ACM, 2012. https://doi.acm.org/10.1145/2247596.2247632.

[15] L. I. Gómez, S. Haesevoets, B. Kuijpers, and A. A. Vaisman. Spatial aggregation: Data model and implementation. *Information Systems (IS)*, 34(6):551–576, 2009. https://dx.doi.org/10.1016/j.is.2009.03.002.

[16] N. Gür, K. Hose, T. B. Pedersen, and E. Zimányi. Enabling Spatial OLAP over Environmental and Farming Data with QB4SOLAP. In *Semantic Technology: 6th Joint International Semantic Technology Conference (JIST'16)*, volume 10055, pages 287–304. Springer, 2016. https://dx.doi.org/10.1007/978-3-319-50112-3_22.

[17] N. Gür, K. Hose, E. Zimányi, and T. B. Pedersen. Modeling and Querying Spatial Data Warehouses on the Semantic Web. In *Semantic Technology: 5th Joint International Semantic Technology Conference (JIST'15)*, volume 9544, pages 1–20. Springer, 2015. https://dx.doi.org/10.1007/978-3-319-31676-5_1.

[18] N. Gür, J. Nielsen, K. Hose, and T. B. Pedersen. GeoSemOLAP: SOLAP on the Semantic Web Made Easy. In *Proceedings of the 26th International Conference Companion on World Wide Web (WWW'17)*. ACM, 2017. https://dx.doi.org/10.1145/3041021.3054731.

[19] J. Han, N. Stefanovic, and K. Koperski. Selective Materialization: An Efficient Method for Spatial Data Cube Construction. In *Research and Development in Knowledge Discovery and Data Mining (PAKDD'98)*, pages 144–158. Springer, 1998. https://dx.doi.org/10.1007/3-540-64383-4_13.

[20] D. Ibragimov, K. Hose, T. B. Pedersen, and E. Zimányi. Processing Aggregate Queries in a Federation of SPARQL Endpoints. In *The Semantic Web: 12th European Semantic Web Conference (ESWC'15)*, pages 269–285. Springer, 2015. https://dx.doi.org/10.1007/978-3-319-18818-8_17.

[21] D. Ibragimov, K. Hose, T. B. Pedersen, and E. Zimányi. Optimizing Aggregate SPARQL Queries Using Materialized RDF Views. In *The Semantic Web: 15th International Semantic Web Conference (ISWC'16)*, pages 341–359. Springer, 2016. https://dx.doi.org/10.1007/978-3-319-46523-4_21.

[22] K. A. Jakobsen, A. B. Andersen, K. Hose, and T. B. Pedersen. Optimizing RDF Data Cubes for Efficient Processing of Analytical Queries. In *Proceedings of the 6th International Workshop on Consuming Linked Data (COLD'15)*, 2015. http://ceur-ws.org/Vol-1426/paper-02.pdf.

[23] B. Kämpgen, S. O'Riain, and A. Harth. Interacting with Statistical Linked Data via OLAP Operations. In *The Semantic Web: ESWC 2012 Satellite Events*, volume 7540, pages 87–101. Springer, 2012. https://dx.doi.org/10.1007/978-3-662-46641-4_7.

[24] M. Koubarakis, M. Karpathiotakis, K. Kyzirakos, C. Nikolaou, and M. Sioutis. Data Models and Query Languages for Linked Geospatial Data. In *Reasoning Web. Semantic Technologies for Advanced Query Answering*, pages 290–328. Springer, 2012. https://dx.doi.org/10.1007/978-3-642-33158-9_8.

[25] K. Kyzirakos, M. Karpathiotakis, and M. Koubarakis. Strabon: A Semantic Geospatial DBMS. In *The Semantic Web: 11th International Semantic Web Conference (ISWC'12)*, pages 295–311. Springer, 2012. https://dx.doi.org/10.1007/978-3-642-35176-1_19.

[26] I. V. Lopez, R. T. Snodgrass, and B. Moon. Spatiotemporal aggregate computation: A survey. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 17(2):271–286, 2005. https://doi.org/10.1109/TKDE.2005.34.

[27] E. Malinowski and E. Zimányi. *Advanced Data Warehouse Design: From Conventional to Spatial and Temporal Applications. Data-Centric Systems and Applications*. Springer, 2008. https://dx.doi.org/10.1007/978-3-540-74405-4.

[28] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. A Foundation

for Capturing and Querying Complex Multidimensional Data. *Information Systems (IS)*, 26(5):383–423, 2001. http://dx.doi.org/10.1016/S0306-4379(01)00023-0.

[29] T. B. Pedersen and N. Tryfona. Pre-aggregation in Spatial Data Warehouses. In *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases (SSTD'01)*, pages 460–478. Springer, 2001. http://dx.doi.org/10.1007/3-540-47724-1_24.

[30] M. Perry and J. Herring. GeoSPARQL: A Geographic Query Language for RDF Data. *OGC Implementation Standard*, 2012.

[31] D. A. Randell, Z. Cui, and A. G. Cohn. A spatial logic based on regions and connection. In *Principles of Knowledge Representation and Reasoning*, volume 92, pages 165–176, 1992.

[32] P. Revesz. *Introduction to Databases: From Biological to Spatio-Temporal*. Springer, 2009.

[33] S. Rivest, Y. Bédard, and P. Marchand. Toward better support for spatial decision making: defining the characteristics of spatial on-line analytical processing (SOLAP). *GEOMATICA*, 55(4):539–555, 2001. Canadian Institute of Geomatics.

[34] G. Rojas, G. Giannopoulos, and J. J. L. Daniel Hladky. Managing Geospatial Linked Data in the GeoKnow Project. In *The Semantic Web in Earth and Space Science. Current Status and Future Directions*, volume 20, page 51. IOS Press, 2015.

[35] J. Rouces, G. de Melo, and K. Hose. FrameBase: Representing N-Ary Relations Using Semantic Frames. In *The Semantic Web: 12th European Semantic Web Conference (ESWC'15)*, pages 505–521. Springer, 2015. https://dx.doi.org/10.1007/978-3-319-18818-8_31.

[36] C. Stadler, J. Lehmann, K. Höffner, and S. Auer. LinkedGeoData: A Core for a Web of Spatial Open Data. *Semantic Web Journal (SWJ)*, 3:333–354, 2012. https://dx.doi.org/10.3233/SW-2011-0052.

[37] A. Vaisman and E. Zimányi. A Multidimensional Model Representing Continuous Fields in Spatial Data Warehouses. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'09)*, pages 168–177. ACM, 2009. https://doi.acm.org/10.1145/1653771.1653797.

[38] J. Varga, A. A. Vaisman, O. Romero, L. Etcheverry, T. B. Pedersen, and C. Thomsen. Dimensional enrichment of statistical linked open data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 40:22–51, 2016. https://dx.doi.org/10.1016/j.websem.2016.07.003.

Appendix

A1. Query Run Times

Table 5 presents the query runtimes for the SOLAP operator examples (Ex. 12, Ex. 13.1, Ex. 13.2, Ex. 14, and Ex. 15) given in Sect. 5 and a nested SOLAP operator example (Ex. 16) given in Sect. 6.2. The SOLAP queries are tested against an instance of the use case dataset (GeoNorthwind) that we have discussed in Sect. 4. In total, *48677* triples are obtained from the GeoNorthwind dataset. The published RDF graph instance is denoted as $\mathcal{C}$ in Table 5. We used Virtuoso Open Source Edition (Column Store and multi threaded) Version 7.2.5 on an Ubuntu 14.04 server with 2.30GHz CPU and 16 GB RAM to publish the triples at the SPARQL endpoint http://lod.cs.aau.dk:8890/sparql. The actual queries used in the paper are available at http://extbi.cs.aau.dk/SOLAP4SW/queries.

Table 5
Runtimes in seconds

SOLAP Operators	**Query Runtime**
Ex. 12 *(s-slice($\mathcal{C}$))*	0.07
Ex. 13.1 *(s-dice($\mathcal{C}$))*	0.09
Ex. 13.2 *(s-dice($\mathcal{C}$))*	1.01
Ex. 14 *(s-roll-up($\mathcal{C}$))*	2.03
Ex. 15 *(s-drill-down($\mathcal{C}$)*	1.86
Ex. 16 *(s-roll-up (s-slice (s-dice($\mathcal{C}$))))*	3.04

The query runtime of the s-slice query in Ex 12 is measured as 0.07 seconds, which is an efficient SOLAP operator to execute, since filtering the given spatial literal is done only for the records of the specified level instances.

The query runtimes of the s-dice queries in Ex. 13.1 and Ex. 13.2 are measured as 0.09 and 1.01 seconds respectively, for the two alternative ways of implementing s-dice in SPARQL. It is clear that the first use of the s-dice operator is more efficient compared to the second, even though both return the same results. The first one performs s-dice by filtering the instances of the customers through their geometries that are within a buffer area of a circle with 5 km radius and the city center geometry as the center point of the circle. The second one performs s-dice by measuring the distances between each customer's location and their city center, and then applies a filter of those measurements to find the ones that are less than 5 km. This is a more expensive approach due to measuring the distances between each customer and city instances with a spatial distance function.

The query runtime of the s-roll-up query in Ex. 14 is measured as 2.03 seconds. The s-roll-up operator has a longer response time than the other operators as it combines a spatial distance function with aggregate operators. Due to its complexity, s-roll-up requires an inner select where it binds the distances of specified spatial level attributes (e.g., customer and supplier geometry), which is calculated with a spatial distance function. Those distance measurements are then

filtered in the outer select with respect to the aggregate function.

The query runtime of the s-drill-down query in Ex. 15 is measured as 1.86 seconds. S-drill-down operates in a very similar manner as s-roll-up. Theoretically, s-drill-down operates on an already spatially rolled up data cube, whilst it is implemented in practice as an s-roll-up on the base cube. The RDF data is at the lowest granularity, which is equal to the definition of the base cube. The SPARQL query of the s-drill-down operator is very similar the s-roll-up operator. The difference of the query runtime between Ex. 14 and Ex. 15 is directly related to the number of instances of the specified levels and the complexity of the spatial functions.

The query runtime for the nested SOLAP query in Ex. 16 is measured as 3.04 seconds. The nesting in the query is the main reason why it has a longer response time. Moreover, the nested query has more triple patterns and clauses that need to be evaluated during query execution. Hence, it takes longer than evaluating a single SOLAP operator.

A2. Table of Contents

Table 6 summarizes a list of all the definitions, extensions, remarks, and algorithms. *Definitions* are enumerated starting from "1". *Extensions* are enumerated starting from "5" to be consistent with the base definition of the extension, e.g., Def. 5 (Dimensions) is followed by Ext. 5 (Spatial dimensions). *Remarks* are enumerated starting from "17" for OLAP operators, which are given before the definitions of the corresponding SOLAP operator, e.g., Remark 17 (Slice) is followed by Def. 17 (S-Slice). *Algorithms* are enumerated starting from "1".

Table 6
Table of Contents

Type	**No.**	**Topic**	**Sect.**	**Page**
Definition	1	Spatial aggregation	3.2	4
Definition	2	Topological relations	3.2	4
Definition	3	Numeric operations	3.2	4
Definition	4	RDF triple	4	7
Definition	5	Dimensions	4.1	8-9
Extension	5	Spatial dimensions	4.1	9
Definition	6	Hierarchies	4.1	9
Extension	6	Spatial hierarchies	4.1	9
Definition	7	Levels	4.1	9
Extension	7	Spatial levels	4.1	9-10
Definition	8	Attributes	4.1	10-11
Extension	8	Spatial attributes	4.1	11
Definition	9	Hierarchy steps	4.1	11
Extension	9	Spatial hierarchy steps	4.1	12
Definition	10	Partial order on levels	4.1	12
Definition	11	Measures	4.1	12
Extension	11	Spatial measures	4.1	12
Definition	12	Fact	4.1	13
Extension	12	Spatial fact	4.1	13
Definition	13	Level members	4.2	14
Definition	14	Attributes of level members	4.2	14
Definition	15	Partial order on level members	4.2	14
Definition	16	Fact members	4.2	14-15
Remark	17	Slice	5	15
Definition	17	S-Slice	5	15-16
Remark	18	Dice	5	17
Definition	18	S-Dice	5	17
Remark	19	Roll-up	5	17-18
Definition	19	S-Roll-up	5	18
Remark	20	Drill-down	5	19
Definition	20	S-Drill-down	5	19
Algorithm	1	RUPath	6.1	20
Algorithm	2	S-SliceGenerator	6.1	21
Algorithm	3	S-DiceGenerator	6.1	23
Algorithm	4	SRUGenerator	6.1	24
Algorithm	5	WriteSPARQL	6.2	25