Assignment 1 part 2:
Mapping, Linking and Querying

Due date:	06 March 2023
Submitted date:	05 March 2023
Group participants *(in alphabetical order)*:	J. Bendler F. H. Campbell S. Wang

Task 1: Write the YARRRML or RML mappings to convert the given CSV data to a KG with given RDF structure and prefix-namespace

a. Triples represent information of the song
b. Triples represent information of the artists
c. Triples represent information of the genre

```
prefixes:
    ex: http://example.org/
    schema: https://schema.org/
    mo: http://purl.org/ontology/mo/

mappings:
  song:
    sources:
      - ['music.csv~csv']
    s: http://example.com/song_$(index)
    po:
      - [a, schema:MusicRecording]
      - [schema:byArtist, ex:$(artist)~ini]
      - [schema:genre, ex:$(top genre)~ini]
      - [schema:datePublished, {value: $(year), datatype: xsd:gYear}]
      - [rdfs:label, {value: $(title), language: en}]
  artist:
    sources:
      - ['music.csv~csv']
    s: ex:$(artist)
    po:
      - [a, schema:Person]
      - [rdfs:label, {value: $(artist), language: en}]
  genre:
    sources:
      - ['music.csv~csv']
    s: ex:$(top genre)
    po:
      - [a, mo:Genre]
      - [rdfs:label, {value: $(top genre), language: en}]
```

Download yarrrml file at: yarrrml.yaml
Download resulting knowledge graph at: KG.ttl

Task 2: Linking KG to DBpedia using LIMES
a. All entities with an rdfs:label should be linked to DBpedia entities
b. Add triples with `owl:sameAs` to represent your linking

Using the LIMES Web UI, we generated three configuration files. These are:
- song_config.xml which maps the songs from our knowledge graph to the corresponding dbpedia entries.
- artist_config.xml which maps the artists to the corresponding dbpedia entries.
- genre_config.xml which maps the genres to the corresponding dbpedia entries.

Building blocks in LIMES Web UI used for linking the song name to dbpedia

With the generated XML-files the LIMES program was executed to automatically find matches in dbpedia. The levenshtein value were chosen as follow:
- `0.98` for songs. Despite the rather high value, many songs were mapped to multiple dbpedia entries. This happened very often with songs that were created by different artists but had the same name. Without adding additional information on the artist, this cannot be prevented.
- `0.75` for artists. With this mapping we encountered the problem that either too few (~140) or too many (~70k) artists were found. We choose the result with fewer mappings, because with those it is far more likely that we got “the right ones”.
- `0.4` for genres. Even with this low value, we hardly found any matches. Only with values below 0.4 did we get any results

After the received the following files containing the accepted matches:
- accepted_songs.nt
- accepted_artists.nt
- accepted_genre.nt

The three resulting files were transformed from a N-Triple file (.nt) into a Turtle file (.ttl) using a python script. This script can be found here: transform_to_ttl.ipynb.
Finally we appended the results to our existing knowledge graph. The resulting knowledge graph can be found here: full_KG.ttl.

Task 3: Add the birth date and birth place information of each artist to your KG by querying DBpedia. An example of retrieving all DBpedia information of Ed Sheeran

Finding a way to get the birthdate and birthplace information for each artist from dbpedia caused us much trouble. We already had the dbpedia URIs for each artist from LIMES (see Task 2) and we knew what we were looking for.

The LIMES software couldn't be used to retrieve the birthdate and birthplace information, because LIMES (as far as we are concerned) is only used for connecting similar entities, not for retrieving new information.

The other option was to run SPARQL queries on the dbpedia database. For each artist the query would look like this:

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
SELECT ?birthdate ?birthplace
WHERE {
<artistURI> dbo:birthDate ?birthdate .
<artistURI> dbp:birthPlace ?birthplace .
}
```

To automate these queries and to run them on all our acquired artist URIs, we used another Python script. This script loops over all the artist URIs, creates the SPARQL query and runs it on the http://dbpedia.org/sparql endpoint.
The script can be viewed at: get_birthday_place.ipynb

The resulting properties were converted into triples and saved into a separate knowledge graph file, that can be found here: bdpl.ttl
Finally we added these new triples to our existing graph, resulting in the final knowledge graph. This one includes the following information:

- The data provided from the dataset, that was generated in Task 1
- The `owl:sameAs` connections to dbpedia that were generated in Task 2
- The `dbo:birthDate` and `dbp:birthPlace` information gathered in Task 3

The final resulting knowledge graph can be found here: full_KG_final.ttl

Note:
We first used `dbo:birthPlace` instead of `dbp:birthPlace` which also seemed to work, but for Task 4d) we needed information about the **country** of birth, which is only present in `dbp:birthPlace`.

Task 4: Provide SPARQL queries to answer the following questions using the resulting integrated KG

All presented queries have been tested by using the final resulting knowledge graph, including all links to http://dbpedia.org, as well as the birthdate and birthplace of all artists.

The Jupyter Notebook for these test, along with the **query results** of the following questions can be viewed at: https://github.com/FHCampbell71/KG/blob/main/Assignment%202/KG_query_test.ipynb

a. Return a list of artists and their names who produce songs with genres other than "pop" and "dance pop."

```
PREFIX schema: <https://schema.org/>

SELECT DISTINCT ?artist ?artist_name
WHERE {
  ?song schema:byArtist ?artist .
  ?song schema:genre ?genre .
  ?artist rdfs:label ?artist_name .
  ?genre rdfs:label ?genre_name .
  FILTER(?genre_name != "pop"@en && ?genre_name != "dance pop"@en)
}
```

b. Return a list of songs released in 2016 by artists born before 1990.

```
PREFIX schema: <https://schema.org/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT DISTINCT ?song_name ?released (YEAR(?birthDate) as ?birthYear)
WHERE {
  ?song schema:byArtist ?artist .
  ?song rdfs:label ?song_name .
  ?song schema:datePublished ?released .
  ?artist owl:sameAs ?realArtist .
  ?realArtist dbo:birthDate ?birthDate .
  FILTER(?released = "2016"^^xsd:gYear && YEAR(?birthDate) < 1990)
}
```

c. Who is the artist that has produced the greatest number of songs?

```
PREFIX ex: <http://example.org/>
PREFIX schema: <https://schema.org/>

SELECT ?artist_name (COUNT(?song) as ?sum_songs)
WHERE {
  ?song schema:byArtist ?artist .
  ?artist rdfs:label ?artist_name .
}
GROUP BY ?artist
ORDER BY DESC(?sum_songs)
LIMIT 1
```

d. Return a list of artists born in the USA, sorted by the number of songs they have produced.

```
PREFIX schema: <https://schema.org/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
```

```
SELECT ?artist_name ?birthPlace (COUNT(?song) as ?sum_songs)
WHERE {
  ?song schema:byArtist ?artist .
  ?artist owl:sameAs ?realArtist .
  ?artist rdfs:label ?artist_name .
  ?realArtist dbp:birthPlace ?birthPlace .
  FILTER regex(?birthPlace, "U.S.", "i")
}
GROUP BY ?artist
ORDER BY DESC(?sum_songs)
```

e. Find artists whose song names contain the word "love" and sort the artists by the resulting number of songs.

```
PREFIX schema: <https://schema.org/>

SELECT ?artist_name (COUNT(?song) as ?sum_songs)
WHERE {
  ?song schema:byArtist ?artist .
  ?song rdfs:label ?song_name .
  ?artist rdfs:label ?artist_name .
  FILTER regex(?song_name, "love", "i")
}
GROUP BY ?artist
ORDER BY DESC(?sum_songs)
```