Semantic Web: Assignment 1

Report

Part I

1. The namespace used for the individuals are:
   - rdf for the prefix name
   - http://www.w3.org/1999/02/22-rdf-syntax-ns# for the URI

2. The namespaces used by the schema of the ontology are:
   - rdfs for the prefix name
   - http://www.w3.org/2000/01/rdf-schema# for the URI

3. According to the Turtle syntax, John:
   - is a person **aged 37**
   - is **Jennifer's husband**
   - is **Mark's father**
   - is **Sophie's and Harry's son**
   - is **Alice's friend**
   - has **12 as shirt size**
   - has **14 as shoes size**
   - has **44 as trousers size**

Part II

1. Both files are loaded in the Corese interface.

2. The query returns the tuples (value, type). With this query, we get **69 answers**. Therefore, John has **three types: person, animal, male**.

3. This query returns the tuples of two persons who are in relationship. With this query, we get **six answers, so there are six couples**.

4. The RDF property used for indicating the shoe size of the people is **"shoesize"**.

5. The query is:

```
PREFIX h: <http://www.inria.fr/2007/09/11/humans.rdfs#>
SELECT ?person ?shoesize
WHERE
{
?person rdf:type h:Person
?person h:shoesize ?shoesize
}
```

We get **7 answers** for this query.

6. To check if the information is available, we add an OPTIONAL, like in the following query:

```
PREFIX h: <http://www.inria.fr/2007/09/11/humans.rdfs#>
SELECT ?person ?shoesize
WHERE
{
?person rdf:type h:Person
?person h:shoesize ?shoesize
OPTIONAL { ?person h:shoesize ?shoesize }
}
```

We get the same number of answers as before, so **7 answers**.

7. The query is given by:

```
PREFIX h: <http://www.inria.fr/2007/09/11/humans.rdfs#>
SELECT ?person ?shoesize
WHERE
{
?person rdf:type h:Person
?person h:shoesize ?shoesize
OPTIONAL { ?person h:shoesize ?shoesize }
FILTER (xsd:integer(?shoesize) > 8)
}
```

8. For this query, we use the UNION:

```
PREFIX h: <http://www.inria.fr/2007/09/11/humans.rdfs#>
SELECT DISTINCT ?person
WHERE
{
?person rdf:type h:Person
 {?person h:shoesize ?shoesize
  FILTER(xsd:integer(?shoesize) > 8)}
UNION
 {?person h:shirtsize ?shirtsize
  FILTER(xsd:integer(?shirtsize) > 12)}
}
```

9. Not succeeded.

10. The first query is given by:

```
PREFIX h: <http://www.inria.fr/2007/09/11/humans.rdfs#>
SELECT ?x
WHERE
{
      {
?x h:hasChild ?y
}
UNION
      {
?y h:hasParent ?x
}
UNION
      {
?y h:hasMother ?x
}
UNION
      {
?y h:hasFather ?x
}
}
```

We get 12 answers with 4 duplicates: Gaston, Laura, John and Catherine. To remove the duplicates, we execute the same query with a "DISTINCT":

```
PREFIX h: <http://www.inria.fr/2007/09/11/humans.rdfs#>
SELECT  DISTINCT ?x
WHERE
{
      {
?x h:hasChild ?y
}
UNION
      {
?y h:hasParent ?x
}
UNION
      {
?y h:hasMother ?x
}
UNION
      {
?y h:hasFather ?x
}
}
```

With this query, we get exactly **8 answers**, which shows that the 4 duplicates have been removed.

11. We use bound in order to show all the men who do not have any children:

```
PREFIX h: <http://www.inria.fr/2007/09/11/humans.rdfs#>
SELECT ?man
WHERE {
?man rdf:type h:Man
OPTIONAL {?man h:hasChild ?w }
OPTIONAL {?x h:hasFather ?man}
OPTIONAL {?y h:hasParent ?man}
OPTIONAL {?z h:hasMother ?man}
FILTER ( ! bound(?w))
FILTER ( ! bound(?x))
```

```
FILTER ( ! bound(?y))
FILTER ( ! bound(?z))
}
```

12. We use again a filter and the integer function in order to show all the people who are more than 100 years old:

```
PREFIX h: <http://www.inria.fr/2007/09/11/humans.rdfs#>
SELECT DISTINCT ?person
WHERE
{
?person rdf:type h:Person
?person h:age ?age
FILTER(xsd:integer(?age) > 100)
}
```

13. The query is given by:

```
PREFIX h: <http://www.inria.fr/2007/09/11/humans.rdfs#>
SELECT DISTINCT *
WHERE
{
?x rdf:type h:Person
?y rdf:type h:Person
?x h:shirtsize ?shirtsizex
?y h:shirtsize ?shirtsizey
FILTER(xsd:integer(?shirtsizex) = xsd:integer(?shirtsizey)
      && ?x != ?y)
}
```

14. For this query, we select all the people **minus** the people who belong to the class Man:

```
PREFIX h: <http://www.inria.fr/2007/09/11/humans.rdfs#>
SELECT DISTINCT ?person
WHERE
{
?person rdf:type h:Person
?person rdf:type ?class
FILTER( xsd:string(?class) != "http://www.inria.fr/2007/09/11/humans.rdfs#Man")
MINUS {
SELECT DISTINCT ?person
WHERE {
?person rdf:type h:Man
?person rdf:type ?class
FILTER( xsd:string(?class) = "http://www.inria.fr/2007/09/11/humans.rdfs#Man")
}
}
}
```

Part III

1. Done.

2. The following queries gives all the classes defined in the ontology:

```
SELECT DISTINCT ?class
WHERE {
?class a rdfs:Class
}
ORDER BY ?class
```

There are 8 classes: Animal, Male, Female, Man, Person, Lecturer, Researcher and Woman.

3. The following query gives all subClassOf relationships:

```
SELECT ?class ?subClass
WHERE {
?subClass rdfs:subClassOf ?class
?class a rdfs:Class
}
ORDER BY ?class
```

4. The following query provides the definitions and translations of “shoe size”:

```
SELECT ?definitions
WHERE
{
?property rdfs:label "shoe size"@en
?property rdfs:comment ?definitions
}
```

5. The following query provides all synonyms of the French term “personne:

```
SELECT ?synonyms
WHERE
{
?person rdfs:label "person"@en
?person rdfs:label ?synonyms
FILTER ( ?synonyms != "personne"@fr
            && lang(?synonyms) = 'fr' )
}
```

Part IV

1. The three following queries count the number of classes, object properties and datatype properties contained in the dbpedia ontology:

```
SELECT COUNT DISTINCT ?class
WHERE
{
?class a owl:Class
}

SELECT COUNT DISTINCT ?objectProperty
WHERE
{
```

```
?objectProperty a owl:ObjectProperty
}

SELECT COUNT DISTINCT ?datatypeProperty
WHERE
{
?datatypeProperty a owl:DatatypeProperty
}
```

2. For this query, we noticed that birthdays of the laureates have two form: a xsd:date type and reference date which is a link. The major part of laureates have both type of dates, however some laureates have only a xsd:date birthday, and some other lauerates have only the reference link of their birthday. The first query, lists all the laureates, with both date types, sorted from the oldest to the youngest. The second query uses a REGEX to list only the laureates who have a xsd:date for their birthday, in order to make the query more consistent:

```
SELECT DISTINCT ?winner ?birth
WHERE {
  ?award rdfs:label "Nobel Prize in Physics"@en .
  ?winner dbo:award ?award .
  ?winner dbo:birthDate ?birth .
}
ORDER BY ?birth
```

```
SELECT DISTINCT ?winner ?birth
WHERE {
  ?award rdfs:label "Nobel Prize in Physics"@en .
  ?winner dbo:award ?award .
  ?winner dbo:birthDate ?birth
  FILTER (REGEX(STR(?birth),"[0-9]{4}-[0-9]{2}-[0-9]{2}"))
}
ORDER BY ?birth
```

3. In this question, we considered that when a laureate purchases a course in a university, then the university has the Nobel Prize in its track record. We used dbo:almaMaster, because it is mentioned in the ontology that there are the schools that the laureate has attended.

```
SELECT ?university (count(?university) as ?count)
WHERE {
  ?award rdfs:label "Nobel Prize in Physics"@en .
  ?winner dbo:award ?award .
  ?winner dbo:almaMater ?university .
}
ORDER BY DESC(?count)
LIMIT 10
```

4. The following query counts the number of winners of the Nobel Prize in physics who are immigrants. We followed the indication and selected the laureates who attended a school in a country different from the country where they are born:

```
SELECT COUNT DISTINCT ?winner
WHERE {
  ?award rdfs:label "Nobel Prize in Physics"@en .
  ?winner dbo:award ?award .
```

```
  ?winner dbo:almaMater ?university .
  ?university dbo:country ?universityCountry .
  ?winner dbo:birthPlace ?birthPlace .
  ?birthPlace dbo:country ?winnerCountry .
  FILTER (?winnerCountry != ?universityCountry) .
}
```

5. The following query provides the list of songs from Bruce Springsteen released between 1980 and 1990:

```
SELECT DISTINCT ?song ?artist ?date
WHERE
{
?song a dbo:Song .
?song dbo:artist ?artist .
?song dbo:releaseDate ?date
FILTER(xsd:string(?artist) = "http://dbpedia.org/resource/Bruce_Springsteen"
       && ?date > "1979-12-31"^^xsd:dateTime
       && ?date < "1991-01-01"^^xsd:dateTime)
}
```

Part V

1. The following query lists all winners of the Nobel Prize in Physics sorted from the oldest to the youngest:

```
SELECT DISTINCT ?winnerLabel
WHERE {
  ?winner p:P166 ?awardStat . # … with an awarded(P166) statement
  ?awardStat ps:P166 wd:Q38104 . # … that has the value Nobel Prize in Physics
(Q38104)
  ?winner wdt:P569 ?birthdate .

SERVICE wikibase:label { bd:serviceParam wikibase:language "en" . }
}
ORDER BY ?birthdate
```

2. The following query lists the top 10 universities with most winners of the Nobel Prize in physics:

```
SELECT ?university ?universityLabel (COUNT(?university) as ?count)
WHERE {
  ?winner p:P166 ?awardStat . # … with an awarded(P166) statement
  ?awardStat ps:P166 wd:Q38104 . # … that has the value Nobel Prize in Physics
(Q38104)
  ?winner wdt:P69 ?university .

SERVICE wikibase:label { bd:serviceParam wikibase:language "en" . }
}
GROUP BY ?university ?universityLabel ?count
ORDER BY DESC(?count)
LIMIT 10
```

3. The following query provides the number of winners of the Nobel Prize in physics who are immigrants:

```
SELECT (COUNT(distinct ?winner) as ?count)
WHERE {
  ?winner p:P166 ?awardStat . # … with an awarded(P166) statement
  ?awardStat ps:P166 wd:Q38104 . # … that has the value Nobel Prize in Physics
(Q38104)
  ?winner wdt:P69 ?university .
  ?university wdt:P17 ?universityCountry .
  ?winner wdt:P19 ?birthPlace .
  ?birthPlace wdt:P17 ?winnerCountry .
  FILTER (?winnerCountry != ?universityCountry)

SERVICE wikibase:label { bd:serviceParam wikibase:language "en" . }
}
```

Performance comparisons with querying the open web

Querying Wikidata seems to be more precise. Indeed, for the winners of Nobel Prize in physics, we obtained 212 results while we obtain 153 distinct results only with the open web. Same remark concerning the immigrants laureates. Indeed, we obtained 66 results with wikidata and only 18 with the Open web. For instance, Marie Curie is immigrant. We find her in the wikidata query, but not in the Open web query.

Part VI

1. Even numbers

```
PREFIX n: <http://km.aifb.kit.edu/projects/numbers/number#NaturalNumber>
SELECT DISTINCT ?number
WHERE
{
  ?number rdf:type n:number   # … Here we precise that number is of type number
  ?number n:primefactor ?p   # … p is the prime factor of number
  FILTER (?p = 2) # … if 2 is a primer factor of the number, it's an even number
}
```

2. Numbers that are successors of one of their prime factors

```
PREFIX n: <http://km.aifb.kit.edu/projects/numbers/number#NaturalNumber>
SELECT DISTINCT ?number
WHERE
{
  ?number rdf:type n:number
  ?number n:primefactor ?p
  FILTER (?number = ?p + 1) # we select numbers that are the successors of one of
}                           # their prime factor
```

3. Odd numbers

```
PREFIX n: <http://km.aifb.kit.edu/projects/numbers/number#NaturalNumber>
SELECT DISTINCT ?number
WHERE
{
  ?number rdf:type n:number
  ?nextNb n:next ?number  # nextNb is the successor of number
  ?nextNb n:primefactor ?p # p is prime factor of nextNb
```

```
  FILTER (?p = 2) # if 2 is prime factor of next number, then next number
}                 # is even, so number is odd
```

4. Prime numbers:

```
PREFIX n: <http://km.aifb.kit.edu/projects/numbers/number#NaturalNumber>
SELECT DISTINCT ?number
WHERE
{
  ?number rdf:type n:number
  ?number n:primefactor ?p
  FILTER (?number = ?p)  # if a number is his own prime factor, then it is a
}                        # prime number
```

5. Non-prime numbers:

```
PREFIX n: <http://km.aifb.kit.edu/projects/numbers/number#NaturalNumber>
SELECT DISTINCT ?number
WHERE
{
  ?number rdf:type n:number
  ?number n:primefactor ?p
  FILTER (?p != ?number) # if a number has a prime factor different than
}                        # itself, it means that this is not a prime number
```

6. Twin primes:

```
PREFIX n: <http://km.aifb.kit.edu/projects/numbers/number#NaturalNumber>
SELECT DISTINCT ?number1 ?number2
WHERE
{
  ?number1 rdf:type n:number
  ?number2 rdf:type n:number
  ?number1 n:primefactor ?p1
  ?number2 n:primefactor ?p2
  FILTER (?p1 != number1    # Here we check that number 1 is prime
          && ?p2 > number2  # Here that number 2 is prime
          && ?number1 = ?number2 + 2)  # Here we filter that the two numbers
}                              # have a distance of 2
```

• • • END • • •
