Validating statistical index data represented in RDF using SPARQL queries

Jose Emilio Labra Gayo
WESO Research Group
University of Oviedo
Spain labra@uniovi.es

Jose M. Álvarez Rodríguez
South East European Research Center
Greece
jmalvarez@seerc.org

1 Introduction

The creation and use of quantitative indexes is a widely accepted practice that has been applied to numerous domains like Bibliometrics (Impact factor), research and academic performance (H-Index or Shanghai rankings), cloud computing (Global Cloud Index, by CISCO), etc. We consider that those indexes and rankings could benefit from a Linked Data approach where the rankings could be seen, tracked and verified by their users.

We participated in the Web Index project (`http://thewebindex.org`), which created an index to measure the Web impact in different countries. The 2012 version offered a data portal[1] whose data was obtained by transforming the raw observations and precomputed values from Excel sheets to RDF[3]. In the 2013 version of that data portal, we are working on both validating and computing observations to automatically derivate and populate new values to automate the validation and even the generation of the index from raw data.

We have defined a generic vocabulary of computational index structures which could be applied to compute and validate any other kind of index and can be seen as an instance of the RDF Data Cube vocabulary [2]. The validation process employs SPARQL [4] queries to model the different integrity constraints and computation steps in a declarative way.

At this moment, we have a running example and a validator which reads and executes the SPARQL queries. Source code and some examples are available at `https://github.com/weso/computex`. Although our prototype validator has been implemented in Scala, our approach is independent of any programming language as far as it can load and execute SPARQL 1.1 queries.

Along the paper we will use Turtle and SPARQL notation and assume that the namespaces have been declared using the most common prefixes found in `http://prefix.cc`.

2 Example data and Index computation process

Our data model consists of a list of observations which can be raw observations obtained from an external source or computed observations derived from other observations. An example observation can be:

[1] `http://data.webfoundation.org`

```
obs:obsM23 a qb:Observation ;
 cex:computation [ a cex:Z-Score ;
     cex:observation obs:obsA23 ; cex:slice slice:sliceA09 ; ] ;
 cex:value 0.56 ;
 cex:md5-checksum "2917835203..." ;
 cex:indicator indicator:A ;
 cex:concept country:ESP ;
 qb:dataSet dataset:A-Normalized ;
 # ... other declarations omitted for brevity
```

Where we declare that `obs:obsM23` is an observation whose value is `0.56` that has been obtained as the Z-Score of the observation `obs:A23` using the slice `slice:sliceA09`. The observations refers to indicator `indicator:A`, to the concept `country:ESP` and to the dataset `dataset:A-Normalized`.

For each observation, we also add a value for `cex:md5-checksum` which is obtained as a combination of the different values of the observation and allows a user to verify the values asserted to that observation.

3 Computex vocabulary

The *Computex* vocabulary is available at `http://purl.org/weso/computex`. It defines terms related to the computation of statistical index data and is compatible with RDF Data Cube vocabulary. Some terms defined in the vocabulary are:

- **Concept** represents the entities that we are indexing. In the case of the Web Index project, the concepts are the different countries. In other applications it could be Universities, journals, services, etc.
- **Indicator**. A dimension whose values add information to the Index. Indicators can be simple dimensions, for example: the mobile phone suscriptions per 100 population, or can be composed from other indicators.
- **Observation**. This term is compatible with the RDF Data Cube `qb:Observation`. It contains values for the properties: `cex:value`, `cex:indicator` and `cex:concept`, etc. The value of the observation can be a Raw value obtained from an external source or a computed value obtained from other observations.
- **Computation**. We have declared the main computation types that we needed for the WebIndex project, which have been summarized in table 1. That list of computation types is non-exhaustive and can be further extended in the future.
- **WeightSchema** a weight schema for a list of indicators. It consists of a weight associated for each indicator which can be used to compute an aggregated observation.

4 Validation approach

The validation approach was inspired by the integrity constraint specification proposed by the RDF Data Cube vocabulary which employs a set of SPARQL `ASK` queries to check the integrity of RDF Data Cube

Table 1: Some types of statistical computations

Computation	Description	Properties
Raw	No computation. Raw value obtained from external source.	
Mean	Mean of a set of observations	`cex:observation` `cex:slice`
Increment	Increment an observation by a given amount	`cex:observation` `cex:amount`
Copy	A copy of another observation	`cex:observation`
Z-score	A normalization of an observation using the values from a Slice.	`cex:observation` `cex:slice`
Ranking	Position in the ranking of a slice of observations.	`cex:observation` `cex:slice`
AverageGrowth(N)	Expected average growth of N observations	`cex:observations` a collection of observations
WeightedMean	Weighted mean of an observation	`cex:observation` `cex:slice` `cex:weightSchema`

data. Although `ASK` queries provide a good means to check integrity, in practice their boolean nature does not offer too much help when a dataset does not accomplish with the data model.

We decided to use `CONSTRUCT` queries which, in case of error, contain an error message and a list of error parameters that can help to spot the problematic data.

We transformed the `ASK` queries defined in the RDF Data Cube specification to `CONSTRUCT` queries. For example, the query to validate the RDF Data Cube integrity constraint 4 (IC-4) is:

```
CONSTRUCT {
 [ a cex:Error ; cex:errorParam [cex:name "dim"; cex:value ?dim ] ;
   cex:msg "Every Dimension must have a declared range" . ]
} WHERE { ?dim a qb:DimensionProperty .
  FILTER NOT EXISTS { ?dim rdfs:range [] }
}
```

In order to make our error messages compatible with EARL [1], we have defined `cex:Error` as a subclass of `earl:TestResult` and declared it to have the value `earl:failed` for the property `earl:outcome`.

We have also created our own set of SPARQL `CONSTRUCT` queries to validate the *Computex* vocabulary terms, specially the computation of index data. For example, the following query validates that every observation has at most one value.

```
CONSTRUCT {
 [ a cex:Error ; cex:errorParam # ... omitted
    cex:msg "Observation has two different values" . ]
} WHERE { ?obs a qb:Observation .
 ?obs cex:value ?value1 . ?obs cex:value ?value2 .
```

```
 FILTER ( ?value1 != ?value2 )
}
```

Using this approach, it is possible to define more expressive validations. For example, we are able to validate that an observation has been obtained as the mean of other observations.

```
CONSTRUCT {
 [ a cex:Error ; cex:errorParam # ...omitted
   cex:msg "Mean value does not match" ] .
} WHERE {
   ?obs a qb:Observation ;
        cex:computation ?comp ;
        cex:value ?val .
  ?comp a cex:Mean .
  { SELECT (AVG(?value) as ?mean) ?comp WHERE {
     ?comp cex:observation ?obs1 .
        ?obs1 cex:value ?value ;
  } GROUP BY ?comp }
 FILTER( abs(?mean - ?val) > 0.0001)
}
```

5 Expressivity limits of SPARQL queries

Validating statistical computations using SPARQL queries offered a good exercise to check SPARQL expressivity. Although we were able to express most of the computation types, some of them had to employ functions that are not part of SPARQL 1.1 or had to be defined in a limited way. In this section we review some of the challenges that we found.

- The Z-score of a value x_i is defined as $\frac{x-\bar{x}}{\sigma}$ where $\bar{x}$ is the mean and $\sigma = \sqrt{\frac{\sum_{i=1}^{N}(\bar{x}-x_i)^2}{N-1}}$ is the standard deviation. To validate that computation using SPARQL queries, it is necessary to employ the `sqrt` function. This function is not available in SPARQL 1.1 although some implementations like Jena ARQ[2] provide it.

- In order to validate the ranking of an observation (in which position it appears in a list of observations), we have found two approaches. One is to check all the observations that are below the value of that observation. This approach requires to check the value of each observation against all the other values. The other approach is to use a subquery that groups all the observations ordered by their value using the `GROUP_CONCAT`. However, SPARQL does not offer a function to calculate the position of a substring in a string[3], so we divided the length of the substring before the concept's name by the length of the concept's name. This approach is more efficient but only works when all the names have the same length.

- Given a list of values $x_1, x_2 \ldots x_n$ the expected value x_{n+1} can be extrapolated using the forward average growth formula: $x_n \times \frac{\frac{x_n}{x_{n-1}}+\ldots+\frac{x_2}{x_1}}{n-1}$. Accessing RDF collections in SPARQL 1.1 requires property paths and offers limited expressivity. In this particular case the query can be expressed as[4]:

[2] `http://jena.apache.org/documentation/query/library-function.html`
[3] This function is called `strpos` in PHP or `indexOf` in Java
[4] This query has been suggested by Joshua Taylor

```
CONSTRUCT {
 # ... omitted for brevity
} WHERE {
 ?obs cex:computation [a cex:AverageGrowth; cex:observations ?ls] ;
 cex:value ?val .
 ?ls rdf:first [cex:value ?v1 ] .
 { SELECT ( SUM(?v_n / ?v_n1)/COUNT(*) as ?meanGrowth) WHERE {
    ?ls rdf:rest* [ rdf:first [ cex:value ?v_n ] ;
                rdf:rest [ rdf:first [ cex:value ?v_n1 ]]] .
 }}
 FILTER (abs(?meanGrowth * ?v1 - ?val) > 0.001) }
```

6 Conclusions

Using SPARQL queries to validate and compute index data seems a promising use case for linked data applications. Although we have succesfully employed this approach to validate most of the statistical computations we needed for the WebIndex project, we have found some limitations in current SPARQL 1.1 expressivity with regards to built-in functions on maths and strings.

Our future work is to automate the declarative computation of index data from the raw observations and to check the performance with real data like the Web Index data. We are also considering the feasibility of this approach for online calculation of index scores and rankings.

References

[1] Evaluation and report language EARL 1.0 schema. http://www.w3.org/TR/EARL10-Schema/, 2011. W3C Working Draft.

[2] The RDF data cube vocabulary. http://www.w3.org/TR/vocab-data-cube/, 2013. W3c Candidate Recommendation.

[3] J. M. Alvarez Rodríguez, J. Clement, J. E. Labra Gayo, H. Farhan, and P. Ordoñez. *Cases on Open-Linked Data and Semantic Web Applications*, chapter Publishing Statistical Data following the Linked Open Data Principles: The Web Index Project., pages 199–226. IGI Global, 2013. doi:10.4018/978-1-4666-2827-4.ch011.

[4] S. Harris and A. Seaborne. SPARQL 1.1 query language. http://www.w3.org/TR/sparql11-query/, 2013.