

Create Integrity Constraints SPARQL Queries from RDF data cube definition

mja@statgroup.dk

2015-01-11

Contents

```
library(RCurl)
```

```
## Loading required package: methods
## Loading required package: bitops
```

```
library(XML)
library(devtools)
```

```
qbURL<-"http://www.w3.org/TR/2014/REC-vocab-data-cube-20140116/"
if (! url.exists(qbURL) ) {
  stop(paste0("Can not access URL ",qbURL))
}
```

```
# Acknowledgement: I got the approach from
# http://stackoverflow.com/questions/1395528/scraping-html-tables-into-r-data-frames-using-the-xml-pack
```

```
webpage <- getURL(qbURL)
# The following two lines is suggested in the stackoverflow post
# Apparantly not needed here
## Process escape characters
## webpage <- readLines(tc <- textConnection(webpage)); close(tc)
```

```
# Parse the html tree, ignoring errors on the page
pagetree <- htmlTreeParse(webpage, error=function(...) {}, useInternalNodes = TRUE)
```

```
# appears that integrity checks starte with h3 and then a table with class bordered-table
# so that's what we look for
both<- getNodeSet(pagetree,"//*[h3[@id] | /*/table[@class='bordered-table']/tbody/tr/td/pre")
```

```
qbIClist<- list()
```

```
for (i in 1:(length(both)-1)) {
  icname<- xmlGetAttr(both[[i]],"id",default="none")
  if (grepl('ic-[1-9]([0-9])*', icname) ) {
    ictitle<- xmlValue(xmlChildren(both[[i]])$text )
    # print(paste0("Node ", i, ", IC name ", icname, " - ", ictitle ))
    qbIClist[[icname]]<- paste0(
      paste0("# ", ictitle, "\n" ),
      xmlValue(xmlChildren(both[[i+1]])$text),
      sep="\n"
```

```
)  
}  
}
```

Here are the integrity constraints:

```
for (icname in names(qbIClist)) {  
##   fileConn<-file(paste0(icname, ".rq"))  
##   cat( qbIClist[[icname]] )  
##   close(fileConn)  
}
```

```
## # IC-1. Unique 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  
## 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  
## ASK {  
##   ?dsd a qb:DataStructureDefinition .  
##   FILTER NOT EXISTS { ?dsd qb:component [qb:componentProperty [a qb:MeasureProperty]] }  
## }  
##  
## # IC-4. Dimensions have range  
## ASK {  
##   ?dim a qb:DimensionProperty .  
##   FILTER NOT EXISTS { ?dim rdfs:range [] }  
## }  
##  
## # IC-5. Concept dimensions have code lists
```

```

## ASK {
##   ?dim a qb:DimensionProperty ;
##       rdfs:range skos:Concept .
##   FILTER NOT EXISTS { ?dim qb:codeList [] }
## }
##
## # IC-6. Only attributes may be optional
## 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
## ASK {
##   ?sliceKey a qb:SliceKey .
##   FILTER NOT EXISTS { [a qb:DataStructureDefinition] qb:sliceKey ?sliceKey }
## }
##
## # IC-8. Slice Keys consistent with DSD
## 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
## 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
## ASK {
##   ?slice qb:sliceStructure [qb:componentProperty ?dim] .
##   FILTER NOT EXISTS { ?slice ?dim [] }
## }
##
## # IC-11. All dimensions required
## 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
## 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
## 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
## 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
## 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
## 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
## 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
## ASK {
##   ?dataset qb:slice      ?slice .
##   ?slice   qb:observation ?obs .
##   FILTER NOT EXISTS { ?obs qb:dataSet ?dataset . }
## }
##
## # IC-19. Codes from code list
## 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
## 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/<$p>* ?v }
## }
##
## # IC-21. Codes from hierarchy (inverse)
## ASK {
##      ?obs qb:dataSet/qb:structure/qb:component/qb:componentProperty ?dim .
##      ?dim a qb:DimensionProperty ;
##      qb:codeList ?list .
##      ?list a qb:HierarchicalCodeList .
##      ?obs ?dim ?v .
##      FILTER NOT EXISTS { ?list qb:hierarchyRoot/(~<$p>)* ?v }
## }
##

```

The script is intended to run from the package root as

```
knit("inst/data-raw/create-qb-IC-dataset.Rmd")
```

knit runs the script in the data-raw directory, so it would be expected to use `pkg=“../..”` to store the qbIClist in the data directory. However, it did not work - hence the `setwd` below-

Data are stored in the data directory, following [R packages by Hadley Wickham](#) and [Writing R Extensions](#).

```
devtools::use_data(qbIClist, pkg=“../..”, overwrite=TRUE)
```

```

# This stores the qbIClist in the data directory
# Consider making it internal

setwd("../..")
devtools::use_data(qbIClist, overwrite=TRUE)

```

```
## Saving qbIClist to data/qbIClist.rda
```

```
print("Done")
```

```
[1] "Done"
```