W3C Recommendation

The RDF Data Cube Vocabulary

W3C Recommendation 16 January 2014

This version:
http://www.w3.org/TR/2014/REC-vocab-data-cube-20140116/
Latest published version:
http://www.w3.org/TR/vocab-data-cube/
Implementation report:
http://www.w3.org/2011/gld/wiki/Data_Cube_Implementations
Previous version:
http://www.w3.org/TR/2013/PR-vocab-data-cube-20131217/
Editors:
Richard Cyganiak, DERI, NUI Galway
Dave Reynolds, Epimorphics Ltd
Contributors:
Jeni Tennison

Please refer to the **errata**, a list of issues with this document discovered after publication.

This document is also available in this non-normative format: diff to previous version

The English version of this specification is the only normative version. Non-normative translations may also be available.

Abstract

There are many situations where it would be useful to be able to publish multi-dimensional data, such as statistics, on the web in such a way that it can be linked to related data sets and concepts. The Data Cube vocabulary provides a means to do this using the W3C RDF (Resource Description Framework) standard. The model underpinning the Data Cube vocabulary is compatible with the cube model that underlies SDMX (Statistical Data and Metadata eXchange), an ISO standard for exchanging and sharing statistical data and metadata among organizations. The Data Cube vocabulary is a core foundation which supports extension vocabularies to enable publication of other aspects of statistical data flows or other multi-dimensional data sets.

The namespace for all terms in this ontology is: http://purl.org/linked-data/cube#

The vocabulary defined in this document is also available in these non-normative formats: Turtle.

Status of This Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

This document has been reviewed by W3C Members, by software developers, and by other W3C groups and interested parties, and is endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This vocabulary was originally developed and published outside of W3C, but has been extended and further developed within the Government Linked Data Working Group.

This document was published by the Government Linked Data Working Group as a Recommendation. If you wish to make comments regarding this document, please send them to public-gld-comments@w3.org (subscribe, archives). All comments are welcome.

Please see the Working Group's implementation report.

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

Table of Contents

1. Outline of the vocabulary

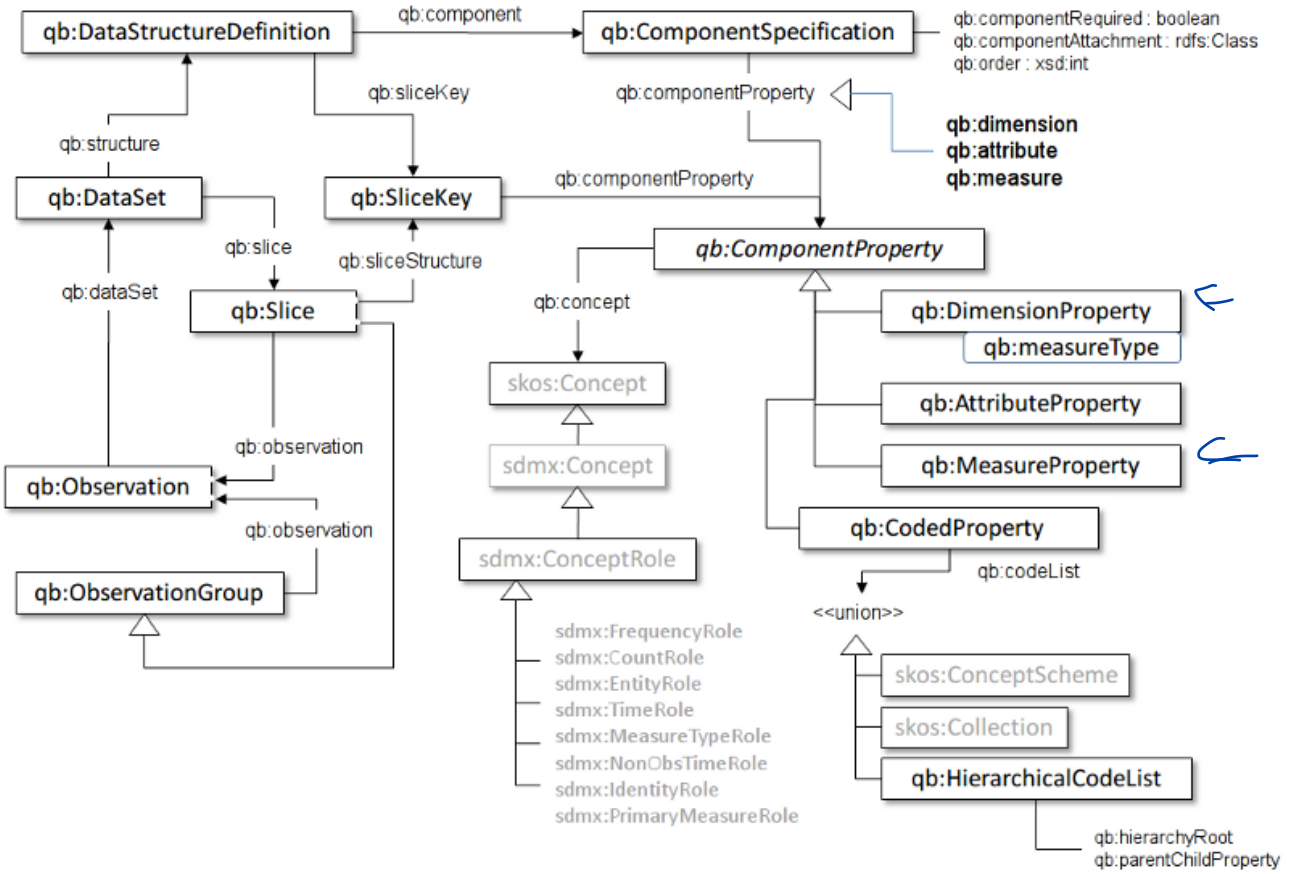

Fig. 1 Pictorial summary of key terms and their relationship

1.1 Vocabulary index

Classes: qb:Attachable qb:AttributeProperty qb:CodedProperty qb:ComponentProperty qb:ComponentSet qb:ComponentSpecification qb:DataSet qb:DataStructureDefinition qb:DimensionProperty qb:HierarchicalCodeList qb:MeasureProperty qb:Observation qb:Slice qb:ObservationGroup qb:SliceKey

Properties: qb:attribute qb:codeList qb:component qb:componentAttachment qb:componentProperty qb:componentRequired qb:concept qb:dataSet qb:dimension qb:hierarchyRoot qb:measure qb:measureDimension qb:measureType qb:observation qb:observationGroup qb:order qb:parentChildProperty qb:slice qb:sliceKey qb:sliceStructure qb:structure

2. Introduction

This section is non-normative.

Statistical data is a foundation for policy prediction, planning and adjustments and underpins many of the mash-ups and visualisations we see on the web. There is strong interest in being able to publish statistical data in a web-friendly format to enable it to be linked and combined with related information.

At the heart of a statistical dataset is a set of observed values organized along a group of dimensions, together with associated metadata. The Data Cube vocabulary enables such information to be represented using the W3C RDF (Resource Description Framework) standard and published following the principles of linked data. The vocabulary is based upon the approach used by the SDMX ISO standard for statistical data exchange. This *cube* model is very general and so the Data Cube vocabulary can be used for other data sets such as survey data, spreadsheets and OLAP data cubes [OLAP].

The Data Cube vocabulary is focused purely on the publication of multi-dimensional data on the web. We envisage a series of modular vocabularies being developed which extend this core foundation. In particular, we see the need for an SDMX extension vocabulary to support the publication of additional context to statistical data (such as the encompassing Data Flows and associated Provision Agreements). Other extensions are possible to support metadata for surveys (so called "micro-data", as encompassed by DDI) or publication of statistical reference metadata.

The Data Cube in turn builds upon the following existing RDF vocabularies:

- SKOS for concept schemes
- SCOVO for core statistical structures
- Dublin Core Terms for metadata
- VoiD for data access
- FOAF for agents
- ORG for organizations

2.1 RDF and Linked Data

Linked data is an approach to publishing data on the web, enabling datasets to be linked together through references to common concepts. The approach [LOD] recommends use of HTTP URIs to name the entities and concepts so that consumers of the data can look-up those URIs to get more information, including links to other related URIs. RDF [RDF-PRIMER] provides a standard for the representation of the information that describes those entities and concepts, and is returned by dereferencing the URIs.

There are a number of benefits to being able to publish multi-dimensional data, such as statistics, using RDF and the linked data approach:

- The individual observations, and groups of observations, become (web) addressable. This allows publishers and third parties to annotate and link to this data; for example a report can reference the specific figures it is based on allowing for fine grained provenance trace-back.
- Data can be flexibly combined across datasets sets (for example *find all Religious schools in census areas with high values for National Indicators pertaining to religious tolerance*). The statistical data becomes an integral part of the broader web of linked data.
- For publishers who currently only offer static files then publishing as linked-data offers a flexible, non-proprietary, machine readable means of publication that supports an out-of-the-box web API for programmatic access.
- It enables reuse of standardized tools and components.

2.2 SDMX and related standards

The Statistical Data and Metadata Exchange (SDMX) Initiative was organised in 2001 by seven international organizations (BIS, ECB, Eurostat, IMF, OECD, World Bank and the UN) to realise greater efficiencies in statistical practice. These organisations all collect significant amounts of data, mostly from the national level, to support policy. They also disseminate data at the supra-national and international levels.

There have been a number of important results from this work: two versions of a set of technical specifications - ISO:TS 17369 (SDMX) - and the release of several recommendations for structuring and harmonising cross-domain statistics, the SDMX Content-Oriented Guidelines. All of the products are available at www.sdmx.org. The standards are now being widely adopted around the world for the collection, exchange, processing, and dissemination of aggregate statistics by official statistical organisations. The UN Statistical Commission recommended SDMX as the preferred standard for statistics in 2007.

The SDMX specification defines a core *information model* which is reflected in concrete form in two syntaxes - SDMX-ML (an XML syntax) and SDMX-EDI.

The RDF Data Cube vocabulary builds upon the core of the the SDMX 2.0 Information Model [SDMX20].

Readers may find the SDMX User Guide [SDMX-GUIDE] useful background.

A key component of the SDMX standards package are the **Content-Oriented Guidelines** (COGs), a set of cross-domain concepts, code lists, and categories that support interoperability and comparability between datasets by providing a shared terminology between SDMX implementers [COG]. RDF versions of these terms are available separately for use along with the Data Cube vocabulary, see Content oriented guidelines for further details. These external resources do not form a normative part of the Data Cube Vocabulary specification.

2.3 Audience and scope

This document describes the Data Cube vocabulary It is aimed at people wishing to publish statistical or other multi-dimension data in RDF. Mechanics of cross-format translation from other formats such as SDMX-ML are not covered here.

3. Namespaces and Document Conventions

The names of RDF entities -- classes, predicates, individuals -- are URIs. These are usually expressed using a compact notation where the name is written `prefix:localname`, and where the `prefix` identifies a *namespace URI*. The namespace identified by the prefix is prepended to the `localname` to obtain the full URI.

The following namespaces are used in this document:

Prefix	Namespace	Reference
qb	http://purl.org/linked-data/cube#	*This document*
skos	http://www.w3.org/2004/02/skos/core#	[SKOS-REFERENCE]
scovo	http://purl.org/NET/scovo#	[SCOVO] [HAUS09]
void	http://rdfs.org/ns/void#	[void]
foaf	http://xmlns.com/foaf/0.1/	[FOAF]
org	http://www.w3.org/ns/org#	[ORG]
dct	http://purl.org/dc/terms/	[DC11]
owl	http://www.w3.org/2002/07/owl#	[OWL2-PRIMER]
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#	[RDF-CONCEPTS]
rdfs	http://www.w3.org/2000/01/rdf-schema#	[RDF-SCHEMA]
admingeo	http://data.ordnancesurvey.co.uk/ontology/admingeo/	*(Non-normative, used for examples only)*
interval	http://reference.data.gov.uk/def/intervals/	*(Non-normative, used for examples only)*

Prefix	Namespace	Reference
eg	http://example.org/ns#	*(Non-normative, used for examples only)*

All RDF examples are written in Turtle syntax [turtle].

4. Conformance

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

The key words MUST, MUST NOT, REQUIRED, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this specification are to be interpreted as described in [RFC2119].

A data interchange, however that interchange occurs, is conformant with Data Cube if:

- it uses terms (classes and properties) from Data Cube in a way consistent with their semantics as declared in this specification, in particular the exchanged RDF graphs constitute either *well-formed* or *well-formed abbreviated* Data Cubes;
- it does **not** use terms from other vocabularies **instead** of ones defined in this vocabulary that could reasonably be used (use of such terms **in addition** to Data Cube terms is permissible).

A conforming data interchange:

- MAY include terms from other vocabularies;
- MAY use only a subset of Data Cube terms.

5. Data cubes

This section is non-normative.

5.1 Data Sets

This section is non-normative.

A DataSet is a collection of statistical data that corresponds to a defined structure. The data in a data set can be roughly described as belonging to one of the following kinds:

Observations
: This is the actual data, the measured values. In a statistical table, the observations would be the values in the table cells.

Organizational structure
: To locate an observation within the hypercube, one has at least to know the value of each dimension at which the observation is located, so these values must be specified for each observation. Datasets can have additional organizational structure in the form of *slices* as described in section 7.2.

Structural metadata
: Having located an observation, we need certain metadata in order to be able to interpret it. What is the unit of measurement? Is it a normal value or a series break? Is the value measured or estimated? These metadata are provided as *attributes* and can be attached to individual observations, or to higher levels.

Reference metadata
: This is metadata that describes the dataset as a whole, such as categorization of the dataset, its publisher, and a SPARQL endpoint where it can be accessed. External metadata is described in section 9.

5.2 The cube model - dimensions, attributes, measures

This section is non-normative.

A statistical data set comprises a collection of observations made at some points across some logical space. The collection can be characterized by a set of dimensions that define what the observation applies to (e.g. time, area, gender) along with metadata describing what has been measured (e.g. economic activity, population), how it was measured and how the observations are expressed (e.g. units, multipliers, status). We can think of the statistical data set as a multi-dimensional space, or hyper-cube, indexed by those dimensions. This space is commonly referred to as a *cube* for short; though the name shouldn't be taken literally, it is not meant to imply that there are exactly three dimensions (there can be more or fewer) nor that all the dimensions are somehow similar in size.

A cube is organized according to a set of *dimensions*, *attributes* and *measures*. We collectively call these *components*.

The *dimension* components serve to identify the observations. A set of values for all the dimension components is sufficient to identify a single observation. Examples of dimensions include the time to which the observation applies, or a geographic region which the observation covers.

The *measure* components represent the phenomenon being observed.

The *attribute* components allow us to qualify and interpret the observed value(s). They enable specification of the units of measure, any scaling factors and metadata such as the status of the observation (e.g. *estimated*, *provisional*).

5.3 Introducing Slices

This section is non-normative.

It is frequently useful to group subsets of observations within a dataset. In particular to fix all but one (or a small subset) of the dimensions and be able to refer to all observations with those dimension values as a single entity. We call such a selection a *slice* through the cube. For example, given a data set on regional performance indicators then we might group together all the observations about a given indicator and a given region. Each such group would be a slice representing a time series of observed values.

A data publisher may identify slices through the data for various purposes. They can be a useful grouping to which metadata might be attached, for example to note a change in measurement process which affects a particular time or region. Slices also enable the publisher to identify and label particular subsets of the data which should be presented to the user - they can enable the consuming application to more easily construct the appropriate graph or chart for presentation.

In statistical applications it is common to work with slices in which a single dimension is left unspecified. In particular, to refer to such slices in which the single free dimension is time as *Time Series* and to refer slices along non-time dimensions as *Sections*. Within the Data Cube vocabulary we allow arbitrary dimensionality slices and do not give different names to particular types of slice. Such sub-classes of slice could be added in extension vocabularies.

5.4 An example

This section is non-normative.

In order to illustrate the use of the data cube vocabulary we will use a small demonstration data set extracted from StatsWales report number 003311 which describes life expectancy broken down by region (unitary authority), age and time. The extract we will use is:

	2004-2006		2005-2007		2006-2008	
	Male	**Female**	**Male**	**Female**	**Male**	**Female**
Newport	76.7	80.7	77.1	80.9	77.0	81.5
Cardiff	78.7	83.3	78.6	83.7	78.7	83.4
Monmouthshire	76.6	81.3	76.5	81.5	76.6	81.7
Merthyr Tydfil	75.5	79.1	75.5	79.4	74.9	79.6

We can see that there are three dimensions - time period (rolling averages over three year timespans), region and sex. Each observation represents the life expectancy for that population (the measure) and we will need an attribute to define the units (years) of the measured values.

An example of slicing the data would be to define slices in which the time and sex are fixed for each slice. Such slices then show the variation in life expectancy across the different regions, i.e. corresponding to the columns in the above tabular layout.

A complete encoding of this data as a Data Cube, including such a slice structure, is shown in Appendix C.

6. Creating data structure definitions

A `qb:DataStructureDefinition` defines the structure of one or more datasets. In particular, it defines the dimensions, attributes and measures used in the dataset along with qualifying information such as ordering of dimensions and whether attributes are required or optional. For well-formed data sets much of this information is implicit within the RDF component properties found on the observations. However, the explicit declaration of the structure has several benefits:

- it enables verification that the data set matches the expected structure, in particular helps with detection of incoherent sets obtained by combining differently structured source data;
- it allows a consumer to easily determine what dimensions are available for query and their presentational order, which in turn simplifies data consumption, for example for UI construction;
- it supports transmission of the structure information in associated SDMX data flows (see below).

It is common, when publishing statistical data, to have a regular series of publications which all follow the same structure. The notion of a Data Structure Definition (DSD) allows us to define that structure once and then reuse it for each publication in the series. Consumers can then be confident that the structure of the data has not changed.

6.1 Dimensions, attributes and measures

The Data Cube vocabulary represents the dimensions, attributes and measures as RDF properties. Each is an instance of the abstract `qb:ComponentProperty` class, which in turn has sub-classes `qb:DimensionProperty`, `qb:AttributeProperty` and `qb:MeasureProperty`.

A component property encapsulates several pieces of information:

- the concept being represented (e.g. time or geographic area),
- the nature of the component (dimension, attribute or measure) as represented by the type of the component property,
- the type or code list used to represent the value.

The same *concept* can be manifested in different components. For example, the concept of *currency* may be used as a dimension (in a data set dealing with exchange rates) or as an attribute (when describing the currency in which an observed trade took place). The concept of time is typically used only as a dimension but may be encoded as a data value (e.g. an `xsd:dateTime`) or as a symbolic value (e.g. a URI drawn from the reference time URI set developed by data.gov.uk). In statistical agencies it is common to have a standard thesaurus of statistical concepts which underpin the components used in multiple different data sets.

To support this reuse of general statistical concepts the data cube vocabulary provides the `qb:concept` property which links a `qb:ComponentProperty` to the concept it represents. We use the SKOS vocabulary [SKOS-PRIMER] to represent such concepts. This is very natural for those cases where the concepts are already maintained as a controlled term list or thesaurus. When developing a data structure definition for an informal data set there may not be an appropriate concept already. In those cases, if the concept is likely to be reused in other guises it is recommended to publish a `skos:Concept` along with the specific `qb:ComponentProperty`. However, if such reuse is not expected then it is not required to do so - the `qb:concept` link is optional and a simple instance of the appropriate subclass of `qb:ComponentProperty` is sufficient.

The representation of the possible values of the component is described using the `rdfs:range` property of the component in the usual RDF manner. Thus, for example, values of a time dimension might be represented using literals of type `xsd:dateTime` or as URIs drawn from a time reference service.

In statistical data sets it is common for values to be encoded using some (possibly hierarchical) code list and it can be useful to be able to easily identify the overall code list in some more structured form. To cater for this a component can also be optionally annotated with a `qb:codeList` to indicate a set of `skos:Concept`s which may be used as codes. The `qb:codeList` value may be a `skos:ConceptScheme`, `skos:Collection` or `qb:HierarchicalCodeList`. In such a case the `rdfs:range` of the component might be left as simply `skos:Concept` but a useful design pattern is to also define an `rdfs:Class` whose members are all the `skos:Concept`s within a particular scheme. In that way the `rdfs:range` can be made more specific which enables generic RDF tools to perform appropriate range checking.

Note that in any SDMX extension vocabulary there would be one further item of information to encode about components - the role that they play within the structure definition. In particular, it is sometimes convenient for consumers to be able to easily identify which is the time dimension, which component is the primary measure and so forth. It turns out that such roles are intrinsic to the concepts and so this information can be encoded by providing subclasses of `skos:Concept` for each role. The particular choice of roles here is specific to the SDMX standard and so is not included within the core Data Cube vocabulary.

Before illustrating the components needed for our running example, there is one more piece of machinery to introduce, a reusable set of concepts and components based on SDMX.

6.2 Content oriented guidelines

This section is non-normative.

The SDMX standard includes a set of *content oriented guidelines* (COG) [COG] which define a set of common statistical concepts and associated code lists that are intended to be reusable across data sets. A community group has developed RDF encodings of these guidelines. These comprise:

Prefix	Namespace	Description
`sdmx-concept`	http://purl.org/linked-data/sdmx/2009/concept#	SKOS Concepts for each COG defined concept
`sdmx-code`	http://purl.org/linked-data/sdmx/2009/code#	SKOS Concepts and ConceptSchemes for each COG defined code list
`sdmx-dimension`	http://purl.org/linked-data/sdmx/2009/dimension#	component properties corresponding to each COG concept that can be used as a dimension
`sdmx-attribute`	http://purl.org/linked-data/sdmx/2009/attribute#	component properties corresponding to each COG concept that can be used as an attribute
`sdmx-measure`	http://purl.org/linked-data/sdmx/2009/measure#	component properties corresponding to each COG concept that can be used as a measure

These community resources are provided as a convenience and do not form part of the Data Cube specification. However, they are used by a number of existing Data Cube publications and so we will reference them within our worked examples.

6.3 Example dimensions and measure

This section is non-normative.

Turning to our example data set then we can see there are three dimensions to represent - time period, region (unitary authority) and sex. There is a single (primary) measure which corresponds to the topic of the data set (life expectancy) and encodes a value in years. Hence, we need the following components.

Time. There is a suitable predefined concept in the SMDX-COG for this, REF_PERIOD, so we could reuse the corresponding component property `sdmx-dimension:refPeriod`. However, to represent the time period itself it would be convenient to use the data.gov.uk reference time service and to declare this within the data structure definition.

> EXAMPLE 1

```
eg:refPeriod  a rdf:Property, qb:DimensionProperty;
    rdfs:label "reference period"@en;
    rdfs:subPropertyOf sdmx-dimension:refPeriod;
    rdfs:range interval:Interval;
    qb:concept sdmx-concept:refPeriod .
```

Region. Again there is a suitable COG concept and associated component that we can use for this, and again we can customize the range of the component. In this case we can use the Ordnance Survey Administrative Geography Ontology [OS-GEO].

EXAMPLE 2

```
eg:refArea  a rdf:Property, qb:DimensionProperty;
    rdfs:label "reference area"@en;
    rdfs:subPropertyOf sdmx-dimension:refArea;
    rdfs:range admingeo:UnitaryAuthority;
    qb:concept sdmx-concept:refArea .
```

Sex. In this case we can use the corresponding COG component `sdmx-dimension:sex` directly, since the default code list for it includes the terms we need.

Measure. This property will give the value of each observation. We could use the default `smdx-measure:obsValue` for this (defining the topic being observed using metadata). However, it can aid readability and processing of the RDF data sets to use a specific measure corresponding to the phenomenon being observed.

EXAMPLE 3

```
eg:lifeExpectancy  a rdf:Property, qb:MeasureProperty;
    rdfs:label "life expectancy"@en;
    rdfs:subPropertyOf sdmx-measure:obsValue;
    rdfs:range xsd:decimal .
```

Unit measure attribute. The primary measure on its own is a plain decimal value. To correctly interpret this value we need to define what units it is measured in (years in this case). This is defined using attributes which qualify the interpretation of the observed value. Specifically in this example we can use the predefined `sdmx-attribute:unitMeasure` which in turn corresponds to the COG concept of `UNIT_MEASURE`. To express the value of this attribute we would typically use a common thesaurus of units of measure. For the sake of this simple example we will use the DBpedia resource `http://dbpedia.org/resource/Year` which corresponds to the topic of the Wikipedia page on "Years".

This covers the minimal components needed to define the structure of this data set.

6.4 ComponentSpecifications and DataStructureDefinitions

To combine the components into a specification for the structure of this dataset we need to declare a `qb:DataStructureDefinition` resource which in turn will reference a set of `qb:ComponentSpecification` resources. The `qb:DataStructureDefinition` will be reusable across other data sets with the same structure.

In the simplest case the `qb:ComponentSpecification` simply references the corresponding `qb:ComponentProperty` (usually using one of the sub properties `qb:dimension`, `qb:measure` or `qb:attribute`). However, it is also possible to qualify the component specification in several ways.

- Attributes may be declared as optional or required. If an attribute is required to be present for every observation then the specification should set `qb:componentRequired`. In the absence of such a declaration an attribute is assumed to be optional. The `qb:componentRequired` declaration may only be applied to component specifications of attributes - measures and dimensions are always required.
- The components may be ordered by giving an integer value for `qb:order`. This order carries no semantics but can be useful to aid consuming agents in generating appropriate user interfaces. It can also be useful in the publication chain to enable synthesis of appropriate URIs for observations.
- By default the values of all of the components will be attached to each individual observation, this is called the *normalized* representation. This allows such observations to stand alone, so that a SPARQL query to retrieve the observation can immediately locate the attributes which enable the observation to be interpreted. However, it is also permissible to attach attributes to the overall data set, to an intervening slice or to a specific Measure (in the case of multiple measures). This reduces some of the redundancy in the encoding of the instance data. To declare such an abbreviated structure, the `qb:componentAttachment` property of the specification should reference the class corresponding to the attachment level (e.g. `qb:DataSet` for attributes that will be attached to the overall data set). The classes which can be used as such attachment levels are all subclasses of `qb:Attachable`.

In the case of our running example the dimensions can be usefully ordered. There is only one attribute, the unit measure, and this is required. In the interest of illustrating the vocabulary use we will declare that this attribute will be attached at the level of the data set, however normalized representations are in general easier to query and combine.

So the structure of our example data set (and other similar datasets) can be declared by:

EXAMPLE 4

```
eg:dsd-le a qb:DataStructureDefinition;
    # The dimensions
    qb:component [ qb:dimension eg:refArea;         qb:order 1 ];
    qb:component [ qb:dimension eg:refPeriod;       qb:order 2 ];
    qb:component [ qb:dimension sdmx-dimension:sex; qb:order 3 ];
    # The measure(s)
    qb:component [ qb:measure eg:lifeExpectancy];
    # The attributes
    qb:component [ qb:attribute sdmx-attribute:unitMeasure;
                   qb:componentRequired "true"^^xsd:boolean;
                   qb:componentAttachment qb:DataSet; ] .
```

Note that we have given the data structure definition (DSD) a URI since it will be reused across different datasets with the same structure. Similarly the component properties themselves can be reused across different DSDs. However, the component specifications are only useful within the scope of a particular DSD and so we have chosen to represent them using blank nodes.

6.5 Handling multiple measures

Our example data set is relatively simple in having a single observable (in this case "life expectancy") that is being measured. In other data sets there can be multiple measures. These measures may be of similar nature (e.g. a data set on local government performance might provide multiple different performance indicators for each region) or quite different (e.g. a data set on trades might provide quantity, value, weight for each trade).

There are two approaches to representing multiple measures supported by the Data Cube vocabulary.

In the first approach each observation records a single observed value for one measure. We introduce an additional dimension whose value indicates the measure being conveyed by each observation. This *measure dimension* approach is the one supported by the SDMX information model.

In the second approach a single observation can provide values for multiple different measures. This is particularly appropriate in cases where each of those values relates to a single observational event such as a multi-spectral sensor measurement. This *multi-measure* approach is commonly used in applications such as Business Intelligence and OLAP.

The Data Cube vocabulary permits either representation approach to be used though they cannot be mixed within the same data set.

Both representation approaches require that, for every point in the space of dimensions for which there is an observation, a value must be given for every measure. In the case of multi-measure observations each measure must be present on each observation. In cubes which use a measure dimension there are sets of observations for each populated point in the cube and within each of those sets there must be an observation giving each measure.

Multi-measure observations

This approach allows multiple observed values to be attached to an individual observation. It is suited to representation of things like sensor data and OLAP cubes. To use this representation you simply declare multiple `qb:MeasureProperty` components in the data structure definition and attach an instance of each property to the observations within the data set.

For example, if we have a set of shipment data containing unit count and total weight for each shipment then we might have a data structure definition such as:

EXAMPLE 5

```
eg:dsd1 a qb:DataStructureDefinition;
    rdfs:comment "shipments by time (multiple measures approach)"@en;
    qb:component
        [ qb:dimension  sdmx-dimension:refTime; ],
        [ qb:measure    eg-measure:quantity; ],
        [ qb:measure    eg-measure:weight; ] .
```

This would correspond to individual observations such as:

EXAMPLE 6

```
eg:dataset1 a qb:DataSet;
    qb:structure eg:dsd1 .

eg:obs1a  a qb:Observation;
    qb:dataSet eg:dataset1;
    sdmx-dimension:refTime "2010-07-30"^^xsd:date;
    eg-measure:weight 1.3 ;
    eg-measure:quantity 42 ;
    .
```

Note that one limitation of the multi-measure approach is that it is not possible to attach an attribute to a single observed value. An attribute attached to the observation instance will apply to the whole observation (e.g. to indicate who made the observation). Attributes can also be attached directly to the `qb:MeasureProperty` itself (e.g. to indicate the *unit of measure* for

that measure) but that attachment applies to the whole data set (indeed any data set using that measure property) and cannot vary for different observations. For applications where this limitation is a problem then use the *measure dimension* approach.

Measure dimension

This approach restricts observations to having a single measured value but allows a data set to carry multiple measures by adding an extra dimension, a *measure dimension*. The value of the measure dimension denotes which particular measure is being conveyed by the observation. This is the representation approach used within SDMX and an extension vocabulary could introduce a sub-class of `qb:DataStructureDefinition` which enforces such a single-measure restriction.

To use this representation you declare an additional dimension within the data structure definition to play the role of the measure dimension. For use within the Data Cube vocabulary we provide a single distinguished component for this purpose -- `qb:measureType`. An extension vocabulary could generalize this through the provision of roles to identify concepts which act as measure types, enabling other measure dimensions to be declared.

In the special case of using `qb:measureType` as the measure dimension, the set of allowed measures is assumed to be those measures declared within the DSD. There is no need to define a separate code list or enumerated class to duplicate this information. Thus, `qb:measureType` is a “magic” dimension property with an implicit code list. This notion of an implicit code list for `qb:measureType` is a small divergence from SDMX usage.

The data structure definition for our above example, using this representation approach, would then be:

EXAMPLE 7

```
eg:dsd2 a qb:DataStructureDefinition;
    rdfs:comment "shipments by time (measure dimension approach)"@en;
    qb:component
        [ qb:dimension  sdmx-dimension:refTime; ],
        [ qb:measure    eg-measure:quantity; ],
        [ qb:measure    eg-measure:weight; ],
        [ qb:dimension  qb:measureType; ] .
```

This would correspond to individual observations such as:

EXAMPLE 8

```
eg:dataset2 a qb:DataSet;
    qb:structure eg:dsd2 .

eg:obs2a  a qb:Observation;
    qb:dataSet eg:dataset2;
    sdmx-dimension:refTime "2010-07-30"^^xsd:date;
    qb:measureType eg-measure:weight ;
    eg-measure:weight 1.3 .

eg:obs2b  a qb:Observation;
    qb:dataSet eg:dataset2;
    sdmx-dimension:refTime "30-07-2010"^^xsd:date;
    qb:measureType eg-measure:quantity ;
    eg-measure:quantity 42 .
```

Note the duplication of having the measure property show up both as the property that carries the measured value, and as the value of the measure dimension. We accept this duplication as necessary to ensure the uniform cube/dimension mechanism and a uniform way of declaring and using measure properties on all kinds of datasets.

Those familiar with SDMX should also note that in the RDF representation there is no need for a separate "primary measure" which subsumes each of the individual measures, those individual measures are used directly. Extension vocabularies could address the round-tripping of the SDMX primary measure by use of a separate annotation on the data structure definition.

7. Expressing data sets

7.1 Data sets and observations

A resource representing the entire data set is created and typed as `qb:DataSet` and linked to the corresponding data structure definition via the `qb:structure` property.

Pitfall: Note the capitalization of `qb:DataSet`, which differs from the capitalization in other vocabularies, such as void:Dataset and dcat:Dataset. This unusual capitalization is chosen for compatibility with the SDMX standard. The same applies to the related property `qb:dataSet`.

Each observation is represented as an instance of type `qb:Observation`. In the basic case then values for each of the attributes, dimensions and measurements are attached directly to the observation (remember that these components are all RDF properties). The observation is linked to the containing data set using the `qb:dataSet` property.

Thus for our running example we might expect to have:

EXAMPLE 9

```
eg:dataset-le1 a qb:DataSet;
    rdfs:label "Life expectancy"@en;
    rdfs:comment "Life expectancy within Welsh Unitary authorities - extracted from Stats Wales"@en;
    qb:structure eg:dsd-le ;
    .

eg:o1 a qb:Observation;
    qb:dataSet  eg:dataset-le1 ;
    eg:refArea                  ex-geo:newport_00pr ;
    eg:refPeriod                <http://reference.data.gov.uk/id/gregorian-interval/2004-01-01T00:00:00/P3Y> ;
    sdmx-dimension:sex          sdmx-code:sex-M ;
    sdmx-attribute:unitMeasure <http://dbpedia.org/resource/Year> ;
    eg:lifeExpectancy           76.7 ;
    .

eg:o2 a qb:Observation;
    qb:dataSet  eg:dataset-le1 ;
    eg:refArea                  ex-geo:cardiff_00pt ;
    eg:refPeriod                <http://reference.data.gov.uk/id/gregorian-interval/2004-01-01T00:00:00/P3Y> ;
    sdmx-dimension:sex          sdmx-code:sex-M ;
    sdmx-attribute:unitMeasure <http://dbpedia.org/resource/Year> ;
    eg:lifeExpectancy           78.7 ;
    .

eg:o3 a qb:Observation;
    qb:dataSet  eg:dataset-le1 ;
    eg:refArea                  ex-geo:monmouthshire_00pp ;
    eg:refPeriod                <http://reference.data.gov.uk/id/gregorian-interval/2004-01-01T00:00:00/P3Y> ;
    sdmx-dimension:sex          sdmx-code:sex-M ;
    sdmx-attribute:unitMeasure <http://dbpedia.org/resource/Year> ;
    eg:lifeExpectancy           76.6 ;
    .

...
```

This normalized structure makes it easy to query and combine data sets but there is some redundancy here. For example, the unit of measure for the life expectancy is uniform across the whole data set and does not change between observations. To cater for situations like this the Data Cube vocabulary allows components to be attached at a high level in the nested structure. Indeed if we re-examine our original Data Structure Declaration we see that we declared the unit of measure to be attached at the data set level. So an shortened version of the example is:

EXAMPLE 10

```
eg:dataset-le1 a qb:DataSet;
    rdfs:label "Life expectancy"@en;
    rdfs:comment "Life expectancy within Welsh Unitary authorities - extracted from Stats Wales"@en;
    qb:structure eg:dsd-le ;
    sdmx-attribute:unitMeasure <http://dbpedia.org/resource/Year> ;
    .

eg:o1 a qb:Observation;
    qb:dataSet  eg:dataset-le1 ;
    eg:refArea                  ex-geo:newport_00pr ;
    eg:refPeriod                <http://reference.data.gov.uk/id/gregorian-interval/2004-01-01T00:00:00/P3Y> ;
    sdmx-dimension:sex          sdmx-code:sex-M ;
    eg:lifeExpectancy           76.7 ;
    .

eg:o2 a qb:Observation;
    qb:dataSet  eg:dataset-le1 ;
    eg:refArea                  ex-geo:cardiff_00pt ;
    eg:refPeriod                <http://reference.data.gov.uk/id/gregorian-interval/2004-01-01T00:00:00/P3Y> ;
    sdmx-dimension:sex          sdmx-code:sex-M ;
    eg:lifeExpectancy           78.7 ;
    .

eg:o3 a qb:Observation;
    qb:dataSet  eg:dataset-le1 ;
    eg:refArea                  ex-geo:monmouthshire_00pp ;
    eg:refPeriod                <http://reference.data.gov.uk/id/gregorian-interval/2004-01-01T00:00:00/P3Y> ;
    sdmx-dimension:sex          sdmx-code:sex-M ;
    eg:lifeExpectancy           76.6 ;
    .

...
```

In a data set containing just observations with no intervening structure then each observation must have a complete set of dimension values, along with all the measure values. If the set is structured by using slices then further abbreviation is possible, as discussed in the next section.

7.2 Slices and groups of observations

Slices allow us to group subsets of observations together. This is not intended to represent arbitrary selections from the observations but uniform slices through the cube in which one or more of the dimension values are fixed.

Slices may be used for a number of reasons:

- to guide consuming applications in how to present the data (e.g. to organize data as a set of time series);
- to provide an identity (URI) for the slice to enable it to be annotated or externally referenced;
- to reduce the verbosity of the data set by only stating each fixed dimensional value once.

To illustrate the use of slices let us group the sample data set into geographic series. That will enable us to refer to e.g. "male life expectancy observations for 2004-2006" and guide applications to present a comparative chart across regions.

We first define the structure of the slices we want by associating a "slice key" with the data structure definition. This is done by creating a `qb:SliceKey` to list the component properties (which must be dimensions) which will be fixed in the slice. The key is attached to the DSD using `qb:sliceKey`. For example:

EXAMPLE 11

```
eg:sliceByRegion a qb:SliceKey;
    rdfs:label "slice by region"@en;
    rdfs:comment "Slice by grouping regions together, fixing sex and time values"@en;
    qb:componentProperty eg:refPeriod, sdmx-dimension:sex .

eg:dsd-le-slice1 a qb:DataStructureDefinition;
    qb:component
        [ qb:dimension eg:refArea;         qb:order 1 ],
        [ qb:dimension eg:refPeriod;       qb:order 2 ],
        [ qb:dimension sdmx-dimension:sex; qb:order 3 ],
        [ qb:measure eg:lifeExpectancy];
        [qb:attribute sdmx-attribute:unitMeasure; qb:componentAttachment qb:DataSet; ] ;
    qb:sliceKey eg:sliceByRegion .
```

In the instance data then slices are represented by instances of `qb:Slice` which link to the observations in the slice via `qb:observation` and to the key by means of `qb:sliceStructure`. Data sets indicate the slices they contain by means of `qb:slice`. Thus in our example we would have:

EXAMPLE 12

```
eg:dataset-le2 a qb:DataSet;
    rdfs:label "Life expectancy"@en;
    rdfs:comment "Life expectancy within Welsh Unitary authorities - extracted from Stats Wales"@en;
    qb:structure eg:dsd-le-slice2 ;
    sdmx-attribute:unitMeasure <http://dbpedia.org/resource/Year> ;
    qb:slice eg:slice2;
    .

eg:slice2 a qb:Slice;
    qb:sliceStructure  eg:sliceByRegion ;
    eg:refPeriod               <http://reference.data.gov.uk/id/gregorian-interval/2004-01-01T00:00:00/P3Y> ;
    sdmx-dimension:sex         sdmx-code:sex-M ;
    qb:observation eg:o1b, eg:o2b, eg:o3b, ... .

eg:o1b a qb:Observation;
    qb:dataSet  eg:dataset-le2 ;
    eg:refArea                 ex-geo:newport_00pr ;
    eg:refPeriod               <http://reference.data.gov.uk/id/gregorian-interval/2004-01-01T00:00:00/P3Y> ;
    sdmx-dimension:sex         sdmx-code:sex-M ;
    eg:lifeExpectancy          76.7 ;
    .

eg:o2b a qb:Observation;
    qb:dataSet  eg:dataset-le2 ;
    eg:refArea                 ex-geo:cardiff_00pt ;
    eg:refPeriod               <http://reference.data.gov.uk/id/gregorian-interval/2004-01-01T00:00:00/P3Y> ;
    sdmx-dimension:sex         sdmx-code:sex-M ;
    eg:lifeExpectancy          78.7 ;
    .

eg:o3b a qb:Observation;
    qb:dataSet  eg:dataset-le2 ;
    eg:refArea                 ex-geo:monmouthshire_00pp ;
    eg:refPeriod               <http://reference.data.gov.uk/id/gregorian-interval/2004-01-01T00:00:00/P3Y> ;
    sdmx-dimension:sex         sdmx-code:sex-M ;
    eg:lifeExpectancy          76.6 ;
    .

...
```

Note that here we are still repeating the dimension values on the individual observations. This normalized representation means that a consuming application can still query for observed values uniformly without having to first parse the data structure definition and search for slice definitions. If it is desired, this redundancy can be reduced by declaring different attachment levels for the dimensions. For example:

EXAMPLE 13

```
eg:dsd-le-slice3 a qb:DataStructureDefinition;
    qb:component
        [ qb:dimension eg:refArea;         qb:order 1 ];
        [ qb:dimension eg:refPeriod;       qb:order 2; qb:componentAttachment qb:Slice ];
        [ qb:dimension sdmx-dimension:sex; qb:order 3; qb:componentAttachment qb:Slice ];
        [ qb:measure eg:lifeExpectancy];
        [ qb:attribute sdmx-attribute:unitMeasure; qb:componentAttachment qb:DataSet; ] ;
    qb:sliceKey eg:sliceByRegion .

eg:dataset-le3 a qb:DataSet;
    rdfs:label "Life expectancy"@en;
    rdfs:comment "Life expectancy within Welsh Unitary authorities - extracted from Stats Wales"@en;
    qb:structure eg:dsd-le-slice3 ;
    sdmx-attribute:unitMeasure <http://dbpedia.org/resource/Year> ;
    qb:slice eg:slice3 ;
    .

eg:slice3 a qb:Slice;
    qb:sliceStructure  eg:sliceByRegion ;
    eg:refPeriod               <http://reference.data.gov.uk/id/gregorian-interval/2004-01-01T00:00:00/P3Y> ;
    sdmx-dimension:sex         sdmx-code:sex-M ;
    qb:observation eg:o1c, eg:o2c, eg:o3c, ... .

eg:o1c a qb:Observation;
    qb:dataSet  eg:dataset-le3 ;
    eg:refArea                 ex-geo:newport_00pr ;
    eg:lifeExpectancy          76.7 ;
    .

eg:o2c a qb:Observation;
    qb:dataSet  eg:dataset-le3 ;
    eg:refArea                 ex-geo:cardiff_00pt ;
    eg:lifeExpectancy          78.7 ;
    .

eg:o3c a qb:Observation;
    qb:dataSet  eg:dataset-le3 ;
    eg:refArea                 ex-geo:monmouthshire_00pp ;
    eg:lifeExpectancy          76.6 ;
    .

...
```

There are also situations in which a publisher wishes to group a set of observations together for ease of access or presentation purposes but where that set is not defined by simply fixing a set of dimension values. For example, in representing weather observations it can be desirable to group together the latest observation available from each station even though each observation may have been taken at a different time. For those situations the Data Cube vocabulary supports `qb:ObservationGroup`. A `qb:ObservationGroup` can contain an arbitrary collection of observations. A `qb:Slice` is a special case of a `qb:ObservationGroup`.

8. Concept schemes and code lists

8.1 Coded values for components properties

The values for dimensions within a data set must be unambiguously defined. They may be typed values (e.g. `xsd:dateTime` for time instances) or codes drawn from some code list. Similarly, many attributes used in data sets represent coded values from some controlled term list rather than free text descriptions. In the Data Cube vocabulary such codes are represented by URI references in the usual RDF fashion.

Sometimes appropriate URI sets already exist for the relevant dimensions (e.g. the representations of area and time periods in our running example). In other cases the data set being converted may use controlled terms from some scheme which does not yet have associated URIs. In those cases we recommend use of SKOS, representing the individual code values using `skos:Concept` and the overall set of admissible values using `skos:ConceptScheme` or `skos:Collection`.

We illustrate this with an example drawn from the translation of the SDMX COG code list for gender, as used already in our worked example. The relevant subset of this code list is:

EXAMPLE 14

```
sdmx-code:sex a skos:ConceptScheme;
    skos:prefLabel "Code list for Sex (SEX) - codelist scheme"@en;
    rdfs:label "Code list for Sex (SEX) - codelist scheme"@en;
    skos:notation "CL_SEX";
    skos:note "This  code list provides the gender."@en;
    skos:definition <http://sdmx.org/wp-content/uploads/2009/01/02_sdmx_cog_annex_2_cl_2009.pdf> ;
    rdfs:seeAlso sdmx-code:Sex ;
    sdmx-code:sex skos:hasTopConcept sdmx-code:sex-F ;
    sdmx-code:sex skos:hasTopConcept sdmx-code:sex-M .

sdmx-code:Sex a rdfs:Class, owl:Class;
    rdfs:subClassOf skos:Concept ;
```

```
        rdfs:label "Code list for Sex (SEX) - codelist class"@en;
        rdfs:comment "This  code list provides the gender."@en;
        rdfs:seeAlso sdmx-code:sex .

    sdmx-code:sex-F a skos:Concept, sdmx-code:Sex;
        skos:topConceptOf sdmx-code:sex;
        skos:prefLabel "Female"@en ;
        skos:notation "F" ;
        skos:inScheme sdmx-code:sex .

    sdmx-code:sex-M a skos:Concept, sdmx-code:Sex;
        skos:topConceptOf sdmx-code:sex;
        skos:prefLabel "Male"@en ;
        skos:notation "M" ;
        skos:inScheme sdmx-code:sex .
```

`skos:prefLabel` is used to give a name to the code, `skos:note` gives a description and `skos:notation` can be used to record a short form code which might appear in other serializations. The SKOS specification [SKOS-REFERENCE] recommends the generation of a custom datatype for each use of `skos:notation` but here the notation is not intended for use within RDF encodings, it merely documents the notation used in other representations (which do not use such a datatype).

It is convenient and good practice when developing a code list to also create a Class to denote all the codes within the code list, irrespective of hierarchical structure. This allows the range of an `qb:ComponentProperty` to be defined by using `rdfs:range` which then permits standard RDF closed-world checkers to validate use of the code list without requiring custom SDMX-RDF-aware tooling. We do that in the above example by using the common convention that the class name is the same as that of the concept scheme but with leading upper case.

This code list can then be associated with a coded property, such as a dimension:

EXAMPLE 15

```
    eg:sex a qb:DimensionProperty, qb:CodedProperty;
        qb:codeList sdmx-code:sex ;
        rdfs:range sdmx-code:Sex .
```

Explicitly declaring the code list using `qb:codeList` is not mandatory but can be helpful in those cases where a concept scheme has been defined.

8.2 Hierarchical code lists

In some cases code lists have a hierarchical structure. In particular, this is used in SDMX when the data cube includes aggregations of data values (e.g. aggregating a measure across geographic regions). Hierarchical code lists SHOULD be represented using the `skos:narrower` relationship, or a sub-property of it, to link from the `skos:hasTopConcept` codes down through the tree or lattice of child codes. In some publishing tool chains the corresponding transitive closure `skos:narrowerTransitive` will be automatically inferred. The use of `skos:narrower` makes it possible to declare new concept schemes which extend an existing scheme by adding additional aggregation layers on top. All items are linked to the scheme via `skos:inScheme`.

8.3 Non-SKOS hierarchies

It is sometimes convenient to be able to specify a hierarchical arrangement of concepts other than through the use of the SKOS relation `skos:narrower`. There are several situations where this is useful, for example:

- In some cases publishers wish to be able to reuse existing reference data as their code lists. This particularly occurs where a geographic or admin-geographic hierarchy is already maintained by a separate authority but which uses non-SKOS containment or part-of relationships.
- Where such maintained reference data is to be reused there can be multiple hierarchies which relate the same codes. In particular a set of geographic entities may participate in both a geographic-containment hierarchy and an administrative hierarchy which do not precisely align.

The Data Cube vocabulary supports this situation through the `qb:HierarchicalCodeList` class. An instance of `qb:HierarchicalCodeList` defines a set of root concepts in the hierarchy (`qb:hierarchyRoot`) and a parent-to-child relationship (`qb:parentChildProperty`) which links a term in the hierarchy to its immediate sub-terms.

Thus a `qb:HierarchicalCodeList` is similar to a `skos:ConceptScheme` in which `qb:hierarchyRoot` plays the same role as `skos:hasTopConcept`, and the value of `qb:parentChildProperty` plays the same role as `skos:narrower`. In the case where a code list is already available as a SKOS concept scheme or collection, or could reasonable me made so, then those SHOULD be used directly. `qb:HierarchicalCodeList` is provided for cases where the terms are not available as SKOS but are available in some other RDF representation suitable for reuse.

For example, the Ordnance Survey of Great Britain publishes a geographic hierarchy which has eleven roots (European Regions such as Wales, Scotland, the South West) and uses a spatial relations ontology to define a containment hierarchy. This could be represented as a `qb:HierarchicalCodeList` using the following.

EXAMPLE 16

```
@prefix spatial: <http://data.ordnancesurvey.co.uk/ontology/spatialrelations/> .

eg:GBgeoHierarchy a qb:HierarchicalCodeList;
    rdfs:label "Geographic Hierarchy for Great Britain"@en;
    qb:hierarchyRoot
      <http://data.ordnancesurvey.co.uk/id/7000000000041427>, # South West
      <http://data.ordnancesurvey.co.uk/id/7000000000041426>, # West Midlands
      <http://data.ordnancesurvey.co.uk/id/7000000000041421>, # South East
      <http://data.ordnancesurvey.co.uk/id/7000000000041430>, # Yorkshire & the Humber
      <http://data.ordnancesurvey.co.uk/id/7000000000041423>, # East Midlands
      <http://data.ordnancesurvey.co.uk/id/7000000000041425>, # Eastern
      <http://data.ordnancesurvey.co.uk/id/7000000000041428>, # London
      <http://data.ordnancesurvey.co.uk/id/7000000000041431>, # North West
      <http://data.ordnancesurvey.co.uk/id/7000000000041422>, # North East
      <http://data.ordnancesurvey.co.uk/id/7000000000041424>, # Wales
      <http://data.ordnancesurvey.co.uk/id/7000000000041429>; # Scotland
    qb:parentChildProperty  spatial:contains;
    .

eg:geoDimension a qb:DimensionProperty ;
    qb:codeList eg:GBgeoHierarchy .
```

Note that in some cases the hierarchy to be reused may only have a property relating child concepts to parent concepts. This situation is handled by declaring the `qb:parentChildProperty` to be the `owl:inverseOf` of the child-to-parent property. For example:

EXAMPLE 17

```
@prefix spatial: <http://data.ordnancesurvey.co.uk/ontology/spatialrelations/> .

eg:GBgeoHierarchy a qb:HierarchicalCodeList;
    qb:parentChildProperty  [owl:inverseOf spatial:within] .
```

Future extensions of Data Cube may support additional sub classes of `qb:HierarchicalCodeList`, for example to declare hierarchies in which each parent is a disjoint union of its children.

8.4 Aggregation

The use of SKOS, or non-SKOS, hierarchies makes it possible to publish aggregated statistics for the non-leaf concepts in the hierarchy. The Data Cube vocabulary itself imposes no constraints on how such aggregation is done. Indeed in statistical applications the appropriate statistical corrections to make to aggregated values may be non-trivial and dependent on the data and precise analysis methodology. Similarly in other applications such as OLAP a number of different aggregation operators are commonly used.

Vocabulary terms to represent the aggregation operations employed within a given dataset, and how one dataset might be derived from another, are not supported in this version of the Data Cube specification. This area may be addressed by future extensions to Data Cube.

9. DataSet metadata

DataSets should be marked up with metadata to support discovery, presentation and processing. Dublin Core Terms [DC11] SHOULD be used for representing the key metadata annotations commonly needed for DataSets. The RDFS terms for display label (`rdfs:label`) descriptive comment (`rdfs:comment`) SHOULD be given as well for compatibility with earlier versions of Data Cube and common RDF practice.

The recommend core set of metadata terms is:

- `dct:title`
- `rdfs:label` - may be same as `dct:title`
- `dct:description`
- `rdfs:comment` - may be same as `dct:description`
- `dct:issued`
- `dct:modified`
- `dct:subject`
- `dct:publisher`
- `dct:license`

Other documents, notably [DCAT], provide additional recommendations for metadata terms for data sets which may be used for describing Data Cube DataSets.

9.1 Categorizing a data set

Publishers of statistics often categorize their data sets into different statistical domains, such as *Education*, *Labour*, or *Transportation*. We encourage use of `dct:subject` to record such a classification of a whole data set. The classification terms can include coarse grained classifications, such as the List of Subject-matter Domains from the SDMX Content-oriented Guidelines, and fine grained classifications to support discovery of data sets.

The classification schemes are typically represented using the SKOS vocabulary. For convenience the SMDX Subject-matter Domains have been encoded as a SKOS concept scheme at http://purl.org/linked-data/sdmx/2009/subject#.

Thus our sample dataset might be marked up by:

EXAMPLE 18

```
eg:dataset1 a qb:DataSet;
    rdfs:label      "Life expectancy"@en;
    dct:title       "Life expectancy"@en;
    rdfs:comment    "Life expectancy within Welsh Unitary authorities - extracted from Stats Wales"@en;
    dct:description "Life expectancy within Welsh Unitary authorities - extracted from Stats Wales"@en;
    dct:issued      "2010-08-11"^^xsd:date;
    dct:subject
        sdmx-subject:3.2 ,      # regional and small area statistics
        sdmx-subject:1.4 ,      # Health
        eg:Wales;               # Wales
    ...
```

where `eg:Wales` is a `skos:Concept` drawn from an appropriate controlled vocabulary for places.

9.2 Describing publishers

The organization that publishes a dataset should be recorded as part of the dataset metadata. Again we recommend use of the Dublin Core term `dct:publisher` for this. The organization should be represented as an instance of `foaf:Agent`, or some more specific subclass such as `org:Organization` [ORG].

EXAMPLE 19

```
eg:dataset1 a qb:DataSet;
    dct:publisher <http://example.com/meta#organization> .

<http://example.com/meta#organization> a org:Organization, foaf:Agent;
    rdfs:label "Example org" .
```

Extension vocabularies may provide additional metadata properties and may impose constraints on what metadata must be provided.

10. Abbreviated and normalized data cubes

In normal form then the `qb:Observation`s which make up a Data Cube have property values for each of the required dimensions, attributes and measures as declared in the associated data structure definition. This form for a Data Cube is termed ***normalized***. It is a convenient format for querying data and makes it possible to write uniform queries which extract sets of observations, including from across multiple cubes. However, the verbosity of a fully normalized representation incurs overheads in transmission and storage of Data Cubes which may be problematic in some settings. Note that abbreviated form is provided as an option and there is requirement that it be used. In many settings standard compression techniques can eliminate much of the overhead of normalized form.

To address this the Data Cube vocabulary supports a notion of an ***abbreviated*** format in which component properties may be ***attached*** to other levels in the Data Cube. Specifically they may be attached to a `qb:DataSet` or `qb:Slice`. In those cases the attached property is taken to be applied to all the `qb:Observation` instances associated with that attachment point. For illustration see example 4 in which the unit of measure is declared as to be attached to the whole data set and need not be repeated for every observation.

It is also possible to attach attributes to a `qb:MeasureProperty` in which case the attribute is intended to apply only to that property and not to the observations in which that property occurs.

10.1 Normalization algorithm

We define these notions by means of a transformation algorithm which can normalize an abbreviated Data Cube. We express this transformation using the SPARQL 1.1 Update language [sparql11-update]. Use of this notation does not imply that the transformation must be implemented this way. Information exchanges using Data Cube may retain data in abbreviated form and use other techniques such as query rewriting to ease access, may implement the normalization algorithm by other means or may handle all data in normalized form or any mix of these.

The normalization algorithm comprises two sets of SPARQL Update operations which should be applied in turn to a SPARQL Dataset in which the default graph contains the Data Cube RDF graph to be normalized.

The first update operation performs selective type and property closure operations. These serve two purposes. They ensure that `rdf:type` assertions on instances of `qb:Observation` and `qb:Slice` may be omitted in an abbreviated Data Cube. They also simplify the second set of update operations by expanding the sub properties of `qb:componentProperty` (specifically `qb:dimension`, `qb:measure` and `qb:attribute`).

Phase 1: Type and property closure

Phase 1: Type and property closure

```
PREFIX rdf:            <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX qb:             <http://purl.org/linked-data/cube#>

INSERT {
    ?o rdf:type qb:Observation .
} WHERE {
    [] qb:observation ?o .
};

INSERT {
    ?o  rdf:type qb:Observation .
    ?ds rdf:type qb:DataSet .
} WHERE {
    ?o qb:dataSet ?ds .
};

INSERT {
    ?s rdf:type qb:Slice .
} WHERE {
    [] qb:slice ?s.
};

INSERT {
    ?cs qb:componentProperty ?p .
    ?p  rdf:type qb:DimensionProperty .
} WHERE {
    ?cs qb:dimension ?p .
};

INSERT {
    ?cs qb:componentProperty ?p .
    ?p  rdf:type qb:MeasureProperty .
} WHERE {
    ?cs qb:measure ?p .
};

INSERT {
    ?cs qb:componentProperty ?p .
    ?p  rdf:type qb:AttributeProperty .
} WHERE {
    ?cs qb:attribute ?p .
}
```

These closure operations are implied by the RDFS semantics of the Data Cube vocabulary. Data Cube processors MAY apply full RDFS closure in place of the update operation defined here.

The second update operation checks the components of the data structure definition of the data set for declared attachment levels. For each of the possible attachments levels it looks for occurrences of that component to be pushed down to the corresponding observations.

Phase 2: Push down attachment levels

Phase 2: Push down attachment levels

```
PREFIX qb:            <http://purl.org/linked-data/cube#>

# Dataset attachments
INSERT {
    ?obs  ?comp ?value
} WHERE {
    ?spec    qb:componentProperty ?comp ;
             qb:componentAttachment qb:DataSet .
    ?dataset qb:structure [qb:component ?spec];
             ?comp ?value .
    ?obs     qb:dataSet ?dataset.
};

# Slice attachments
INSERT {
    ?obs  ?comp ?value
} WHERE {
    ?spec    qb:componentProperty ?comp;
             qb:componentAttachment qb:Slice .
    ?dataset qb:structure [qb:component ?spec];
             qb:slice ?slice .
    ?slice ?comp ?value;
           qb:observation ?obs .
};

# Dimension values on slices
INSERT {
    ?obs  ?comp ?value
} WHERE {
    ?spec    qb:componentProperty ?comp .
    ?comp a  qb:DimensionProperty .
    ?dataset qb:structure [qb:component ?spec];
             qb:slice ?slice .
    ?slice ?comp ?value;
           qb:observation ?obs .
}
```

11. Well-formed cubes

An instance of an RDF Data Cube should conform to a set of integrity constraints which we define in this section.

A ***well-formed*** RDF Data Cube is an a RDF graph describing one or more instances of `qb:DataSet` for which each of the integrity checks defined here passes.

A ***well-formed abbreviated*** RDF Data Cube is an a RDF graph which, when expanded using the normalization algorithm, yields a well-formed RDF Data Cube.

11.1 Integrity constraints

Each integrity constraint is expressed as narrative prose and, where possible, a SPARQL [sparql11-query] ASK query or query template. If the ASK query is applied to an RDF graph then it will return *true* if that graph contains one or more Data Cube instances which violate the corresponding constraint.

Using SPARQL queries to express the integrity constraints does not imply that integrity checking must be performed this way. Implementations are free to use alternative query formulations or alternative implementation techniques to perform equivalent checks.

Each integrity constraint query assumes the following set of prefix bindings:

```
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:    <http://www.w3.org/2000/01/rdf-schema#>
PREFIX skos:    <http://www.w3.org/2004/02/skos/core#>
PREFIX qb:      <http://purl.org/linked-data/cube#>
PREFIX xsd:     <http://www.w3.org/2001/XMLSchema#>
PREFIX owl:     <http://www.w3.org/2002/07/owl#>
```

The complete set of constraints is listed below.

IC-0. Datatype consistency

The RDF graph must be consistent under RDF D-entailment [RDF-MT] using a datatype map containing all the datatypes used within the graph.

IC-1. Unique DataSet

Every `qb:Observation` has exactly one associated `qb:DataSet`.

```
ASK {
```

```
  {
    # Check observation has a data set
    ?obs a qb:Observation .
    FILTER NOT EXISTS { ?obs qb:dataSet ?dataset1 . }
  } UNION {
    # Check has just one data set
    ?obs a qb:Observation ;
       qb:dataSet ?dataset1, ?dataset2 .
    FILTER (?dataset1 != ?dataset2)
  }
}
```

IC-2. Unique DSD

Every qb:DataSet has exactly one associated qb:DataStructureDefinition.

```
ASK {
  {
    # Check dataset has a dsd
    ?dataset a qb:DataSet .
    FILTER NOT EXISTS { ?dataset qb:structure ?dsd . }
  } UNION {
    # Check has just one dsd
    ?dataset a qb:DataSet ;
       qb:structure ?dsd1, ?dsd2 .
    FILTER (?dsd1 != ?dsd2)
  }
}
```

IC-3. DSD includes measure

Every qb:DataStructureDefinition must include at least one declared measure.

```
ASK {
  ?dsd a qb:DataStructureDefinition .
  FILTER NOT EXISTS { ?dsd qb:component [qb:componentProperty [a qb:MeasureProperty]] }
}
```

IC-4. Dimensions have range

Every dimension declared in a qb:DataStructureDefinition must have a declared rdfs:range.

```
ASK {
  ?dim a qb:DimensionProperty .
  FILTER NOT EXISTS { ?dim rdfs:range [] }
}
```

IC-5. Concept dimensions have code lists

Every dimension with range skos:Concept must have a qb:codeList.

```
ASK {
  ?dim a qb:DimensionProperty ;
       rdfs:range skos:Concept .
  FILTER NOT EXISTS { ?dim qb:codeList [] }
}
```

IC-6. Only attributes may be optional

The only components of a qb:DataStructureDefinition that may be marked as optional, using qb:componentRequired are attributes.

```
ASK {
  ?dsd qb:component ?componentSpec .
  ?componentSpec qb:componentRequired "false"^^xsd:boolean ;
                 qb:componentProperty ?component .
  FILTER NOT EXISTS { ?component a qb:AttributeProperty }
```

```
}
```

IC-7. Slice Keys must be declared

Every qb:SliceKey must be associated with a qb:DataStructureDefinition.

```
ASK {
    ?sliceKey a qb:SliceKey .
    FILTER NOT EXISTS { [a qb:DataStructureDefinition] qb:sliceKey ?sliceKey }
}
```

IC-8. Slice Keys consistent with DSD

Every qb:componentProperty on a qb:SliceKey must also be declared as a qb:component of the associated qb:DataStructureDefinition.

```
ASK {
  ?slicekey a qb:SliceKey;
      qb:componentProperty ?prop .
  ?dsd qb:sliceKey ?slicekey .
  FILTER NOT EXISTS { ?dsd qb:component [qb:componentProperty ?prop] }
}
```

IC-9. Unique slice structure

Each qb:Slice must have exactly one associated qb:sliceStructure.

```
ASK {
  {
    # Slice has a key
    ?slice a qb:Slice .
    FILTER NOT EXISTS { ?slice qb:sliceStructure ?key }
  } UNION {
    # Slice has just one key
    ?slice a qb:Slice ;
           qb:sliceStructure ?key1, ?key2;
    FILTER (?key1 != ?key2)
  }
}
```

IC-10. Slice dimensions complete

Every qb:Slice must have a value for every dimension declared in its qb:sliceStructure.

```
ASK {
  ?slice qb:sliceStructure [qb:componentProperty ?dim] .
  FILTER NOT EXISTS { ?slice ?dim [] }
}
```

IC-11. All dimensions required

Every qb:Observation has a value for each dimension declared in its associated qb:DataStructureDefinition.

```
ASK {
    ?obs qb:dataSet/qb:structure/qb:component/qb:componentProperty ?dim .
    ?dim a qb:DimensionProperty;
    FILTER NOT EXISTS { ?obs ?dim [] }
}
```

IC-12. No duplicate observations

No two qb:Observations in the same qb:DataSet may have the same value for all dimensions.

```
```

```
ASK {
  FILTER( ?allEqual )
  {
    # For each pair of observations test if all the dimension values are the same
    SELECT (MIN(?equal) AS ?allEqual) WHERE {
        ?obs1 qb:dataSet ?dataset .
        ?obs2 qb:dataSet ?dataset .
        FILTER (?obs1 != ?obs2)
        ?dataset qb:structure/qb:component/qb:componentProperty ?dim .
        ?dim a qb:DimensionProperty .
        ?obs1 ?dim ?value1 .
        ?obs2 ?dim ?value2 .
        BIND( ?value1 = ?value2 AS ?equal)
    } GROUP BY ?obs1 ?obs2
  }
}
```

IC-13. Required attributes

Every qb:Observation has a value for each declared attribute that is marked as required.

```
ASK {
    ?obs qb:dataSet/qb:structure/qb:component ?component .
    ?component qb:componentRequired "true"^^xsd:boolean ;
               qb:componentProperty ?attr .
    FILTER NOT EXISTS { ?obs ?attr [] }
}
```

IC-14. All measures present

In a qb:DataSet which does not use a Measure dimension then each individual qb:Observation must have a value for every declared measure.

```
ASK {
    # Observation in a non-measureType cube
    ?obs qb:dataSet/qb:structure ?dsd .
    FILTER NOT EXISTS { ?dsd qb:component/qb:componentProperty qb:measureType }

    # verify every measure is present
    ?dsd qb:component/qb:componentProperty ?measure .
    ?measure a qb:MeasureProperty;
    FILTER NOT EXISTS { ?obs ?measure [] }
}
```

IC-15. Measure dimension consistent

In a qb:DataSet which uses a Measure dimension then each qb:Observation must have a value for the measure corresponding to its given qb:measureType.

```
ASK {
    # Observation in a measureType-cube
    ?obs qb:dataSet/qb:structure ?dsd ;
         qb:measureType ?measure .
    ?dsd qb:component/qb:componentProperty qb:measureType .
    # Must have value for its measureType
    FILTER NOT EXISTS { ?obs ?measure [] }
}
```

IC-16. Single measure on measure dimension observation

In a qb:DataSet which uses a Measure dimension then each qb:Observation must only have a value for one measure (by IC-15 this will be the measure corresponding to its qb:measureType).

```
ASK {
    # Observation with measureType
    ?obs qb:dataSet/qb:structure ?dsd ;
         qb:measureType ?measure ;
         ?omeasure [] .
    # Any measure on the observation
    ?dsd qb:component/qb:componentProperty qb:measureType ;
         qb:component/qb:componentProperty ?omeasure .
    ?omeasure a qb:MeasureProperty .
```

```
    # Must be the same as the measureType
    FILTER (?omeasure != ?measure)
}
```

IC-17. All measures present in measures dimension cube

In a qb:DataSet which uses a Measure dimension then if there is a Observation for some combination of non-measure dimensions then there must be other Observations with the same non-measure dimension values for each of the declared measures.

```
ASK {
  {
      # Count number of other measures found at each point
      SELECT ?numMeasures (COUNT(?obs2) AS ?count) WHERE {
          {
              # Find the DSDs and check how many measures they have
              SELECT ?dsd (COUNT(?m) AS ?numMeasures) WHERE {
                  ?dsd qb:component/qb:componentProperty ?m.
                  ?m a qb:MeasureProperty .
              } GROUP BY ?dsd
          }

          # Observation in measureType cube
          ?obs1 qb:dataSet/qb:structure ?dsd;
                qb:dataSet ?dataset ;
                qb:measureType ?m1 .

          # Other observation at same dimension value
          ?obs2 qb:dataSet ?dataset ;
                qb:measureType ?m2 .
          FILTER NOT EXISTS {
              ?dsd qb:component/qb:componentProperty ?dim .
              FILTER (?dim != qb:measureType)
              ?dim a qb:DimensionProperty .
              ?obs1 ?dim ?v1 .
              ?obs2 ?dim ?v2.
              FILTER (?v1 != ?v2)
          }

      } GROUP BY ?obs1 ?numMeasures
        HAVING (?count != ?numMeasures)
  }
}
```

IC-18. Consistent data set links

If a qb:DataSet D has a qb:slice S, and S has an qb:observation O, then the qb:dataSet corresponding to O must be D.

```
ASK {
    ?dataset qb:slice       ?slice .
    ?slice   qb:observation ?obs .
    FILTER NOT EXISTS { ?obs qb:dataSet ?dataset . }
}
```

IC-19. Codes from code list

If a dimension property has a qb:codeList, then the value of the dimension property on every qb:Observation must be in the code list.

The following integrity check queries must be applied to an RDF graph which contains the definition of the code list as well as the Data Cube to be checked. In the case of a skos:ConceptScheme then each concept must be linked to the scheme using skos:inScheme. In the case of a skos:Collection then the collection must link to each concept (or to nested collections) using skos:member. If the collection uses skos:memberList then the entailment of skos:member values defined by S36 in [SKOS-REFERENCE] must be materialized before this check is applied.

```
ASK {
    ?obs qb:dataSet/qb:structure/qb:component/qb:componentProperty ?dim .
    ?dim a qb:DimensionProperty ;
        qb:codeList ?list .
    ?list a skos:ConceptScheme .
    ?obs ?dim ?v .
    FILTER NOT EXISTS { ?v a skos:Concept ; skos:inScheme ?list }
}

ASK {
    ?obs qb:dataSet/qb:structure/qb:component/qb:componentProperty ?dim .
```

```
    ?dim a qb:DimensionProperty ;
        qb:codeList ?list .
    ?list a skos:Collection .
    ?obs ?dim ?v .
    FILTER NOT EXISTS { ?v a skos:Concept . ?list skos:member+ ?v }
}
```

IC-20. Codes from hierarchy

If a dimension property has a qb:HierarchicalCodeList with a non-blank qb:parentChildProperty then the value of that dimension property on every qb:Observation must be reachable from a root of the hierarchy using zero or more hops along the qb:parentChildProperty links.

This check cannot be made by a simple fixed SPARQL query. Instead a query template is supplied. An instance of the template should be generated for each qb:HierarchicalCodeList which has an IRI value for its qb:parentChildProperty. That is for each binding of ?p in the following instantiation query:

```
SELECT ?p WHERE {
    ?hierarchy a qb:HierarchicalCodeList ;
                qb:parentChildProperty ?p .
    FILTER ( isIRI(?p) )
}
```

The template is then instantiated by replacing the string $p by the IRI found by the instantiation query. The template is:

```
ASK {
    ?obs qb:dataSet/qb:structure/qb:component/qb:componentProperty ?dim .
    ?dim a qb:DimensionProperty ;
        qb:codeList ?list .
    ?list a qb:HierarchicalCodeList .
    ?obs ?dim ?v .
    FILTER NOT EXISTS { ?list qb:hierarchyRoot/<$p>* ?v }
}
```

IC-21. Codes from hierarchy (inverse)

If a dimension property has a qb:HierarchicalCodeList with an inverse qb:parentChildProperty then the value of that dimension property on every qb:Observation must be reachable from a root of the hierarchy using zero or more hops along the inverse qb:parentChildProperty links.

This check cannot be made by a simple fixed SPARQL query. Instead a query template is supplied. An instance of the template should be generated for each qb:HierarchicalCodeList which has a blank-node value for its qb:parentChildProperty, with an associated inverse property. That is for each binding of ?p in the following instantiation query:

```
SELECT ?p WHERE {
    ?hierarchy a qb:HierarchicalCodeList;
                qb:parentChildProperty ?pcp .
    FILTER( isBlank(?pcp) )
    ?pcp  owl:inverseOf ?p .
    FILTER( isIRI(?p) )
}
```

The template is then instantiated by replacing the string $p by the IRI found by the instantiation query. The template is:

```
ASK {
    ?obs qb:dataSet/qb:structure/qb:component/qb:componentProperty ?dim .
    ?dim a qb:DimensionProperty ;
         qb:codeList ?list .
    ?list a qb:HierarchicalCodeList .
    ?obs ?dim ?v .
    FILTER NOT EXISTS { ?list qb:hierarchyRoot/(^<$p>)* ?v }
}
```

12. Vocabulary reference

12.1 DataSets

See Section Expressing data sets.

Class: `qb:DataSet` ***Sub class of:*** `qb:Attachable` ***Equivalent to:*** `scovo:Dataset`

Represents a collection of observations, possibly organized into various slices, conforming to some common dimensional structure.

12.2 Observations

See Section Expressing data sets.

Class: `qb:Observation` ***Sub class of:*** `qb:Attachable` ***Equivalent to:*** `scovo:Item`
A single observation in the cube, may have one or more associated measured values.

Property: `qb:dataSet` **(** ***Domain:*** `qb:Observation` ***-> Range:*** `qb:DataSet` **)**
Indicates the data set of which this observation is a part.

Property: `qb:observation` **(** ***Domain:*** `qb:ObservationGroup` ***-> Range:*** `qb:Observation` **)**
Indicates a observation contained within this slice of the data set.

12.3 Slices

See Section Slices.

Class: `qb:ObservationGroup`
A, possibly arbitrary, group of observations.

Class: `qb:Slice` ***Sub class of:*** `qb:Attachable`, `qb:ObservationGroup`
Denotes a subset of a DataSet defined by fixing a subset of the dimensional values, component properties on the Slice.

Property: `qb:slice` **(** ***Domain:*** `qb:DataSet` ***-> Range:*** `qb:Slice`; ***sub property of:*** `qb:observationGroup` **)**
Indicates a subset of a DataSet defined by fixing a subset of the dimensional values.

Property: `qb:observationGroup` **(** ***Domain: -> Range:*** `qb:ObservationGroup` **)**
Indicates a group of observations. The domain of this property is left open so that a group may be attached to different resources and need not be restricted to a single DataSet.

12.4 Dimensions, Attributes, Measures

See Section Dimensions, attributes and measures.

Class: `qb:Attachable`
Abstract superclass for everything that can have attributes and dimensions.

Class: `qb:ComponentProperty` ***Sub class of:*** `rdf:Property`
Abstract super-class of all properties representing dimensions, attributes or measures.

Class: `qb:DimensionProperty` ***Sub class of:*** `qb:ComponentProperty`, `qb:CodedProperty`
The class of component properties which represent the dimensions of the cube.

Class: `qb:AttributeProperty` ***Sub class of:*** `qb:ComponentProperty`
The class of component properties which represent attributes of observations in the cube, e.g. unit of measurement.

Class: `qb:MeasureProperty` ***Sub class of:*** `qb:ComponentProperty`
The class of component properties which represent the measured value of the phenomenon being observed.

Class: `qb:CodedProperty` ***Sub class of:*** `qb:ComponentProperty`
Superclass of all coded component properties.

12.5 Reusable general purpose component properties

See Section Measure dimensions.

Property: `qb:measureType` **(** ***Domain: -> Range:*** `qb:MeasureProperty` **)**
Generic measure dimension, the value of this dimension indicates which measure (from the set of measures in the DSD) is being given by the observation.

12.6 Data Structure Definitions

See Section ComponentSpecifications and DataStructureDefinitions.

Class: `qb:DataStructureDefinition` ***Sub class of:*** `qb:ComponentSet`
Defines the structure of a DataSet or slice.

Property: `qb:structure` **(** ***Domain:*** `qb:DataSet` ***-> Range:*** `qb:DataStructureDefinition` **)**
Indicates the structure to which this data set conforms

Property: `qb:component` **(** ***Domain:*** `qb:DataStructureDefinition` ***-> Range:*** `qb:ComponentSpecification` **)**
Indicates a component specification which is included in the structure of the dataset.

12.7 Component specifications - for qualifying component use in a DSD

See Section ComponentSpecifications and DataStructureDefinitions.

Class: `qb:ComponentSpecification` ***Sub class of:*** `qb:ComponentSet`
Used to define properties of a component (attribute, dimension etc) which are specific to its usage in a DSD.

Class: `qb:ComponentSet`
Abstract class of things which reference one or more ComponentProperties

Property: `qb:componentProperty` **(*Domain:*** `qb:ComponentSet` ***-> Range:*** `qb:ComponentProperty` **)**
Indicates a ComponentProperty (i.e. attribute/dimension) expected on a DataSet, or a dimension fixed in a SliceKey.

Property: `qb:order` **(*Domain:*** `qb:ComponentSpecification` ***-> Range:*** `xsd:int` **)**
Indicates a priority order for the components of sets with this structure, used to guide presentations - lower order numbers come before higher numbers, un-numbered components come last.

Property: `qb:componentRequired` **(*Domain:*** `qb:ComponentSpecification` ***-> Range:*** `xsd:boolean` **)**
Indicates whether a component property is required (true) or optional (false) in the context of a DSD. Only applicable to components corresponding to an attribute. Defaults to false (optional).

Property: `qb:componentAttachment` **(*Domain:*** `qb:ComponentSpecification` ***-> Range:*** `rdfs:Class` **)**
Indicates the level at which the component property should be attached, this might be an qb:DataSet, qb:Slice or qb:Observation, or a qb:MeasureProperty.

Property: `qb:dimension` **(*Domain: -> Range:*** `qb:DimensionProperty` **; *sub property of:*** `qb:componentProperty` **)**
An alternative to qb:componentProperty which makes explicit that the component is a dimension.

Property: `qb:measure` **(*Domain: -> Range:*** `qb:MeasureProperty` **; *sub property of:*** `qb:componentProperty` **)**
An alternative to qb:componentProperty which makes explicit that the component is a measure.

Property: `qb:attribute` **(*Domain: -> Range:*** `qb:AttributeProperty` **; *sub property of:*** `qb:componentProperty` **)**
An alternative to qb:componentProperty which makes explicit that the component is a attribute.

Property: `qb:measureDimension` **(*Domain: -> Range:*** `qb:DimensionProperty` **; *sub property of:*** `qb:componentProperty` **)**
An alternative to qb:componentProperty which makes explicit that the component is a measure dimension.

12.8 Slice definitions

See Section Slices.

Class: `qb:SliceKey` ***Sub class of:*** `qb:ComponentSet`
Denotes a subset of the component properties of a DataSet which are fixed in the corresponding slices.

Property: `qb:sliceStructure` **(*Domain:*** `qb:Slice` ***-> Range:*** `qb:SliceKey` **)**
Indicates the slice key corresponding to this slice.

Property: `qb:sliceKey` **(*Domain:*** `qb:DataStructureDefinition` ***-> Range:*** `qb:SliceKey` **)**
Indicates a slice key which is used for slices in this dataset.

12.9 Concepts

See Section Concept schemes and code lists.

Property: `qb:concept` **(*Domain:*** `qb:ComponentProperty` ***-> Range:*** `skos:Concept` **)**
Gives the concept which is being measured or indicated by a ComponentProperty.

Property: `qb:codeList` **(*Domain:*** `qb:CodedProperty` ***-> Range:*** `owl:unionOf(skos:ConceptScheme skos:Collection qb:HierarchicalCodeList)` **)**
Gives the code list associated with a CodedProperty.

12.10 Non-SKOS Hierarchies

See Section Non-SKOS hierarchies.

Class: `qb:HierarchicalCodeList`
Represents a generalized hierarchy of concepts which can be used for coding. The hierarchy is defined by one or more roots together with a property which relates concepts in the hierarchy to their child concept . The same concepts may be members of multiple hierarchies provided that different qb:parentChildProperty values are used for each hierarchy.

Property: `qb:hierarchyRoot` **(*Domain:*** `qb:HierarchicalCodeList` **)**
Specifies a root of the hierarchy. A hierarchy may have multiple roots but must have at least one.

Property: `qb:parentChildProperty` **(*Domain:*** `qb:HierarchicalCodeList` ***-> Range:*** `rdf:Property` **)**
Specifies a property which relates a parent concept in the hierarchy to a child concept. Note that a child may have more than one parent.

A. Acknowledgements

This work is based on a collaboration that was initiated in a workshop on Publishing statistical datasets in SDMX and the semantic web, hosted by ONS in Sunningdale, United Kingdom in February 2010 and continued at the ODaF 2010 workshop in Tilburg. The authors would like to thank all the participants at those workshops for their input into this work but especially Arofan Gregory for his patient explanations of SDMX and insight in the need and requirements for a core Data Cube representation.

The editors would like to thank John Sheridan for his comments, suggestions and support for the original work.

Many individuals provided valuable comments on this specification as it made its way through the W3C process. We would like to especially acknowledge the contributions of Benedikt Kaempgen, Sarven Capadisli and Curran Kelleher.

B. Change history

Changes since the W3C Proposed Recommendation 17 December 2013: None.

- Corrected typo in reference to `eg:Wales` in example 18.
- Fixed broken like to SCOVO reference [HAUS09].
- Added namespace, and links to vocabulary file, to the Abstract for ease of reference.
- Added missing closure rule to infer `rdf:type` of `qb:DataSet` from its use in a `qb:Observation`.
- Corrected mis-statement of domain of `qb:sliceKey` in the reference section (was correct in the rest of the specification).
- Corrected typo in statement of IC-8.
- Minor typographical corrections.
- Updated references, including moving Turtle to a non-normative reference.
- Removed CR specific text on implementation feedback and At Risk features.

Changes since W3C Last Call Working Draft 12 March 2013:

- Section 1. Modified diagram to clarify that `qb:Slice` is a sub class of `qb:ObservationGroup`
- Section 5.1. Moved description of data set here for greater clarity, was in section 7.
- Section 6.2. Further clarified the status of COG RDF vocabularies.
- Example 6.3. Retitled for clarity.
- Section 6.4, corrected "with multiple observations" to "with multiple measures".
- Section 6.4. Added mention of `qb:Attachable` (previously only mentioned in the vocabulary reference).
- Section 6.5. Rewritten description of handling of multiple measures for greater clarity.
- Section 6.5.2. Removed reference to "SDMX-in-RDF vocabulary" which is not part of this specification.
- Section 6.5.2 clarified that the special nature of `qb:measureType` is a divergence from SDMX.
- Section 7. Change terminology from "internal and external metadata" to "structural and reference" metadata for compatibility with SDMX terminology.
- Section 8.2. Clarified that sub properties of `skos:narrower` may be used.
- Section 8.3. Removed third bullet on the motivation for `qb:HierarchicalCodeList` (exhaustivity and mutual exclusion).
- Added reference to SDMX User Guide as additional background context.
- Added complete example Data Cube as an appendix, providing a link to it from earlier on in the document.
- Section 10. Clarified that abbreviated form is an option, and there is no requirement to use it in place of normalized form.
- Section 12.2. Corrected domain of `qb:observation` from `qb:Slice` to `qb:ObservationGroup`, was given correctly in the ontology and text body but was mis-stated in the reference section.
- References. Updated ORG reference.
- Various typographical corrections.

Changes since W3C Working Draft 5 April 2012:

- Added section on criteria for well-formed cubes. ISSUE-29
- Added section on normalization algorithm for handling abbreviated cubes.
- Added conformance section.
- Clarified that `qb:componentRequired` is only applicable to attributes and that it defaults to optional.
- Moved vocabulary reference into the normative body of the specification, adding hyperlinks for all qb: terms.
- Added section on non-skos hierarchies. ISSUE-31.
- Added note that aggregation operations and inter-cube relations are out of scope for this version. ISSUE-30.
- Added `qb:ObservationGroup` as a generalization of `qb:Slice`. ISSUE-33.
- Removed `qb:subSlice` as being problematic in how they interact with attachment levels. ISSUE-34.
- Generalized range of `qb:codeList` to allow `skos:Collection`. ISSUE-39.
- Moved namespace definitions to a normative section within the body of the specification.
- Moved Jeni Tennison from being listed as an author to being a contributor.

C. Complete example Data Cube

This is a complete Data Cube encoding of the running example introduced in section 5.4. It uses the abbreviated format so that it can be concisely presented. It passes all the integrity checks (when the declaration of `sdmx-dimension:sex` is included from http://purl.org/linked-data/sdmx/2009/dimension) and so is a well-formed abbreviated Data Cube.

```
@prefix rdf:       <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:      <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl:       <http://www.w3.org/2002/07/owl#> .
```

```
@prefix xsd:      <http://www.w3.org/2001/XMLSchema#> .
@prefix skos:     <http://www.w3.org/2004/02/skos/core#> .
@prefix void:     <http://rdfs.org/ns/void#> .
@prefix dct:      <http://purl.org/dc/terms/> .
@prefix foaf:     <http://xmlns.com/foaf/0.1/> .
@prefix org:      <http://www.w3.org/ns/org#> .
@prefix admingeo: <http://data.ordnancesurvey.co.uk/ontology/admingeo/> .
@prefix interval: <http://reference.data.gov.uk/def/intervals/> .

@prefix qb:       <http://purl.org/linked-data/cube#> .

@prefix sdmx-concept:    <http://purl.org/linked-data/sdmx/2009/concept#> .
@prefix sdmx-dimension:  <http://purl.org/linked-data/sdmx/2009/dimension#> .
@prefix sdmx-attribute:  <http://purl.org/linked-data/sdmx/2009/attribute#> .
@prefix sdmx-measure:    <http://purl.org/linked-data/sdmx/2009/measure#> .
@prefix sdmx-metadata:   <http://purl.org/linked-data/sdmx/2009/metadata#> .
@prefix sdmx-code:       <http://purl.org/linked-data/sdmx/2009/code#> .
@prefix sdmx-subject:    <http://purl.org/linked-data/sdmx/2009/subject#> .

@prefix ex-geo:   <http://example.org/geo#> .
@prefix eg:       <http://example.org/ns#> .

# -- Data Set ------------------------------------------

eg:dataset-le3 a qb:DataSet;
    dct:title       "Life expectancy"@en;
    rdfs:label      "Life expectancy"@en;
    rdfs:comment    "Life expectancy within Welsh Unitary authorities - extracted from Stats Wales"@en;
    dct:description "Life expectancy within Welsh Unitary authorities - extracted from Stats Wales"@en;
    dct:publisher   eg:organization ;
    dct:issued      "2010-08-11"^^xsd:date;
    dct:subject
        sdmx-subject:3.2 ,      # regional and small area statistics
        sdmx-subject:1.4 ,      # Health
        ex-geo:wales;           # Wales
    qb:structure eg:dsd-le3 ;
    sdmx-attribute:unitMeasure <http://dbpedia.org/resource/Year> ;
    qb:slice eg:slice1, eg:slice2, eg:slice3, eg:slice4, eg:slice5, eg:slice6 ;
    .

eg:organization a org:Organization, foaf:Agent;
    rdfs:label "Example org"@en .

# -- Data structure definition --------------------------

eg:dsd-le3 a qb:DataStructureDefinition;
    qb:component
    # The dimensions
        [ qb:dimension eg:refArea;         qb:order 1 ],
        [ qb:dimension eg:refPeriod;       qb:order 2; qb:componentAttachment qb:Slice ],
        [ qb:dimension sdmx-dimension:sex; qb:order 3; qb:componentAttachment qb:Slice ];

    # The measure(s)
    qb:component [ qb:measure eg:lifeExpectancy];

    # The attributes
    qb:component [ qb:attribute sdmx-attribute:unitMeasure;
                   qb:componentRequired "true"^^xsd:boolean;
                   qb:componentAttachment qb:DataSet; ] ;

    # slices
    qb:sliceKey eg:sliceByRegion ;
    .

eg:sliceByRegion a qb:SliceKey;
    rdfs:label "slice by region"@en;
    rdfs:comment "Slice by grouping regions together, fixing sex and time values"@en;
    qb:componentProperty eg:refPeriod, sdmx-dimension:sex ;
    .

# -- Dimensions and measures  --------------------------

eg:refPeriod  a rdf:Property, qb:DimensionProperty;
    rdfs:label "reference period"@en;
    rdfs:subPropertyOf sdmx-dimension:refPeriod;
    rdfs:range interval:Interval;
    qb:concept sdmx-concept:refPeriod ;
    .


eg:refArea  a rdf:Property, qb:DimensionProperty;
    rdfs:label "reference area"@en;
    rdfs:subPropertyOf sdmx-dimension:refArea;
    rdfs:range admingeo:UnitaryAuthority;
    qb:concept sdmx-concept:refArea ;
    .

eg:lifeExpectancy  a rdf:Property, qb:MeasureProperty;
    rdfs:label "life expectancy"@en;
    rdfs:subPropertyOf sdmx-measure:obsValue;
    rdfs:range xsd:decimal ;
```

```
    .

# -- Observations ---------------------------------------

# Column 1

eg:slice1 a qb:Slice;
    qb:sliceStructure  eg:sliceByRegion ;
    eg:refPeriod               <http://reference.data.gov.uk/id/gregorian-interval/2004-01-01T00:00:00/P3Y> ;
    sdmx-dimension:sex         sdmx-code:sex-M ;
    qb:observation eg:o11, eg:o12, eg:o13, eg:o14 ;
    .

eg:o11 a qb:Observation;
    qb:dataSet  eg:dataset-le3 ;
    eg:refArea                 ex-geo:newport_00pr ;
    eg:lifeExpectancy          76.7 ;
    .

eg:o12 a qb:Observation;
    qb:dataSet  eg:dataset-le3 ;
    eg:refArea                 ex-geo:cardiff_00pt ;
    eg:lifeExpectancy          78.7 ;
    .

eg:o13 a qb:Observation;
    qb:dataSet  eg:dataset-le3 ;
    eg:refArea                 ex-geo:monmouthshire_00pp ;
    eg:lifeExpectancy          76.6 ;
    .

eg:o14 a qb:Observation;
    qb:dataSet  eg:dataset-le3 ;
    eg:refArea                 ex-geo:merthyr_tdfil_00ph ;
    eg:lifeExpectancy          75.5 ;
    .

# Column 2

eg:slice2 a qb:Slice;
    qb:sliceStructure  eg:sliceByRegion ;
    eg:refPeriod               <http://reference.data.gov.uk/id/gregorian-interval/2004-01-01T00:00:00/P3Y> ;
    sdmx-dimension:sex         sdmx-code:sex-F ;
    qb:observation eg:o21, eg:o22, eg:o23, eg:o24 ;
    .

eg:o21 a qb:Observation;
    qb:dataSet  eg:dataset-le3 ;
    eg:refArea                 ex-geo:newport_00pr ;
    eg:lifeExpectancy          80.7 ;
    .

eg:o22 a qb:Observation;
    qb:dataSet  eg:dataset-le3 ;
    eg:refArea                 ex-geo:cardiff_00pt ;
    eg:lifeExpectancy          83.3 ;
    .

eg:o23 a qb:Observation;
    qb:dataSet  eg:dataset-le3 ;
    eg:refArea                 ex-geo:monmouthshire_00pp ;
    eg:lifeExpectancy          81.3 ;
    .

eg:o24 a qb:Observation;
    qb:dataSet  eg:dataset-le3 ;
    eg:refArea                 ex-geo:merthyr_tdfil_00ph ;
    eg:lifeExpectancy          79.1 ;
    .

# Column 3

eg:slice3 a qb:Slice;
    qb:sliceStructure  eg:sliceByRegion ;
    eg:refPeriod               <http://reference.data.gov.uk/id/gregorian-interval/2005-01-01T00:00:00/P3Y> ;
    sdmx-dimension:sex         sdmx-code:sex-M ;
    qb:observation eg:o31, eg:o32, eg:o33, eg:o34 ;
    .

eg:o31 a qb:Observation;
    qb:dataSet  eg:dataset-le3 ;
    eg:refArea                 ex-geo:newport_00pr ;
    eg:lifeExpectancy          77.1 ;
    .

eg:o32 a qb:Observation;
    qb:dataSet  eg:dataset-le3 ;
    eg:refArea                 ex-geo:cardiff_00pt ;
    eg:lifeExpectancy          78.6 ;
    .
```

```
eg:o33 a qb:Observation;
    qb:dataSet  eg:dataset-le3 ;
    eg:refArea                  ex-geo:monmouthshire_00pp ;
    eg:lifeExpectancy           76.5 ;
    .

eg:o34 a qb:Observation;
    qb:dataSet  eg:dataset-le3 ;
    eg:refArea                  ex-geo:merthyr_tdfil_00ph ;
    eg:lifeExpectancy           75.5 ;
    .

# Column 4

eg:slice4 a qb:Slice;
    qb:sliceStructure  eg:sliceByRegion ;
    eg:refPeriod               <http://reference.data.gov.uk/id/gregorian-interval/2005-01-01T00:00:00/P3Y> ;
    sdmx-dimension:sex         sdmx-code:sex-F ;
    qb:observation eg:o41, eg:o42, eg:o43, eg:o44 ;
    .

eg:o41 a qb:Observation;
    qb:dataSet  eg:dataset-le3 ;
    eg:refArea                  ex-geo:newport_00pr ;
    eg:lifeExpectancy           80.9 ;
    .

eg:o42 a qb:Observation;
    qb:dataSet  eg:dataset-le3 ;
    eg:refArea                  ex-geo:cardiff_00pt ;
    eg:lifeExpectancy           83.7 ;
    .

eg:o43 a qb:Observation;
    qb:dataSet  eg:dataset-le3 ;
    eg:refArea                  ex-geo:monmouthshire_00pp ;
    eg:lifeExpectancy           81.5 ;
    .

eg:o44 a qb:Observation;
    qb:dataSet  eg:dataset-le3 ;
    eg:refArea                  ex-geo:merthyr_tdfil_00ph ;
    eg:lifeExpectancy           79.4 ;
    .

# Column 5

eg:slice5 a qb:Slice;
    qb:sliceStructure  eg:sliceByRegion ;
    eg:refPeriod               <http://reference.data.gov.uk/id/gregorian-interval/2006-01-01T00:00:00/P3Y> ;
    sdmx-dimension:sex         sdmx-code:sex-M ;
    qb:observation eg:o51, eg:o52, eg:o53, eg:o54 ;
    .

eg:o51 a qb:Observation;
    qb:dataSet  eg:dataset-le3 ;
    eg:refArea                  ex-geo:newport_00pr ;
    eg:lifeExpectancy           77.0 ;
    .

eg:o52 a qb:Observation;
    qb:dataSet  eg:dataset-le3 ;
    eg:refArea                  ex-geo:cardiff_00pt ;
    eg:lifeExpectancy           78.7 ;
    .

eg:o53 a qb:Observation;
    qb:dataSet  eg:dataset-le3 ;
    eg:refArea                  ex-geo:monmouthshire_00pp ;
    eg:lifeExpectancy           76.6 ;
    .

eg:o54 a qb:Observation;
    qb:dataSet  eg:dataset-le3 ;
    eg:refArea                  ex-geo:merthyr_tdfil_00ph ;
    eg:lifeExpectancy           74.9 ;
    .

# Column 6

eg:slice6 a qb:Slice;
    qb:sliceStructure  eg:sliceByRegion ;
    eg:refPeriod               <http://reference.data.gov.uk/id/gregorian-interval/2006-01-01T00:00:00/P3Y> ;
    sdmx-dimension:sex         sdmx-code:sex-F ;
    qb:observation eg:o61, eg:o62, eg:o63, eg:o64 ;
    .

eg:o61 a qb:Observation;
    qb:dataSet  eg:dataset-le3 ;
    eg:refArea                  ex-geo:newport_00pr ;
    eg:lifeExpectancy           81.5 ;
```

```
    .

eg:o62 a qb:Observation;
    qb:dataSet  eg:dataset-le3 ;
    eg:refArea                  ex-geo:cardiff_00pt ;
    eg:lifeExpectancy           83.4 ;
    .

eg:o63 a qb:Observation;
    qb:dataSet  eg:dataset-le3 ;
    eg:refArea                  ex-geo:monmouthshire_00pp ;
    eg:lifeExpectancy           81.7 ;
    .

eg:o64 a qb:Observation;
    qb:dataSet  eg:dataset-le3 ;
    eg:refArea                  ex-geo:merthyr_tdfil_00ph ;
    eg:lifeExpectancy           79.6 ;
    .
```

D. References

D.1 Normative references

[DC11]
Dublin Core metadata initiative. *Dublin Core metadata element set, version 1.1*. July 1999. Dublin Core recommendation. URL: http://dublincore.org/documents/dcmi-terms/

[OWL2-PRIMER]
Pascal Hitzler; Markus Krötzsch; Bijan Parsia; Peter Patel-Schneider; Sebastian Rudolph. *OWL 2 Web Ontology Language Primer (Second Edition)*. 11 December 2012. W3C Recommendation. URL: http://www.w3.org/TR/owl2-primer/

[RDF-CONCEPTS]
Graham Klyne; Jeremy Carroll. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. 10 February 2004. W3C Recommendation. URL: http://www.w3.org/TR/rdf-concepts/

[RDF-MT]
Patrick Hayes. *RDF Semantics*. 10 February 2004. W3C Recommendation. URL: http://www.w3.org/TR/rdf-mt/

[RDF-PRIMER]
Frank Manola; Eric Miller. *RDF Primer*. 10 February 2004. W3C Recommendation. URL: http://www.w3.org/TR/rdf-primer/

[RDF-SCHEMA]
Dan Brickley; Ramanathan Guha. *RDF Schema 1.1*. 9 January 2014. W3C Proposed Edited Recommendation. URL: http://www.w3.org/TR/rdf-schema/

[RFC2119]
S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels.* March 1997. Internet RFC 2119. URL: http://www.ietf.org/rfc/rfc2119.txt

[SKOS-REFERENCE]
Alistair Miles; Sean Bechhofer. *SKOS Simple Knowledge Organization System Reference*. 18 August 2009. W3C Recommendation. URL: http://www.w3.org/TR/skos-reference

[sparql11-query]
Steven Harris; Andy Seaborne. *SPARQL 1.1 Query Language*. 21 March 2013. W3C Recommendation. URL: http://www.w3.org/TR/sparql11-query/

[sparql11-update]
Paul Gearon; Alexandre Passant; Axel Polleres. *SPARQL 1.1 Update*. 21 March 2013. W3C Recommendation. URL: http://www.w3.org/TR/sparql11-update/

D.2 Informative references

[COG]
SDMX Contnent Oriented Guidelines, http://sdmx.org/?page_id=11

[DCAT]
Fadi Maali; John Erickson. *Data Catalog Vocabulary (DCAT)*. W3C Candidate Recommendation. 5 November 2013. URL: http://www.w3.org/TR/vocab-dcat/

[FOAF]
Dan Brickley, Libby Miller. *FOAF Vocabulary Specification 0.98.* 9 August 2010. URL: http://xmlns.com/foaf/spec/

[HAUS09]
Michael Hausenblas; Wolfgang Halb; Yves Raimond; Lee Feigenbaum; Danny Ayers. *SCOVO: Using Statistics on the Web of Data*. 2009. URL: http://mhausenblas.info/pubs/eswc09-inuse-scovo.pdf

[LOD]
Linked Data, http://linkeddata.org/

[OLAP]
Online Analytical Processing Data Cubes, http://en.wikipedia.org/wiki/OLAP_cube

[ORG]
Dave Reynolds. *The Organzation Ontology*. 25 June 2013. W3C Candidate Recommendation. URL: http://www.w3.org/TR/vocab-org/

[OS-GEO]
Ordnance Survey Administrative Geography Ontology, http://data.ordnancesurvey.co.uk/ontology/admingeo/

[SCOVO]
The Statistical Core Vocabulary, http://sw.joanneum.at/scovo/schema.html

[SDMX-GUIDE]
SDMX User Guide, Version 2009.1, January 2009. Statistical Data and Metadata Exchange Initiative. URL: http://sdmx.org/wp-content/uploads/2009/02/sdmx-userguide-version2009-1-71.pdf

[SDMX20]
SDMX Information Model: UML Conceptual Design (Version 2.0), November 2005, Statistical Data and Metadata Exchange Initiative. URL: http://sdmx.org/docs/2_0/SDMX_2_0%20SECTION_02_InformationModel.pdf

[SKOS-PRIMER]
Antoine Isaac; Ed Summers. *SKOS Simple Knowledge Organization System Primer*. 18 August 2009. W3C Note. URL: http://www.w3.org/TR/skos-primer

[turtle]
Eric Prud'hommeaux; Gavin Carothers. *RDF 1.1 Turtle*. 9 January 2014. W3C Proposed Recommendation. URL: http://www.w3.org/TR/turtle/

[void]
Keith Alexander; Richard Cyganiak; Michael Hausenblas; Jun Zhao. *Describing Linked Datasets with the VoID Vocabulary*. 3 March 2011. W3C Note. URL: http://www.w3.org/TR/void/