Derive results in RDF data cube and compare with results in data cube

mja@statgroup.dk

2015-01-11

Contents

For developing

Use

```
library(knitr)
knit("vignettes/check-cube.Rmd")
```

The vignette

```
options(width=200) # long lines

library(xlsx)
```

```
## Loading required package: rJava
## Loading required package: methods
## Loading required package: xlsxjars
```

```
library(foreign)
library(RCurl)
```

```
## Loading required package: bitops
##
## Attaching package: 'RCurl'
##
## The following object is masked from 'package:rJava':
##
##     clone
```

```
library(rrdf)
```

```
## Loading required package: rrdflibs
```

```
library(rrdfqbcrnd0)
```

```
## Warning: replacing previous import by 'rJava::clone' when loading 'rrdfqbcrnd0'
```

```
debug.add.data.triple<- function( cubeData,
subject,
predicate,
data, type) {

print(cbind(subject=subject,predicate=predicate,data=data,type=type))

add.data.triple( cubeData,
subject=subject,
predicate=predicate,
data=data, type=type)

}

check.cube<- function(
  obsFile=paste(tempdir(),"/", "adsl", ".xpt",sep=""),
  dsURL= "https://phuse-scripts.googlecode.com/svn/trunk/scriptathon2014/data/adsl.xpt",
  ds.dataset= "ds:dataset-demog",
  qbfiledir= system.file("extdata/sample-rdf", package="rrdfqbcrnd0"),
  qbfile= system.file("extdata/sample-rdf", "DC-DM-sample.TTL", package="rrdfqbcrnd0"),
  domainName="demog",
  RDFCubeWorkbook = system.file("extdata/sample-cfg", "RDFCubeWorkbook.xlsx", package="rrdfqbcrnd0")
) {

 ## obsFile=paste(tempdir(),"/", "adsl", ".xpt",sep="")
 ## dsURL= NULL
 ## ds.dataset= "ds:dataset-demog"
 ## qbfiledir= system.file("extdata/sample-rdf", package="rrdfqbcrnd0")
 ## qbfile= system.file("extdata/sample-rdf", "DC-DM-sample.TTL", package="rrdfqbcrnd0")
 ## domainName="demog"
 ## RDFCubeWorkbook = NULL

if (!is.null(RDFCubeWorkbook)) {
  common.prefixes <- read.xlsx(RDFCubeWorkbook,sheetName=paste0("CubePrefixes"))
  cubeMetadata <- read.xlsx(RDFCubeWorkbook,sheetName=paste0(domainName,"-Components"))
  metadataSource <-cubeMetadata[grep("metadata", cubeMetadata$compType),]
  if (is.null(obsFile)) {
    obsFile<- metadataSource[ metadataSource$compName=="wasDerivedFrom", "compLabel" ]
  }
  if (is.null(qbfile)) {
    qbfile<-  file.path(qbfiledir,metadataSource[ metadataSource$compName=="dataCubeFileName", "dataCub
  }
  if (is.null(dsURL)) {
    dsURL<-  metadataSource[ metadataSource$compName=="obsURL", "dataCubeFileName" ]
  }
} else {
  common.prefixes <- data.frame(
    prefix=gsub("^prefix","",names(Get.default.crnd.prefixes())),
    namespace=as.character(Get.default.crnd.prefixes() )
    )
}
```

```
if (! is.null(dsURL) ) {
   if (! url.exists(dsURL) ) {
      stop(paste0("Can not access URL ",dsURL))
   }
   download.file( dsURL, obsFile, method="curl")
}

if (! file.exists(obsFile)) {
  stop(paste0("Expected file ", obsFile, " does not exist"))
  }
DataSet<-read.xport(obsFile)

## library(Hmisc)
## DataSet <- sasxport.get(fnDataSet)
## # --- see what have got:
## contents(DataSet)
## label(DataSet)

# str(DataSet)



# Domain-specific prefixes:for /code/, /prop/, /dccs/ and /dataset/
custom.prefixes <-Get.qb.crnd.prefixes(domainName)

# Prefix for storing the results of check each measure in the data cube

  validation.mesure.prefix<- data.frame(prefix=c("validmeas"),
        namespace=c(paste0("http://www.example.org/dc/",domainName,"/validmeas/")
                            ))

prefixes<- rbind(common.prefixes, custom.prefixes, validation.mesure.prefix)

forsparqlprefix<- paste("prefix", paste(prefixes$prefix,":",sep=""), paste("<",prefixes$namespace,">",s

# cat(forsparqlprefix)
# The qbfile also contains prefixes, which are part of the model
# So not adding the prefixes to the model, but using them for adding further
# information to the model when deriving statistics

cubeData = new.rdf(ontology=FALSE)
myprefixes<- qb.def.prefixlist(cubeData, prefixes )
load.rdf( qbfile, format="N3", appendTo= cubeData)
# summarize.rdf(cubeData)


## ------------------------------------------------------------------------
cube.observations1<- sparql.rdf(cubeData,
   paste( forsparqlprefix,
     "select * where {?s ?p ?o .} limit 10"
     )
   );
# print(cube.observations1)
```

```
cube.observations2<- sparql.rdf(cubeData,
   paste( forsparqlprefix,
     "select * where { ?s a qb:Observation ; ?p ?o .} limit 10"
     )
   );
# print(cube.observations2)

## get the dimensions
cube.dimensions<- sparql.rdf(cubeData,
  paste(forsparqlprefix,
"select * where { [] qb:dimension ?p .  }"
));
# print(cube.dimensions)

## get the codelist
codelists.rq<-   paste(forsparqlprefix,
'
select ?p ?cl ?prefLabel where {
[] qb:dimension ?p.
?p qb:codeList ?c.
?c skos:hasTopConcept ?cl .
?cl skos:prefLabel ?prefLabel.
}
'
)

# cat(codelists.rq)

cube.codelists<- as.data.frame(sparql.rdf(cubeData, codelists.rq));
# print(cube.codelists)


codelist.all<- cube.codelists[ cube.codelists$prefLabel=="_ALL_",]
# print(codelist.all)
subsetting.dimensions<- list();

# the variable name/column name in the data frame should be part of the datacube
# this would remove the need for the workaround below using gsub
for (i in 1:nrow(codelist.all))
{
  subsetting.dimensions[[ gsub("[^:]*:","", codelist.all[i,"p"]) ]] <-
      as.character(codelist.all[i,"cl"])
}


## get the dimensions and attributes

cube.dimensionsattr<- sparql.rdf(cubeData,
  paste(forsparqlprefix,
"select * where { {[] qb:dimension ?p . } union {  ?p a qb:AttributeProperty . } }"
));
# print(cube.dimensionsattr)
```

```
selectexpr<-  paste(  "select * where {",
    "    ?s a qb:Observation  ;",
    paste("       qb:dataSet",  ds.dataset, " ;", sep=" ", collapse="\n"), "\n",
    paste0( cube.dimensionsattr, " ", sub("prop:", "?", cube.dimensionsattr), ";", collapse="\n"),
                  "\n",
    "       prop:measure      ?measure ;      \n",
    "       prop:denominator      ?denominator .     \n",
    paste0( "optional{ ", sub("prop:", "?", cube.dimensionsattr), " ",
           "skos:prefLabel",
           " ",
           sub("prop:", "?", cube.dimensionsattr), "value" ,
           " . ", "}",
           collapse="\n"),
    "} ",
    "\n"
   );
# cat(selectexpr)
# cat(  paste(forsparqlprefix, "\n", selectexpr ) )

cube.observations<- sparql.rdf(cubeData,
  paste(forsparqlprefix, "\n",  selectexpr )
  );

# print(head(cube.observations))
# str(cube.observations)


# cts:cdiscSubmissionValue could also be used instead of skos:prefLabel
# by code:saffl-Y does not have cts:cdiscSubmissionValue

# using list for key-value lookup to function for descriptive statistic
univfunc1= list(
  "code:procedure-MEAN"=mean,
  "code:procedure-STDDEV"=sd,
  "code:procedure-MEDIAN"=median,
  "code:procedure-MIN"=min,
  "code:procedure-MAX"=max
  )

univfunc2= list(
  "code:procedure-COUNT"=length,
  "code:procedure-COUNTDISTINCT"=function(x){length(unique(x))}
  )


for (r in  1:nrow(cube.observations )  ) {
thisrow<-  cube.observations[r,]
# print(thisrow)

data.subset.logical= rep(TRUE, nrow(DataSet))
for (v in names(subsetting.dimensions)) {
#   print( c(v, thisrow[v ],  subsetting.dimensions[[ v ]] ) )
  if ( thisrow[v ] != subsetting.dimensions[[ v ]] ) {
```

```
#   print( paste0(v, ": ", thisrow[v ], " = ", thisrow[ paste0(v,"value") ]))
#   print( DataSet[,toupper(v)] == thisrow[ paste0(v,"value") ] )
    data.subset.logical= data.subset.logical & ( DataSet[,toupper(v)] == thisrow[ paste0(v,"value") ] )
  }
}


has.result= FALSE

if (thisrow["procedure"] %in% names(univfunc1) ) {
# print("univfunc1")
  AOD= DataSet[data.subset.logical,thisrow["factorvalue"]]
  result= univfunc1[[ thisrow["procedure"] ]](AOD)
  has.result= TRUE
#   print(paste("AOD number of observations", nrow(AOD)))
} else if (thisrow["procedure"] %in% names(univfunc2) ) {
# print(paste0("univfunc2"," ",thisrow["procedure" ],collapse=" "))
  AOD= DataSet[data.subset.logical,"USUBJID"]  # take the first variable, would be better if there was a
  result= univfunc2[[ thisrow["procedure" ] ]](AOD)
  has.result= TRUE
#   print(paste0("AOD number of observations", nrow(AOD), sep=" "))
} else if (thisrow["procedure"]== "code:procedure-PERCENT" & thisrow["factor"]== "code:factor-PROPORTIO

#  print(c("proportion",thisrow["denominator"]))

#  denom.def<- thisrow[ c("saffl", "trt01a", "race", "sex" ) ]
#  denom.def<- thisrow[ c( "trt01a", "race", "sex" ) ]
  denom.def<- thisrow[ names(subsetting.dimensions) ]

  denom.def[ tolower(thisrow["denominator"]) ] =paste0("code:",tolower(thisrow["denominator"]),"-_ALL_"
  AOD= DataSet[data.subset.logical,] # should use a variable name - USUBJID like for count

denom.data.subset.logical= rep(TRUE, nrow(DataSet))
# print( denom.def )
for (v in names(denom.def)) {
  if ( denom.def[v ] != subsetting.dimensions[[ v ]] ) {
    denom.data.subset.logical= denom.data.subset.logical & ( DataSet[,toupper(v)] == thisrow[ paste0(v,"
  }
}
  denom.data.frame<-  DataSet[denom.data.subset.logical , ]

  result= nrow(AOD) / nrow( denom.data.frame ) * 100;
  has.result=TRUE
  # print(paste("AOD number of observations", nrow(AOD)))
  # print(paste("denom.data.frame number of observations", nrow(denom.data.frame)))
}



if (has.result) {
# print(paste("result", result, sep=" "))
add.data.triple( cubeData,
subject=gsub("ds:", myprefixes$prefixDS, thisrow["s"]), # add triple does not resolve prefix
```

```
predicate=gsub("validmeas:", myprefixes$prefixVALIDMEAS, "validmeas:result"),
data=paste(result), type="float")
} else {
#  print( paste( thisrow["s"], ifelse(has.result, result, "No result determined") ) )
}


} # for

cube.measure.result<-  sparql.rdf(cubeData,
  paste(forsparqlprefix,
    "select * where {",
    "    ?s a qb:Observation  ;",
    paste("       qb:dataSet",  ds.dataset, " ;", sep=" ", collapse=" "),
    "       prop:measure      ?measure ;      ",
    "      optional{ ?s validmeas:result      ?result }      ",
    "} order by ?s"
                )
  );

# print(cube.measure.result)


cube.check<- sparql.rdf(cubeData,
  paste(forsparqlprefix,
    "select * where {",
    "    ?s a qb:Observation  ;",
    paste("       qb:dataSet",  ds.dataset, " ;", sep=" ", collapse=" "),
    "       prop:measure      ?measure ;      ",
    "      optional{ ?s validmeas:result      ?result }      ",
    " filter ( ?measure != ?result ) ",
    "} order by ?s"
                )
  );

print("If the result is <0 x 0> matrix then all value matches")
cube.check


}



check.cube(
   obsFile=paste(tempdir(),"/", "adsl", ".xpt",sep=""),
   dsURL= "https://phuse-scripts.googlecode.com/svn/trunk/scriptathon2014/data/adsl.xpt",
   ds.dataset= "ds:dataset-demog",
 qbfile= system.file("extdata/sample-rdf", "DC-DM-sample.TTL", package="rrdfqbcrnd0"),
 domain="demog",
 RDFCubeWorkbook=NULL
 )
```

[1] “If the result is <0 x 0> matrix then all value matches” <0 x 0 matrix>

```
check.cube(
obsFile=paste(tempdir(),"/", "adae", ".xpt",sep=""),
dsURL= "https://phuse-scripts.googlecode.com/svn/trunk/scriptathon2014/data/adae.xpt",
ds.dataset= "ds:dataset-ae",
qbfile= system.file("extdata/sample-rdf", "DC-AE-sample.TTL", package="rrdfqbcrnd0"),
domain="ae",
RDFCubeWorkbook=NULL
)
```

[1] "If the result is <0 x 0> matrix then all value matches" <0 x 0 matrix>

```
check.cube(
  obsFile=paste(tempdir(),"/", "adsl", ".xpt",sep=""),
  dsURL= NULL,
  ds.dataset= "ds:dataset-demog",
  qbfile= system.file("extdata/sample-rdf", "DC-DM-sample.TTL", package="rrdfqbcrnd0"),
  domain="demog",
  RDFCubeWorkbook=NULL
 )
```

[1] "If the result is <0 x 0> matrix then all value matches" <0 x 0 matrix>

```
check.cube(
 obsFile=paste(tempdir(),"/", "adae", ".xpt",sep=""),
 dsURL=  NULL,
 ds.dataset= "ds:dataset-ae",
 qbfile= system.file("extdata/sample-rdf", "DC-AE-sample.TTL", package="rrdfqbcrnd0"),
 domain="ae",
 RDFCubeWorkbook=NULL
 )
```

[1] "If the result is <0 x 0> matrix then all value matches" <0 x 0 matrix>

End of file.