oefolkes@stud.ntnu.no

TDT44 - Semantic Web

Eirik Folkestad

Task 1: RDF Turtle Ontology

For my ontology for live concerts I have used URI´s for rdf, rdfs, xsd and created my own live concert URI:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX : <http://example.org/schemas/live_concert#>
```

As specified in the task I have built an ontology with rdf and rdfs relations.

In my ontology I have defined seven classes; ConcertTour, LiveConcert, Artist, Song, Repertoire, Location and Genre. The classes are defined as with *rdfs:Class*:

```
:LiveConcert a rdfs:Class .
```

Under Genre I have defined five subclasses; Rock, Electronica, Doom, HipHop and Metal. All are defined with rdfs:subClassOf:

```
:Electronica rdfs:subClassOf :Genre .
```

Every genre is a subclass of Genre with the exception of Metal which is a subclass of Rock.

The properties of a class are defined with rdf:Properties where the range of the property are Classes that I have defined classes defined in rdfs or xsd:

```
:concertArtist a rdf:Property ;
     rdfs:domain :LiveConcert ;
     rdfs:range :Artist .

:concertDate a rdf:Property ;
    rdfs:domain :LiveConcert ;
    rdfs:range xsd:dateTime .
```

```
:location a rdfs:Property ;
    rdfs:domain :Location ;
    rdfs:range rdfs:Literal .
```

ConcertTour
A ConcertTour concists of one or more :LiveConcerts.

LiveConcert
A LiveConcert is defined by the properties :concertArtist, :concertDate and :concertLocation with respective ranges as :Artist, xsd:dateTime and :Location.

Artist
I treat single artists and bands as the same under a class called :Artist. I have done this as I saw it unnecessary for the this assignment to do it otherwise, however this could just as well have been a difference. If I were to differentiate between the two, e.g. I could have defined a superclass called Performer and have Artist and Band as subclasses of this. As a result some queries could have been different than what they are now.

Location
As I have solved this assignment I could just as well have defined :concertLocation's range to be a rdfs:Literal or xsd:String in :LiveConcert but I kept the class just in case I wanted to add more properties to a location e.g. such as gps-coordinates or address and not just a location name.

Genre
Genre is defined as a rdfs:Class but it is not intended have instances of this class. It is intended as an abstract superclass for class for actual genres. I have defined subclasses for this purpose. :Artist have a property called :artistGenre which is defined with a range of :Genre and I have intended instances to be instantiated with one of the subclasses I mentioned earlier. I could have created a subclass of Genre and made every individual of genre be of the type of this class. I believe this would have been smarter for later adding more genres. It would have made for a more logical approach when I rephrased the ontology to OWL DL.

Repertoire
A repertoire is defined with the property :repertoireSongs which takes one or more :Song objects.

Song
A song is defined with the property :songName which takes a rdfs:Literal. For the purpose of this assignment I did not add any more properties but for instance I could have added properties like :songArtist or :songGenre to better describe the song.

Task 2: Querying Ontology

The SPARQL queries are written through the Apache Jena Fuseki 2.4.1 server by uploading my turtle files.

First query

Formulate a SPARQL query that lists all concerts of a given genre and its subgenres that have not yet been held.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX : <http://example.org/schemas/live_concert#>


SELECT ?concert ?artist ?genre ?date
WHERE {
    ?concert a :LiveConcert ;
        :concertDate ?date ;
        :concertArtist ?artist .
    ?artist :artistGenre ?genre .
    ?genre rdfs:subClassOf* :Rock .
    FILTER(?date > "2016-07-02T22:00:00"^^xsd:dateTime)
}
```

In this query I list which concerts (with artist, genre and date) of the rock genre that occur after a given date. I do this by picking all :artistGenres that is rock or a subclass of rock and afterwards filtering out dates that are before the date I check against.

Second Query

Use SPARQL to count and list the songs that two artists share.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX : <http://example.org/schemas/live_concert#>


SELECT ?artist1 ?artist2 ?song1 ?count
WHERE {
```

```
    {
      SELECT DISTINCT ?artist1 ?artist2 ?song1
      WHERE {
        ?artist1 rdf:type :Artist ;
            :artistRepertoire ?rep1 .
        ?rep1 :repertoireSongs ?song1 .
        ?artist2 rdf:type :Artist ;
            :artistRepertoire ?rep2 .
        ?rep2 :repertoireSongs ?song2 .
        FILTER(?artist1 = :Mastodon && ?artist2 = :Gorillaz && ?song1 = ?song2)
      }
    }
  UNION
    {
      SELECT ?artist1 ?artist2 (STR(COUNT(DISTINCT ?song1)) AS ?count)
      WHERE {
        ?artist1 rdf:type :Artist ;
            :artistRepertoire ?rep1 .
        ?rep1 :repertoireSongs ?song1 .
        ?artist2 rdf:type :Artist ;
            :artistRepertoire ?rep2 .
        ?rep2 :repertoireSongs ?song2 .
        FILTER(?artist1 = :Mastodon && ?artist2 = :Gorillaz && ?song1 = ?song2)
      }
      GROUP BY ?artist1 ?artist2
    }
  }
```

This query is divided into two subqueries which are unioned to get the answer in one query but I think this the answer is just as good in two separate queries. In the first subquery I match one distinct artist against another distinct if they share a song. In the next subquery I count the result of the same query as in the first subquery. The result is all songs the two artist share plus one record of the count of the songs.

Third Query

We want to know more about the artists than what is included in your ontology. Show how you in SPARQL can combine your ontology with DBpedia to list the birth dates and possibly the children of the artists you added into your ontology.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX : <http://example.org/schemas/live_concert#>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
```

```
SELECT ?name ?bandMembers (MIN(?bandMembersName) AS ?BandMembersName) ?child
(MIN(?childName) AS ?ChildName) (STR(MIN(?birthDates)) AS ?BirthDates)
WHERE {
    ?artist a :Artist ;
        :artistName ?artistName .
    SERVICE <http://dbpedia.org/sparql> {
        ?band a dbo:Band ;
            dbp:name ?name ;
            dbo:bandMember ?bandMembers .
            OPTIONAL{ ?bandMembers dbp:name ?bandMembersName }
            OPTIONAL{ ?bandMembers dbo:birthDate ?birthDates . }
            OPTIONAL{
                ?child dbo:parent ?bandMembers .
                ?child dbp:name ?childName .
            }
    }
    FILTER(str(?name) = ?artistName)
}
GROUP BY ?name ?bandMembers ?child
```

In this query I needed access to DBpedia's endpoint so that I could use the ontology and property URI's of DPpedia. For this I used the SERVICE keyword which allowed me to use these URI's. Since I could not use local variables inside service I had to do the filtering outside the SERVICE scope.
From DBpedia I get all bands, their names and band members. I it exists (OPTIONAL keyword) I get the band members name, birthdate and their children. Since some of these variables have several values I only choose one of them by the MIN keyword. I also had to include the resources in SELECT since various values were missing in DBpedia.

Fourth Query

How can you use SPARQL to find the most similar – as you define it – artist to a given artist.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX : <http://example.org/schemas/live_concert#>


SELECT ?artist1 ?artist2 (COUNT(DISTINCT ?song1) AS ?count)
WHERE {
    ?artist1 a :Artist ;
        :artistRepertoire ?rep1 .
    ?rep1 :repertoireSongs ?song1 .
    ?artist2 a :Artist ;
        :artistRepertoire ?rep2 .
```

```
    ?rep2 :repertoireSongs ?song2 .
    FILTER(?artist1 = :Mastodon && ?artist2 != :Mastodon && ?song1 = ?song2)
}
GROUP BY ?artist1 ?artist2
ORDER BY DESC(?count)
LIMIT 1
```

This query is very similar to the last subquery in the second query. The only difference is that the second artist is not the same as the distinct first one. Then I sort the result in a descending order and limit the query to return only one “record”. This should be the artist that share most songs with the given one which I define to be the most similar artist.

Task 3: Description Logic

Rephrase/Recreate the ontology in Task 1 in Description Logic (e.g. OWL 2 DL). Explain any additional assumptions you need to make.

Give examples of DL constructions in the ontology that are useful in the ontology, but impossible to include in the RDF(S) ontology. Demonstrate how reasoning works in your ontology.

In my rephrased my ontology with owl so that I can create logical relations between objects which is impossible to include in the RDF(S) ontology. I have put restrictions on object properties in an attempt to make invalid individuals of objects (from my point of view). In the example below I restrict LiveConcert to only have :Artist objects as :concertArtist and :Location as the :concertLocation. I have done likewise for all object properties in my updated ontology.

```
:LiveConcert rdf:type owl:Class ;
             rdfs:subClassOf [ rdf:type owl:Restriction ;
                               owl:onProperty :concertArtist ;
                               owl:allValuesFrom :Artist
                             ] ,
                             [ rdf:type owl:Restriction ;
                               owl:onProperty :concertLocation ;
                               owl:allValuesFrom :Location
                             ] ;
             owl:disjointWith :Location ,
                              :Repertoire ,
                              :Song ,
                              rdfs:Property .
```

I have also set a minimum cardinality on certain properties like :repertoireSongs

```
:Repertoire rdf:type owl:Class ;
            rdfs:subClassOf [ rdf:type owl:Restriction ;
                              owl:onProperty :repertoireSongs ;
                              owl:allValuesFrom :Song
                            ] ,
                            [ rdf:type owl:Restriction ;
                              owl:onProperty :repertoireSongs ;
                              owl:minQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
                              owl:onClass :Song
                            ] ;
            owl:disjointWith :Song ,
                             rdfs:Property .
```

I have also have also restricted :artistGenre to be only one of the subclasses of genre.

```
:Artist rdf:type owl:Class ;
        rdfs:subClassOf [ rdf:type owl:Restriction ;
                          owl:onProperty :artistGenre ;
                          owl:allValuesFrom [ rdf:type owl:Class ;
                                              owl:unionOf ( :Doom
                                                            :Electronica
                                                            :HipHop
                                                            :Metal
                                                            :Rock
                                                          )
                                            ]
                        ] ,
```

I have also updated my ontology so that all classes are disjoint which I believe makes them guaranteed to not be the same since all owl:Class classes inherit from owl:Thing. This would also restrict the open world assumption in my ontology.

```
owl:disjointWith  :Artist ,
                  :ConcertTour ,
                  :Genre ,
                  :LiveConcert ,
                  :Location ,
                  :Repertoire ,
                  :Song ,
                  rdfs:Property .
```

Task 4: Formalism

In the open world assumption a statement is unknown to be true if not explicitly stated. This means that e.g. If the statement is “Eirik lives in Norway.”, it is unknown whether or not Nils also lives in Norway.

In the closed world assumption a statement is false if not explicitly stated otherwise. This means that e.g. if the statement is “Eirik lives in Norway.”, it is false that Nils also lives in Norway.

From my ontology in Task 1 I can use the keyword BIND to demonstrate that if an instance exists I can say something about it.

```
PREFIX : <http://example.org/schemas/live_concert#>


SELECT ?isArtist
WHERE {
    :Gorillaz a ?artist .
    BIND(( ?artist = :Artist) AS ?isArtist) .
}
```

I have defined Gorillaz in my otology and the SPARQL query above returns true since Gorillaz actually is an artist.

```
PREFIX : <http://example.org/schemas/live_concert#>


SELECT ?isArtist
WHERE {
    :ABBA a ?artist .
    BIND(( ?artist = :Artist) AS ?isArtist) .
}
```

I have not defined ABBA and the SPARQL query above returns nothing as there is no artist ABBA. It is not false, but there is nothing to return since it is not explicitly stated that ABBA is an artist..

Appendix

The ontologies are in the delivery folder.