RRDF gotcha

mja@statgroup.dk

2016-03-08

Contents

Setup **1**

Getting information out from a store **1**

`sparql.rdf` return result is a character matrix **2**

Same triple repeated: difference between Apache/Jena Store and SPARQL insert using Apache/Jena **3**

Setup

First load the package.

```
library(rrdf)
library(rrdfancillary)
```

Getting information out from a store

```
storeex<- new.rdf(ontology=FALSE)
query.rq<- '
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>
PREFIX ex:   <http://example.org/>
SELECT ?s (isIRI(?s) as ?isiri_s) ?p (isIRI(?p) as ?isiri_p)  ?o (isIRI(?o) as ?isiri_o)  (lang(?o) as 
where {
values (?s ?p ?o) {
(ex:a ex:p 1)
(ex:a ex:p 1.0)
(ex:a ex:p "1.00"^^xsd:float)
(ex:a ex:p "1.000"^^xsd:double)
(ex:a ex:p "NaN"^^xsd:float)
(ex:a ex:p "NaN"^^xsd:double)
(ex:b ex:p "a")
(ex:b ex:p "a"@en)
(ex:b ex:p "a"^^xsd:string)
("a" "b" "c")
}
}
'
```

```
query.res<- sparql.rdf( storeex, query.rq )
knitr::kable(query.res)
```

s	isiri_s	p	isiri_p	o	isiri_o	olang	idatatype
ex:a	true	ex:p	true	1	false		xsd:integer
ex:a	true	ex:p	true	1.0	false		xsd:decimal
ex:a	true	ex:p	true	1.00	false		xsd:float
ex:a	true	ex:p	true	1.000	false		xsd:double
ex:a	true	ex:p	true	NaN	false		xsd:float
ex:a	true	ex:p	true	NaN	false		xsd:double
ex:b	true	ex:p	true	a	false		xsd:string
ex:b	true	ex:p	true	a	false	en	http://www.w3.org/1999/02/22-rdf-syntax-ns#langString
ex:b	true	ex:p	true	a	false		xsd:string
a	false	b	false	c	false		xsd:string

`sparql.rdf` return result is a character matrix

```
storeex2<- new.rdf(ontology=FALSE)
query2.rq<- '
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>
PREFIX ex:   <http://example.org/>
SELECT ?s ?p  ?o  (datatype(?o) as ?idatatype)
where {
values (?s ?p ?o) {
(ex:a ex:p 1)
(ex:a ex:p 1.0)
(ex:a ex:p 1.0E0)
(ex:a ex:p "NaN"^^xsd:double )
}
}
'

query2.res<- sparql.rdf( storeex2, query2.rq )
knitr::kable(query2.res)
```

s	p	o	idatatype
ex:a	ex:p	1	xsd:integer
ex:a	ex:p	1.0	xsd:decimal
ex:a	ex:p	1.0E0	xsd:double
ex:a	ex:p	NaN	xsd:double

```
query2.df<- as.data.frame(query2.res, stringsAsFactors=FALSE)
knitr::kable(query2.df)
```

s	p	o	idatatype
ex:a	ex:p	1	xsd:integer

s	p	o	idatatype
ex:a	ex:p	1.0	xsd:decimal
ex:a	ex:p	1.0E0	xsd:double
ex:a	ex:p	NaN	xsd:double

```
str(query2.df)
```

```
## 'data.frame':    4 obs. of  4 variables:
##  $ s        : chr  "ex:a" "ex:a" "ex:a" "ex:a"
##  $ p        : chr  "ex:p" "ex:p" "ex:p" "ex:p"
##  $ o        : chr  "1" "1.0" "1.0E0" "NaN"
##  $ idatatype: chr  "xsd:integer" "xsd:decimal" "xsd:double" "xsd:double"
```

So, if the numeric columns are needed, they must explicitely be made by coversion. Below a new column `on` contains the numeric representation of the `o` column.

```
query21.df<- query2.df
query21.df$on <- as.numeric(query21.df$o)
knitr::kable(query21.df)
```

s	p	o	idatatype	on
ex:a	ex:p	1	xsd:integer	1
ex:a	ex:p	1.0	xsd:decimal	1
ex:a	ex:p	1.0E0	xsd:double	1
ex:a	ex:p	NaN	xsd:double	NaN

```
Map(mode,query21.df)
```

```
## $s
## [1] "character"
##
## $p
## [1] "character"
##
## $o
## [1] "character"
##
## $idatatype
## [1] "character"
##
## $on
## [1] "numeric"
```

Same triple repeated: difference between Apache/Jena Store and SPARQL insert using Apache/Jena

When the exactly same triples are inserted - only one triple remains.

```
store1<- new.rdf(ontology=FALSE)

sparql.rdf( store1, "select ?s ?p ?o (lang(?o) as ?lang) (datatype(?o) as ?datatype) where {?s ?p ?o  }
```

```
## <0 x 0 matrix>
```

```
SPARQLinsert<- '
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>
INSERT DATA
{
  <http://example.org/subject1>  <http://example.org/property1> "mytext"^^xsd:string .
  <http://example.org/subject1>  <http://example.org/property1> "mytext"^^xsd:string .
}
'
cat(SPARQLinsert,"\n")
```

```
##
## PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>
## INSERT DATA
## {
##   <http://example.org/subject1>  <http://example.org/property1> "mytext"^^xsd:string .
##   <http://example.org/subject1>  <http://example.org/property1> "mytext"^^xsd:string .
## }
##
```

```
update.rdf( store1, SPARQLinsert )
```

```
## [1] TRUE
```

```
sparql.rdf( store1, "select ?s ?p ?o (lang(?o) as ?lang) (datatype(?o) as ?datatype) where {?s ?p ?o  }
```

```
##      s                                 p                                 o
## [1,] "http://example.org/subject1" "http://example.org/property1" "mytext"
##      lang datatype
## [1,] ""   "http://www.w3.org/2001/XMLSchema#string"
```

Now, of course, it is always better only to store the values once, when it is intended to store one copy.

However, as I thougth that only one trippel is stored, so I was less carefull in some of the code.

Here is what Apache/Jena does when using the RRDF interface.

```
store2<- new.rdf(ontology=FALSE)
add.data.triple(
    store2,
    subject="http://example.org/subject1",
    predicate="http://example.org/property1",
    data="mytext",
    lang="en"
)
```

```
add.data.triple(
    store2,
    subject="http://example.org/subject1",
    predicate="http://example.org/property1",
    data="mytext",
    type="string"
)
```

Now query the store:

```
sparql.rdf( store2, "select ?s ?p ?o where {?s ?p ?o}"  )
```

```
##      s                             p                             o
## [1,] "http://example.org/subject1" "http://example.org/property1" "mytext"
## [2,] "http://example.org/subject1" "http://example.org/property1" "mytext"
```

The two rows look identical. The next query also show language and datatype associate with the object.

```
sparql.rdf( store2, "select ?s ?p ?o (lang(?o) as ?lang) (datatype(?o) as ?datatype) where {?s ?p ?o  }
```

```
##      s                             p                             o
## [1,] "http://example.org/subject1" "http://example.org/property1" "mytext"
## [2,] "http://example.org/subject1" "http://example.org/property1" "mytext"
##      lang datatype
## [1,] ""   "http://www.w3.org/2001/XMLSchema#string"
## [2,] "en" "http://www.w3.org/1999/02/22-rdf-syntax-ns#langString"
```

The same tripel appears twice! That learned me that the language and data type are important. They make a difference, so to speak.

Now using the same datatype, `string`, gives two triples again.

```
store3<- new.rdf(ontology=FALSE)
add.data.triple(
    store3,
    subject="http://example.org/subject1",
    predicate="http://example.org/property1",
    data="mytext",
    type="string"
)

add.data.triple(
    store3,
    subject="http://example.org/subject1",
    predicate="http://example.org/property1",
    data="mytext",
    type="string"
)

sparql.rdf( store3, "select ?s ?p ?o where {?s ?p ?o}"  )
```

```
##      s                              p                              o
## [1,] "http://example.org/subject1" "http://example.org/property1" "mytext"
## [2,] "http://example.org/subject1" "http://example.org/property1" "mytext"
```

```
sparql.rdf( store3, "select ?s ?p ?o (lang(?o) as ?lang) (datatype(?o) as ?datatype) where {?s ?p ?o  }
```

```
##      s                              p                              o
## [1,] "http://example.org/subject1" "http://example.org/property1" "mytext"
## [2,] "http://example.org/subject1" "http://example.org/property1" "mytext"
##      lang datatype
## [1,] ""   "http://www.w3.org/2001/XMLSchema#string"
## [2,] ""   "http://www.w3.org/2001/XMLSchema#string"
```

Mixing `INSERT DATA` and RRDF `add.data.triple gives` same result - two triples.

```
store4<- new.rdf(ontology=FALSE)

SPARQLinsert<- '
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>
INSERT DATA
{
  <http://example.org/subject1>  <http://example.org/property1> "mytext"^^xsd:string .
}
'
update.rdf( store4, SPARQLinsert )
```

```
## [1] TRUE
```

```
sparql.rdf( store4, "select ?s ?p ?o (lang(?o) as ?lang) (datatype(?o) as ?datatype) where {?s ?p ?o  }
```

```
##      s                              p                              o
## [1,] "http://example.org/subject1" "http://example.org/property1" "mytext"
##      lang datatype
## [1,] ""   "http://www.w3.org/2001/XMLSchema#string"
```

One triple inserted, one triple in the store. Fine!

Now add one triple - exactly the same as the previos.

```
add.data.triple(
    store4,
    subject="http://example.org/subject1",
    predicate="http://example.org/property1",
    data="mytext",
    type="string"
)

sparql.rdf( store4, "select ?s ?p ?o (lang(?o) as ?lang) (datatype(?o) as ?datatype) where {?s ?p ?o  }
```

```
##      s                              p                              o
## [1,] "http://example.org/subject1" "http://example.org/property1" "mytext"
```

```
## [2,] "http://example.org/subject1" "http://example.org/property1" "mytext"
##      lang datatype
## [1,] ""   "http://www.w3.org/2001/XMLSchema#string"
## [2,] ""   "http://www.w3.org/2001/XMLSchema#string"
```

Two triples in the store.

What if doing two `INSERT DATA`?

```
store5<- new.rdf(ontology=FALSE)

sparql.rdf( store5, "select ?s ?p ?o (lang(?o) as ?lang) (datatype(?o) as ?datatype) where {?s ?p ?o  }
```

```
## <0 x 0 matrix>
```

```
SPARQLinsert<- '
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>
INSERT DATA
{
  <http://example.org/subject1>  <http://example.org/property1> "mytext"^^xsd:string .
}
'
update.rdf( store5, SPARQLinsert )
```

```
## [1] TRUE
```

```
sparql.rdf( store5, "select ?s ?p ?o (lang(?o) as ?lang) (datatype(?o) as ?datatype) where {?s ?p ?o  }
```

```
##      s                              p                              o
## [1,] "http://example.org/subject1" "http://example.org/property1" "mytext"
##      lang datatype
## [1,] ""   "http://www.w3.org/2001/XMLSchema#string"
```

One triple - as expected, only triple was inserted.

```
update.rdf( store5, SPARQLinsert )
```

```
## [1] TRUE
```

```
sparql.rdf( store5, "select ?s ?p ?o (lang(?o) as ?lang) (datatype(?o) as ?datatype) where {?s ?p ?o  }
```

```
##      s                              p                              o
## [1,] "http://example.org/subject1" "http://example.org/property1" "mytext"
##      lang datatype
## [1,] ""   "http://www.w3.org/2001/XMLSchema#string"
```

One triple - as expected, as the triple already existed.

Lessons learned:

- Apache/Jena interface R to Java behaves differently than Apache Jena handling of Update Scripts.
- Be carefull when changing code.