Semantic Web & Ontologies

Pablo Mollá Chárlez

September 30, 2024

Contents

1 Exercise 1

Name	Director	Release Date	Actor
Pulp Fiction	Quentin Tarantino	1994	John Travolta
Taste of Cherry	Abbas Kiarostami	1997	Homayoun Ershadi
Saturday Night Fever	John Badham	1977	John Travolta

Festival Name	Year	Name	Country
Cannes Film Festival	1994	Pulp Fiction	USA
Cannes Film Festival	1997	Taste of Cherry	Iran

1.1 Turtle Code

Let's first design the graph and then write the Turtle code of both tables.

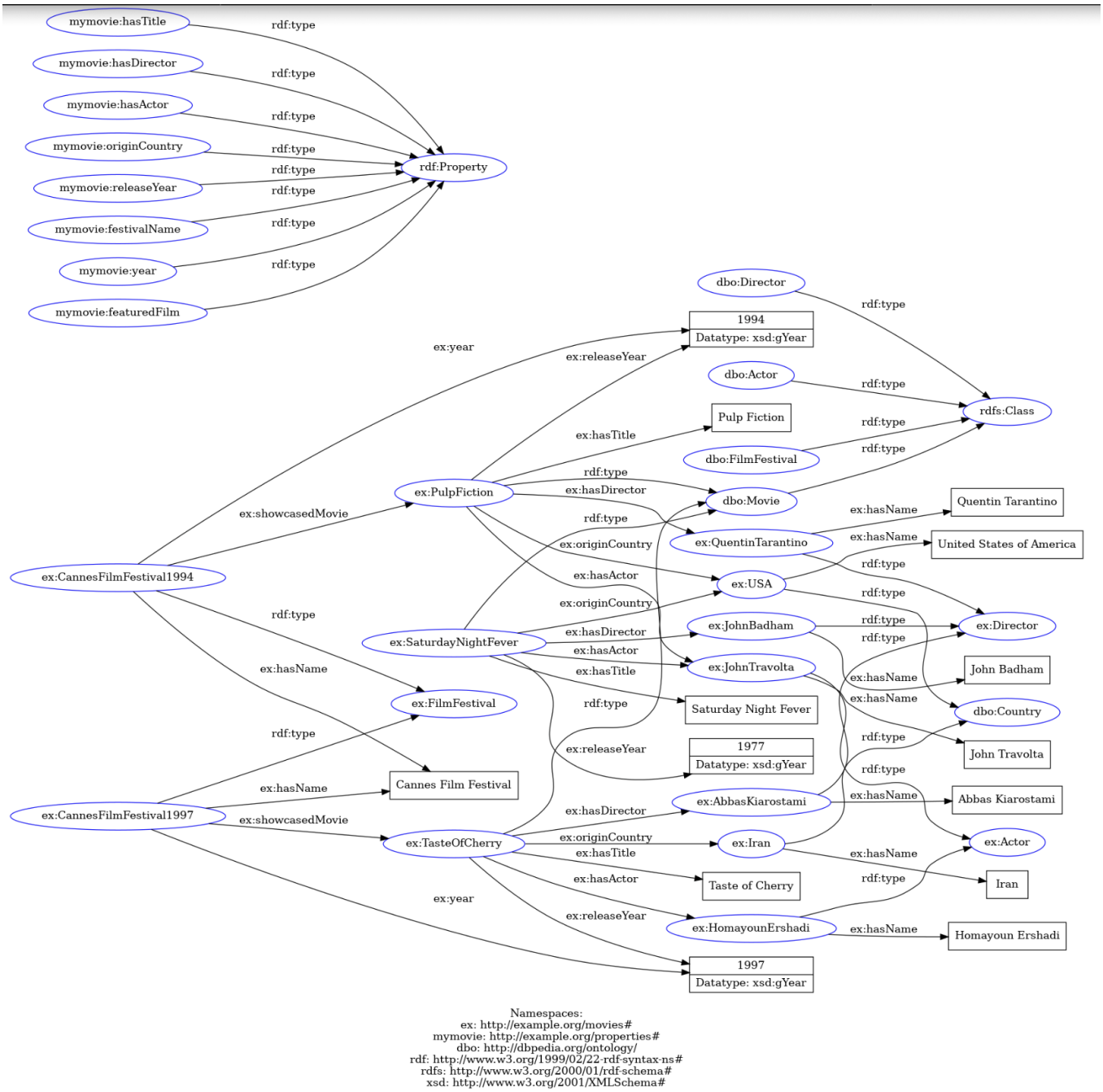
```
# No need to define rdf, rdfs and xsd
# Custom namespace for movies and festivals
@prefix ex: <http://example.org/movies#> .
# Custom namespace for our custom properties
@prefix mymovie: <http://example.org/properties#> .
# DBpedia Dataset
@prefix dbo: <http://dbpedia.org/ontology/> .
```

```
# Define Classes
dbo:Movie rdf:type rdfs:Class .
dbo:Director rdf:type rdfs:Class .
dbo:Actor rdf:type rdfs:Class .
dbo:FilmFestival rdf:type rdfs:Class .


# Define Properties
mymovie:hasTitle rdf:type rdf:Property .
mymovie:hasDirector rdf:type rdf:Property .
mymovie:hasActor rdf:type rdf:Property .
mymovie:originCountry rdf:type rdf:Property .
mymovie:releaseYear rdf:type rdf:Property .
mymovie:festivalName rdf:type rdf:Property .
mymovie:year rdf:type rdf:Property .
mymovie:featuredFilm rdf:type rdf:Property .


# Entities: Countries, Directors and Actors
# Countries
ex:USA rdf:type dbo:Country ;
    ex:hasName "United States of America" .
ex:Iran rdf:type dbo:Country ;
    ex:hasName "Iran" .

# Directors
ex:QuentinTarantino rdf:type ex:Director ;
    ex:hasName "Quentin Tarantino" .
ex:AbbasKiarostami rdf:type ex:Director ;
    ex:hasName "Abbas Kiarostami" .
ex:JohnBadham rdf:type ex:Director ;
    ex:hasName "John Badham" .

# Actors
ex:JohnTravolta rdf:type ex:Actor ;
    ex:hasName "John Travolta" .
ex:HomayounErshadi rdf:type ex:Actor ;
    ex:hasName "Homayoun Ershadi" .


# Movies
# Generating instance PulpFiction from the predefined class Movie within DBpedia
ex:PulpFiction rdf:type dbo:Movie ;
    ex:hasTitle "Pulp Fiction" ;
    ex:hasDirector ex:QuentinTarantino ;
    # gYear is predefined in the XML Schema
    ex:releaseYear "1994"^^xsd:gYear ;
    ex:hasActor ex:JohnTravolta ;
    ex:originCountry ex:USA .

ex:TasteOfCherry rdf:type dbo:Movie ;
    ex:hasTitle "Taste of Cherry" ;
    ex:hasDirector ex:AbbasKiarostami ;
    ex:releaseYear "1997"^^xsd:gYear ;
    ex:hasActor ex:HomayounErshadi ;
    ex:originCountry ex:Iran .

ex:SaturdayNightFever rdf:type dbo:Movie ;
    ex:hasTitle "Saturday Night Fever" ;
    ex:hasDirector ex:JohnBadham ;
    ex:releaseYear "1977"^^xsd:gYear ;
    ex:hasActor ex:JohnTravolta ;
    ex:originCountry ex:USA .
```

```
# Festivals
ex:CannesFilmFestival1994 rdf:type ex:FilmFestival ;
    ex:hasName "Cannes Film Festival" ;
    ex:year "1994"^^xsd:gYear ;
    ex:showcasedMovie ex:PulpFiction .

ex:CannesFilmFestival1997 rdf:type ex:FilmFestival ;
    ex:hasName "Cannes Film Festival" ;
    ex:year "1997"^^xsd:gYear ;
    ex:showcasedMovie ex:TasteOfCherry .
```

1.2 RDF Graph

The RDF graph is described as follows:



2 Exercise 2

Consider the following RDF document in Turtle format with information about celestial bodies.

```
@prefix  ex: <http://example.org/> .
ex:sun          ex:radius      "1.392e6"^^xsd:double ;
                ex:satellite   ex:mercury , ex:venus , ex:earth , ex:mars  .
ex:mercury      ex:radius      "2439.7"^^xsd:double  .
ex:venus        ex:radius      "6051.9"^^xsd:double  .
ex:earth        ex:radius      "6372.8"^^xsd:double  ;
                ex:satellite    ex:moon .
ex:mars         ex:radius       "3402.5"^^xsd:double  ;
                ex:satellite    ex:phobos, ex:deimos  .
ex:moon         ex:name         "Lune@fr", "Moon@en"  ;
                ex:radius       "1737.1"^^xsd:double  .
ex:phobos       ex:name         "Phobos" .
ex:deimos       ex:name         "Deimos" .
```

- **Question 1**: Please give the Graph-based Data Model representation for the RDF triples above.
- **Question 2**: Consider the following SPARQL query for "Object which orbits around the sun or around a satellite of the sun".

```
PREFIX ex: <http://example.org/> .
SELECT ?object
WHERE {
{ ex:Sun ex:satellite ?object . } UNION
{ ex:Sun ex:satellite ?object_tmp .
?object_tmp ex:satellite ?object . }
}
```

  Please give the answer to this query based on the RDF document above (in ttl format). Localise the answer nodes in the graph from Q1.

- **Question 3**: Consider the three SPARQL queries given in Table 1 and the following three questions in natural language.
  - **Option A**: Objects with a satellite for which an English name is given, and which furthermore are satellites of an object with radius greater than 3000 (km).
  - **Option B**: Objects with two or more satellites. Assume for this that different URIs denote different objects.
  - **Option C**: Objects with a radius greater than 3000 (km) together with the object – if it exists – of which they are a satellite.

  For each question above, which is its corresponding SPARQL query from the table below ?

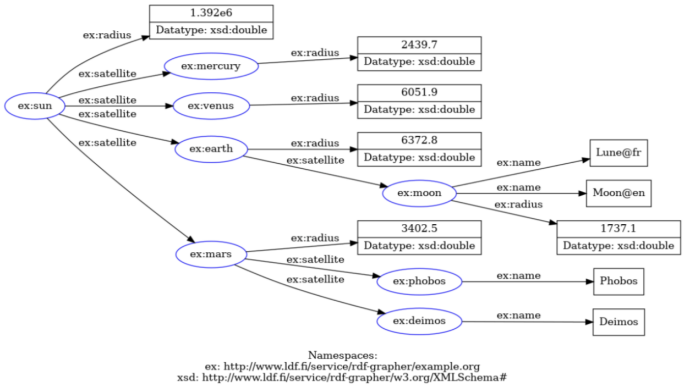
Query 1	`PREFIX ex: <http://example.org/> .`
	`SELECT ?object ?center`
	`WHERE {`
	`{ ?object ex:radius ?rad . }`
	`OPTIONAL { ?center ex:satellite ?object . }`
	`FILTER (?rad > 3000)`
	`}`

Query 2	`PREFIX ex: <http://example.org/> .`
	`SELECT ?object`
	`WHERE {`
	`?object ex:satellite ?satellite .`
	`?satellite ex:name ?name .`
	`?center ex:satellite ?object .`
	`?center ex:radius ?rad .`
	`FILTER (langMATCHES(LANG(?name), "en"))`
	`FILTER (?rad > 3000)`
	`}`

Query 3	`PREFIX ex: <http://example.org/> .`
	`SELECT ?object`
	`WHERE {`
	`?object ex:satellite ?satellite1 .`
	`?object ex:satellite ?satellite2 .`
	`FILTER (!sameTerm(?satellite1, ?satellite2))`
	`}`

- **Question 4**: Can you use ChatGPT on this example? What conclusions can you get compared with queries on the structured data ?

2.1 Answer Q1



2.2 Answer Q2

The query is attempting to find objects (satellites) of the "ex:sun" by looking for:

1. Direct satellites of the sun.
2. Satellites of the satellites (i.e., moons of planets that orbit the sun).

Let's break it down:

- **First part of the UNION**:

```
ex:Sun ex:satellite ?object .
```

  This looks for the direct satellites of the sun. According to the data, "ex:sun" has four direct satellites: "ex:mercury", "ex:venus", "ex:earth", and "ex:mars".

- **Second part of the UNION**:

```
ex:Sun ex:satellite ?object\_tmp .
?object\_tmp ex:satellite ?object
```

  This looks for the satellites of the direct satellites of the sun. From the data:

  – "ex:earth" has one satellite: "ex:moon".
  – "ex:mars" has two satellites: "ex:phobos" and "ex:deimos".

  The results of the query would be a combination of:

- Direct satellites of the sun ("ex:mercury", "ex:venus", "ex:earth", "ex:mars").
- Satellites of those planets ("ex:moon", "ex:phobos", "ex:deimos").

2.3 Answer Q3

- Option A $\longleftrightarrow$ Query 2
- Option B $\longleftrightarrow$ Query 3
- Option C $\longleftrightarrow$ Query 1

3 Exercise 3

Write SPARQL queries to answer the movie related questions below using the displayed Turtle data.

- Names of all movies.
- Names of movies and directors sorted descending by the year the movie appeared.
- Names and directors of all movies before1996.
- Names all movies whose genre is Crime.
- Names of all actors who are above 50 (at 2016).
- Names of all movies whose directors are above 70 (at 2016).

```
@prefix ex: <http://example.org/movies/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ex:m1   rdf:type ex:Movie;
        ex:genre ex:Drama;
        ex:year "2006"^^xsd:gYear;
        rdfs:label "Marie_Antoinette";
        ex:country ex:USA;
        ex:director ex:p1;
        ex:actor ex:p2 .
ex:p1   rdf:type ex:Director;
        foaf:familyName "Coppola";
        foaf:givenName  "Sofia";
        ex:birthYear "1971"^^xsd:gYear .
ex:p2   rdf:type ex:Actor;
        foaf:familyName "Dunst";
        foaf:givenName  "Kirsten";
        ex:birthYear "1982"^^xsd:gYear .
ex:p5   rdf:type ex:Actor;
        foaf:familyName "De_Niro";
        foaf:givenName  "Robert";
        ex:birthYear "1943"^^xsd:gYear .
ex:m2 rdf:type ex:Movie;
        ex:genre ex:Crime;
        ex:year "1995"^^xsd:gYear;
        rdfs:label "Heat";
        ex:country ex:USA;
        ex:director ex:p3;
        ex:actor ex:p4 , ex:p5.
ex:p3   rdf:type ex:Director;
        foaf:familyName "Mann";
        foaf:givenName  "Michael";
        ex:birthYear "1943"^^xsd:gYear .
ex:p4   rdf:type ex:Actor;
        foaf:familyName "Pacino";
        foaf:givenName  "Al";
        ex:birthYear "1940"^^xsd:gYear .
```

The Turtle code is described below:

1. Name of all movies:

```
PREFIX ex: <http://example.org/movies/> .
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?filmName
WHERE  {
        ?film rdf:type ex:Movie ;
                rdfs:label ?filmName .

}
```

2. Names of movies and directors sorted descending by the year the movie appeared.

```
PREFIX ex: <http://example.org/movies/> .
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
PREFIX foaf: <http://xmlns.com/foaf/0.1/> .

SELECT ?filmName ?directorName ?filmYear
WHERE  {
        ?film rdf:type ex:Movie ;
                # Use "?" to specify the data to retrieve
                rdfs:label ?filmName ;
                ex:year ?filmYear ;
                ex:director ?director .
        ?director foaf:familyName ?familyName ;
                    foaf:givenName ?givenName .
        # Concatenation of givenName + familyName
        # and assigned to variable directorName
        BIND(CONCAT(?givenName, " ", ?familyName) AS ?directorName)

}

ORDER BY desc(?filmYear)
```

3. Names and directors of all movies before 1996.

```
PREFIX ex: <http://example.org/movies/> .
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
PREFIX foaf: <http://xmlns.com/foaf/0.1/> .

# It will return one attribute more (year) but I prefer it this way
SELECT ?filmName ?directorName ?filmYear
WHERE {
        ?film rdf:type ex:Movie ;
              rdfs:label ?filmName ;
              ex:year ?filmYear ;
              ex:director ex:?director .
        ?director rdf:type ex:Director ;
                  foaf:givenName ?givenName ;
                  foaf:familiName ?familyName .
        FILTER(?year < 1996)
        BIND(CONCAT(?givenName, " ", ?familyName), AS ?directorName)

}
```

4. Names all movies whose genre is Crime.

```
PREFIX ex: <http://example.org/movies/> .
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
PREFIX foaf: <http://xmlns.com/foaf/0.1/> .

SELECT  ?filmName ?filmGenre
WHERE {
        ?film rdf:type ex:Movie ;
              rdfs:label ?filmName ;
              ex:genre ex:Crime .
}
```

5. Names of all actors who are above 50 (at 2016).

```
PREFIX ex: <http://example.org/movies/> .
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
PREFIX foaf: <http://xmlns.com/foaf/0.1/> .

SELECT ?actorName
WHERE {
       ?actor rdf:type ex:Actor ;
              foaf:familyName ?actorFamName ;
              foaf:givenName ?actorGivName ;
              ex:birthYear ?actorYear .
        FILTER((2016 - ?actorYear) > 50)
        BIND(CONCAT(?actorGivName, " ", ?actorFamName) AS ?actorName)
}
```

6. Names of all movies whose directors are above 70 (at 2016).

```
PREFIX ex: <http://example.org/movies/> .
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
PREFIX foaf: <http://xmlns.com/foaf/0.1/> .

SELECT ?filmName
WHERE {
       ?film rdf:type ex:Movie ;
              rdfs:label ?filmName ;
              ex:director ?director .
        ?director rdf:type ex:Director ;
                  ex:birthYear ?directorYear .
        FILTER((2016 - ?directorYear) > 70)
        # Another way would be
        # BIND((2016 - ?directorYear) AS ?age)
        # FILTER(?age > 70)
}
```

4 Exercise 4

Install Fuseki (go to https ://jena.apache.org/download to download, then start "fuseki-server" on the command line and go to http ://localhost :3030). Load the data from Exercise 2 (in .ttl format) and try the corresponding SPARQL queries. Experiment with leaving out and/or adding further triple patterns. Done.

5 Exercise 5

The task is to create SPARQL queries for DBpedia. You can test the queries on SPARQL or Live SPARQL, or in code (see an example ex.py). Please note that DBpedia data in this endpoint is subject to change (in particular the live version will be frequently updated). In some cases, this can result in the queries below no longer working and/or the changes in properties or classes.

For each question, we provide you with the DBpedia entities, which are not in the RDF, RDFS or OWL namespace (e.g. rdf:type is not listed). The following prefixes should be used (note that the DBpedia endpoint already has those predefined as listed in here so you do not need to manually add them in your SPARQL queries):

```
PREFIX rdf: <http://www.w3. org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX res: <http://dbpedia.org/resource/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX yago: <http://dbpedia.org/class/yago/>
```

We distinguish between individuals (specific entities e.g. Angela Merkel), classes (sets of individuals) and properties (connecting individuals), so you can more easily define your queries.

1. How tall is Claudia Schiffer ? (Classes : –, Properties : **dbo:height**, Individuals : **res:Claudia_Schiffer**)

```
PREFIX rdf: <http://www.w3. org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX res: <http://dbpedia.org/resource/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX yago: <http://dbpedia.org/class/yago/>

SELECT ?ClaudiaSchifferHeight
WHERE  {
        res:Claudia_Schiffer dbo:height ?ClaudiaSchifferHeight .

}
```

2. Give me all female Russian astronauts. (Classes : **yago:RussianCosmonauts**, **yago :FemaleAstronauts**, **yago :WikicatRussianCosmo-nauts**, Properties : –, Individuals : –)

```
PREFIX rdf: <http://www.w3. org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX res: <http://dbpedia.org/resource/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX yago: <http://dbpedia.org/class/yago/>

SELECT ?astronautLink ?astronautName
WHERE  {
       # Searching for entity belonging to 3 classes
       ?astronautLink rdf:type yago:RussianCosmonauts ;
       ?astronautLink rdf:type yago:FemaleAstronauts ;
       ?astronautLink rdf:type yago:WikicatRussianCosmo-nauts ;

       # Supposing the class has a property rdfs:label to extract the name
       ?astronautLink rdfs:label ?nameAstronaut .

       # Filtering by English language
       FILTER(lang(?astronautName) = "en")

}
```

Question: How to find out the proper URIs of classes/properties/individuals to be used in a query ?
Answer: I would visit DBpedia and search for entities like "Russian astronauts" or "Female astronauts" to check their properties and classes. This method helps to discover the exact properties and URIs.

3. How many monarchical countries are there in Europe?(Classes : **yago:EuropeanCountries**, Properties : **dbo :governmentType**, Individuals : –)

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX res: <http://dbpedia.org/resource/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX yago: <http://dbpedia.org/class/yago/>

SELECT (COUNT(?country) AS ?monarchicalCountriesCount)
WHERE {
      ?country rdf:type yago:EuropeanCountries ;
               dbo:governmentType ?governmentType .
      # To check if governmentType contains the string "monarchy"
      # "i" stands for case-insensitive, meaning it will
      # match any variation of the word "monarchy"
      FILTER regex(?governmentType, "monarchy", "i")
}
```

4. Which states of Germany are governed by the Social Democratic Party ? (Classes : **yago:StatesOfGermany**, Properties : **dbp:rulingParty**, Individuals : **res:Social_Democratic_Party_of_Germany**)

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX res: <http://dbpedia.org/resource/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX yago: <http://dbpedia.org/class/yago/>

SELECT ?germanStates
WHERE {
      ?germanStates rdf:type yago:StatesOfGermany ;
                    dbp:rulingParty res:Social\_Democratic\_Party\_of\_Germany .
}
```

5. Which monarchs of the United Kingdom were married to a German? (Classes : **yago:MonarchsOfTheUnitedKin**, Properties : **dbo:spouse**, **dbo:birthPlace**, Individuals : **res:Germany**)

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX res: <http://dbpedia.org/resource/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX yago: <http://dbpedia.org/class/yago/>

SELECT ?monarchs
WHERE {
      ?monarchs rdf:type yago:MonarchsOfTheUnitedKingdom ;
                dbo:spouse ?spouse .
      ?spouse dbo:birthPlace res:Germany .
}
```

6. Which countries have places with more than two caves? (Classes : **dbo:Cave**, **dbo:Country**, Properties : **dbo:location**, Individuals : -)

```
PREFIX rdf: <http://www.w3. org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX res: <http://dbpedia.org/resource/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX yago: <http://dbpedia.org/class/yago/>

SELECT ?country (COUNT(?cave) AS ?CaveCount)
WHERE {
      ?country rdf:type dbo:Country .
      ?cave rdf:type dbo:Cave ;
            dbo:location ?country .
}
# GROUP BY used to group the results by one or more variables (?country)
GROUP BY ?country
# HAVING is used only with aggregate functions and is typically
# used to filter the results after the aggregation is performed
# Similar to FILTER but only works on results
# of aggregated data not individual rows
HAVING (COUNT(?cave) > 2)
```

7. Give me all cities in New Jersey with more than 100000 inhabitants. (Classes : **dbo:City**, Properties : **dbo:isPartOf**, **dbp:populationTotal**, Individuals : **res:New_Jersey**)

```
PREFIX rdf: <http://www.w3. org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX res: <http://dbpedia.org/resource/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX yago: <http://dbpedia.org/class/yago/>

SELECT ?city
WHERE {
      ?city rdf:type dbo:City ;
            dbo:isPartOf res:New_Jersey ;
            dbp:populationTotal ?population .
}
FILTER (?population > 100000)
```

8. Is proinsulin a protein ? (Classes : **dbo:Protein**, Properties : –, Individuals : **res:Proinsulin**)

```
PREFIX rdf: <http://www.w3. org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX res: <http://dbpedia.org/resource/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX yago: <http://dbpedia.org/class/yago/>

SELECT ?proinsulin
WHERE {
      res:Proinsulin rdf:type dbo:Protein .
}
```

We need to point out that the variable ?proinsulin is not explicitly used in the WHERE clause. If we don't need to retrieve any properties related to Proinsulin, and we only want to check if res:Proinsulin is of type dbo:Protein, the variable isn't necessary.

Another possible solution would be:

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX res: <http://dbpedia.org/resource/>

ASK WHERE {
          res:Proinsulin rdf:type dbo:Protein .
}
```

9. Is Frank Herbert still alive ? (Classes : –, Properties : **dbo:deathDate**, Individuals : **res:Frank_Herbert**)

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX res: <http://dbpedia.org/resource/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX yago: <http://dbpedia.org/class/yago/>

SELECT ?FrankHerbert
WHERE {
      res:Frank_Herbert dbo:deathDate ?deathDate ;
}
```

To check if Frank Herbert is still alive, we can query for his death date. If a death date exists, it means he is not alive. Another possible solution would be:

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX res: <http://dbpedia.org/resource/>

ASK WHERE {
  res:Frank_Herbert dbo:deathDate ?deathDate .
}
```

10. Which mountain is the highest after the Annapurna ? (Classes : **dbo:Mountain**, Properties : **dbo:elevation**, Individuals : **res:Annapurna**)

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX res: <http://dbpedia.org/resource/>

SELECT ?mountain ?elevation
WHERE {
  ?mountain rdf:type dbo:Mountain ;
            dbo:elevation ?elevation .
  FILTER (?mountain != res:Annapurna)
}
ORDER BY DESC(?elevation)
LIMIT 1
```