

---

Université de Montpellier

Année 2023-2024

Faculté des Sciences

30 Place E.Bataillon, 34095 Montpellier

# Rapport de Travaux Pratiques

## TP n°3

## Interrogation de données RDF

par

Romain GALLERNE

*Encadrant de TP :* M Frederico Ulliana

*Responsable du module :* M Frederico Ulliana

# Table des matières

<b>1</b>	<b>SPARQL : Interrogation et méta-interrogation</b>	<b>3</b>
1.1	Séparer les triplets contenant des connaissances ontologiques des triples représentant des données. . . . .	3
1.2	Requêtes SPARQL . . . . .	4
1.2.1	Donner la requête SPARQL qui sélectionne tous les (identifiants des) films . . . . .	4
1.2.2	Donner la requête SPARQL qui sélectionne tous les (identifiants des) films, cette fois ci en prenant en compte la sémantique de <code>rdfs :subClassOf</code> . . . . .	4
1.2.3	Donner la requête SPARQL qui sélectionne toutes les sous-classes de la classe <code>artiste</code> . . . . .	4
1.2.4	Donner la requête SPARQL qui sélectionne tous les artistes, cette fois ci en prenant en compte la sémantique de <code>rdfs :subClassOf</code> . . .	4
<b>2</b>	<b>L'exploration d'un endpoint SPARQL : le cas de DBPedia</b>	<b>5</b>
2.1	Choisissez une classes de l'ontologie qui commence par la première lettre de votre nom, et donnez la liste de ses sous-classes . . . . .	5
2.2	Propriétés . . . . .	6
2.2.1	Donner la liste des propriétés employées pour décrire les instances de la classe choisie . . . . .	6
2.2.2	Vérifier si ces propriétés ont des sous-propriétés . . . . .	7
<b>3</b>	<b>Interrogation du réseau social de l'UE</b>	<b>8</b>
3.1	Donner des requêtes SPARQL permettant de répondre aux questions suivantes . . . . .	8
3.1.1	Est il vrai que <code>GpersonalFriend C= GsocialFriend</code> ? . . . . .	8
3.1.2	Est il vrai que <code>GsocialFriend C= GpersonalFriend</code> ? . . . . .	9

3.1.3	Est il vrai que $G_{\text{personalFriend}}, G_{\text{socialFriend}} \subseteq G_{\text{knows}}$ ? . . . .	10
3.2	Répondez aux questions suivantes pour chaque graphe ( $G_{\text{personalFriend}}$ , et $G_{\text{socialFriend}}$ ) . . . . .	10
3.2.1	Combien de noeuds et d'arêtes y a-t-il dans le graphe ? . . . . .	10
3.2.2	S'agit-il d'un graphe fortement connexe ? . . . . .	11
3.2.3	S'agit-il d'un graphe connexe ? . . . . .	11
3.2.4	Donner la liste de vos voisins dans le graphe . . . . .	13
3.2.5	Calculez le degré de chaque noeud (nombre de voisins) . . . . .	14
3.2.6	Combien de triangles y-a-t il dans le graphe ? . . . . .	15
3.2.7	Combien de triangles peuvent encore se former dans le graphe ? . .	16
3.2.8	Donner la liste de toutes les cliques de cardinalité 4 dans le graphe .	16
3.2.9	Donner la taille de chaque composante connexe du graphe . . . . .	17

# SPARQL : Interrogation et méta-interrogation

## 1.1 Séparer les triplets contenant des connaissances ontologiques des triples représentant des données.

On distingue deux types de triplets dans un document RDF. Les triplets ontologiques représentent les connaissances du domaine qu'on obtient le plus souvent auprès des experts (exemple : un film est dirigé par un directeur). Les triplets des données représentent les données que l'on souhaite enregistrer (exemple : Le film "Star wars" a été dirigé par "Georges Lucas").

Soient les triplets représentant les connaissances ontologiques :

- `movies :directedBy rdfs :domain movies :Movie .`
- `movies :title rdfs :domain movies :Movie .`
- `movies :directedBy rdfs :range movies :Director .`
- `movies :playsIn rdfs :domain movies :Actor .`
- `movies :playsIn rdfs :range movies :Movie .`
- `movies :Actor rdfs :subClassOf movies :Artist .`
- `movies :Director rdfs :subClassOf movies :Artist .`
- `movies :title rdf :type owl :DataTypeProperty .`

Les triplets représentants les données :

- `movies :m2 movies :title "Vertigo" .`
- `movies :m1 rdf :type movies :Movie .`
- `movies :m3 movies :directedBy dbp :Alfred_Hitchcock .`
- `movies :a1 movies :playsIn movies :m4 .`

## 1.2 Requêtes SPARQL

### 1.2.1 Donner la requête SPARQL qui sélectionne tous les (identifiants des) films

```
SELECT ?filmID WHERE ?filmID a movies :Movie
```

Il s'agit d'une simple interrogation sur les éléments de la classe Movie.

### 1.2.2 Donner la requête SPARQL qui sélectionne tous les (identifiants des) films, cette fois ci en prenant en compte la sémantique de rdfs :subClassOf

```
SELECT ?filmID WHERE ?filmID a ?Class . ?Class rdfs :subClassOf movies :Movie .
```

Il s'agit d'une méta-interrogation car on interroge les éléments du schéma, notamment les classes qui sont des sous-classes de Movie.

### 1.2.3 Donner la requête SPARQL qui sélectionne toutes les sous-classes de la classe artiste

```
SELECT ?Class WHERE ?Class rdfs :subClassOf movies :Artist .
```

Il s'agit d'une méta-interrogation car on interroge les éléments du schéma, notamment les classes qui sont des sous-classes de Artist.

### 1.2.4 Donner la requête SPARQL qui sélectionne tous les artistes, cette fois ci en prenant en compte la sémantique de rdfs :subClassOf

```
SELECT ?artistID WHERE ?artistID a ?Class . ?Class rdfs :subClassOf movies :Artist
```

Il s'agit d'une méta-interrogation car on interroge les éléments du schéma, notamment les classes qui sont des sous-classes de Artist.

# L'exploration d'un endpoint SPARQL : le cas de DBPedia

## 2.1 Choisissez une classes de l'ontologie qui commence par la première lettre de votre nom, et donnez la liste de ses sous-classes

On execute la requête suivante pour trouver toutes les ontologies commençant par la lettre "G" (pour GALLERNE) :

```
PREFIX rdf : <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl : <http://www.w3.org/2002/07/owl#>
SELECT DISTINCT ?varClass
WHERE { ?varClass rdf:type owl:Class
FILTER regex(varClass,"http://dbpedia.org/ontology/G")}
LIMIT 100
```

On choisit à l'issue de cela l'ontologie "Genre" dont voici les sous-classes suite à la requête :

```
PREFIX rdf : <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl : <http://www.w3.org/2002/07/owl#>
PREFIX rdfs : <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbpedia : <http://dbpedia.org/ontology/>
SELECT DISTINCT ?varClass
WHERE { ?varClass rdfs:subClassOf dbpedia:Genre .
?varClass rdf:type owl:Class .} LIMIT 100
```

On obtient alors les sous-classes :

- ArtisticGenre
- MovieGenre
- LiteraryGenre
- MusicGenre

## 2.2 Propriétés

### 2.2.1 Donner la liste des propriétés employées pour décrire les instances de la classe choisie

```
PREFIX rdf : <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl : <http://www.w3.org/2002/07/owl#>
PREFIX rdfs : <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbpedia : <http://dbpedia.org/ontology/>
SELECT DISTINCT ?property
WHERE { ?instance rdf:type dbpedia:Genre .
?instance ?property ?value.}
LIMIT 100
```

La liste des propriétés obtenus est très grande, trop grande pour être entièrement retranscrite ici, en voici quelques éléments :

- <http://www.w3.org/2002/07/owl#differentFrom>
- <http://www.w3.org/2000/01/rdf-schema#seeAlso>
- <http://xmlns.com/foaf/0.1/name>
- <http://dbpedia.org/property/name>
- <http://dbpedia.org/ontology/wikiPageID>
- <http://dbpedia.org/ontology/wikiPageRevisionID>
- <http://dbpedia.org/ontology/wikiPageWikiLink>
- ...

### 2.2.2 Vérifier si ces propriétés ont des sous-propriétés

Pour obtenir les sous propriétés on entre la requête suivante :

```
PREFIX rdf : <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl : <http://www.w3.org/2002/07/owl#>
PREFIX rdfs : <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbpedia : <http://dbpedia.org/ontology/>
SELECT DISTINCT ?subproperty
WHERE { ?instance rdf:type dbpedia:Genre .
?instance ?property ?value.
?subproperty rdfs:subPropertyOf ?property. }
LIMIT 100
```

Le résultat étant vide, on en déduit qu'il n'y a pas de sous propriétés associés aux propriétés de la classe "Genre".



# Interrogation du réseau social de l'UE

## 3.1 Donner des requêtes SPARQL permettant de répondre aux questions suivantes

### 3.1.1 Est il vrai que $G_{\text{personalFriend}} \subseteq G_{\text{socialFriend}}$ ?

```
SELECT ?s ?o
WHERE { ?s tsd :personalFriend ?o.
MINUS { ?s tsd :socialFriend ?o. }}
ORDER BY ?s ?o
```

La conclusion est que  $G_{\text{personalFriend}}$  n'est pas inclus dans  $G_{\text{socialFriend}}$ , sinon le résultat de cette requête serait vide, or ici on obtient le tableau suivant :

s	o
tsd:Adam_ElHorri	tsd:Tai_Nguyen
tsd:Adam_ElHorri	tsd:otya_kikadidi-zoula
tsd:Adam_ElHorri	tsd:rafik_amara
tsd:Adam_ElHorri	tsd:rayan_malandain
tsd:Ibrahim_Simpara	tsd:Bomboclaat
tsd:aurelien_milla	tsd:aurelien_milla
tsd:elliott_mazerand	tsd:loris_gpExplorer2
tsd:g_d	tsd:loris_gp
tsd:loris_gp	tsd:sebastien_vial
tsd:otya_kikadidi-zoula	tsd:rafik_amara
tsd:rafik_amara	tsd:Tai_Nguyen
tsd:romain_campillo	tsd:Lord_laurent
tsd:romain_gallerne	tsd:alice_dupont
tsd:sebastien_vial	tsd:Bamboclaat
tsd:sebastien_vial	tsd:loris_gp
tsd:thomas_razafimbahoaka	tsd:Tom_Gimat

Figure 1 : Résultat de la première requête MINUS.

### 3.1.2 Est il vrai que GsocialFriend C= GpersonalFriend ?

```
SELECT ?s ?o
WHERE { ?s tsd :socialFriend ?o.
MINUS { ?s tsd :personalFriend ?o. }}
ORDER BY ?s ?o
```

La conclusion est que GsocialFriend n'est pas inclus dans GsocialFriend, sinon le résultat de cette requête serait vide, or ici on obtient le tableau suivant :

s	o
tsd:Adam_ElHorri	tsd:Ibrahim_simpara
tsd:Adam_ElHorri	tsd:Thomas_Geoffroy
tsd:Adam_ElHorri	tsd:amaury_rebillard
tsd:Adam_ElHorri	tsd:marta_boshkovska
tsd:Adam_ElHorri	tsd:thomas_razafimbahoaka
tsd:Bomboclaat	tsd:Ibrahim_Simpara
tsd:Bomboclaat	tsd:Loreena_Barrere
tsd:Bomboclaat	tsd:Tai_Nguyen
tsd:Bomboclaat	tsd:Thomas_Geoffroy
tsd:Bomboclaat	tsd:Wemby
tsd:Bomboclaat	tsd:otya_kikadidi-zoula
tsd:Ibrahim_Simpara	tsd:Tom
tsd:Ibrahim_Simpara	tsd:marta_boshkovska
tsd:Loreena_Barrere	tsd:Adam_ElHorri
tsd:Loreena_Barrere	tsd:Tai_Nguyen
tsd:Tom	tsd:Bomboclaat
tsd:Tom	tsd:Ibrahim_Simpara
tsd:Tom	tsd:amaury_rebillard
tsd:Tom	tsd:jules_cassan
tsd:Tom	tsd:otya_kikadidi-zoula
tsd:Tom	tsd:thomas_razafimbahoaka
tsd:Wemby	tsd:Bomboclaat
tsd:Wemby	tsd:Loreena_Barrere
tsd:Wemby	tsd:Tai_Nguyen
tsd:Wemby	tsd:Thomas_Geoffroy
tsd:Wemby	tsd:ibrahim_simpara
tsd:Wemby	tsd:otya_kikadidi-zoula
tsd:amaury_rebillard	tsd:Adam_ElHorri
tsd:amaury_rebillard	tsd:Tom
tsd:amaury_rebillard	tsd:thomas_razafimbahoaka
tsd:chaymae_chouari	tsd:charles_langlois
tsd:chaymae_chouari	tsd:laurencia_dovi-late
tsd:chaymae_chouari	tsd:marta_boshkovska
tsd:elliott_mazerand	tsd:sebastienCortexpyramide
tsd:g_d	tsd:marta_boshkovska
tsd:laurencia_dovi-late	tsd:g_n
tsd:loris_gp	tsd:elliott_mazerand
tsd:marta_boshkovska	tsd:Ibrahim_Simpara
tsd:marta_boshkovska	tsd:Tai_Nguyen
tsd:marta_boshkovska	tsd:g_n

Figure 2 : Résultat de la seconde requête MINUS.

### 3.1.3 Est il vrai que $G_{\text{personalFriend}}, G_{\text{socialFriend}} \subseteq G_{\text{knows}}$ ?

```
SELECT ?s ?o WHERE {  
  {SELECT ?s ?o WHERE { ?s tsd :personalFriend ?o. }} UNION  
  {SELECT ?s ?o WHERE { ?s tsd :socialFriend ?o. }}  
  MINUS { ?s foaf :knows ?o. } ORDER BY ?s ?o.
```

Ici, on remarque que cette requête ne renvoie aucun résultat, on en conclut donc que  $G_{\text{personalFriend}}, G_{\text{socialFriend}} \subseteq G_{\text{knows}}$ . C'est logique puisque  $G_{\text{personalFriend}}$  et  $G_{\text{socialFriend}}$  sont définis comme des sous propriétés de  $G_{\text{knows}}$ .

## 3.2 Répondez aux questions suivantes pour chaque graphe ( $G_{\text{personalFriend}}$ , et $G_{\text{socialFriend}}$ )

### 3.2.1 Combien de noeuds et d'arêtes y a-t-il dans le graphe ?

```
SELECT (count(DISTINCT ?s) as ?sommet)  
WHERE {  
  { ?s tsd :personalFriend ?o }  
  UNION { ?o tsd :personalFriend ?s }  
}
```

```
SELECT (count(DISTINCT *) as ?arrete)  
WHERE { ?s tsd :personalFriend ?o. }
```

Il y a 54 sommets et 55 arêtes dans le graphe  $G_{\text{personalFriend}}$ .

```
SELECT (count(DISTINCT ?s) as ?sommet)  
WHERE {  
  { ?s tsd :socialFriend ?o }  
  UNION { ?o tsd :socialFriend ?s }  
}
```

```
SELECT (count(DISTINCT *) as ?arrete)  
WHERE { ?s tsd :socialFriend ?o. }
```

Il y a 37 sommets et 92 arêtes dans le graphe  $G_{\text{socialFriend}}$ .

### 3.2.2 S'agit-il d'un graphe fortement connexe ?

Un graphe orienté est dit fortement connexe s'il existe un chemin dirigé entre chaque paire de nœuds dans le graphe. Cela signifie qu'il est possible de se déplacer de n'importe quel nœud à n'importe quel autre nœud en suivant les flèches du graphe.

Le graphe orienté est fortement connexe si et seulement si, pour chaque paire de nœuds (s, o) dans le graphe, il existe un chemin dirigé de s à o et un chemin dirigé de o à s.

```
SELECT (COUNT(*) as ?count)
WHERE { ?s tsd :personalFriend+ ?o.
MINUS { ?o tsd :personalFriend+ ?s.}}
```

Le résultat est ici 89, ce graphe n'est donc pas fortement connexe. S'il l'était le résultat de cette requête serait 0 car pour tout couple de sommet s et o il existerait un chemin de s vers o et de o vers s.

```
SELECT (COUNT(*) as ?count)
WHERE { ?s tsd :socialFriend+ ?o.
MINUS { ?o tsd :socialFriend+ ?s.}}
```

Le résultat est ici 175, ce graphe n'est donc pas fortement connexe.

### 3.2.3 S'agit-il d'un graphe connexe ?

```
SELECT ?s (count( ?o) as ?nbConnexe)
WHERE { ?s (tsd :personalFriend|^ tsd :personalFriend)+ ?o.
FILTER ( ?s != ?o)} GROUP BY ?s
```

Pour vérifier que le graphe est connexe on compte pour chaque personne le nombre de personne avec qui elle est connexe. On remarque donc qu'il y a plusieurs composantes connexe dans le graphe : une à 22 sommets, une à deux sommets et une à quatre sommets. Le graphe n'est donc, globalement, pas connexe.

s	nbConnexe
tsd:charles_langlois	21
tsd:marta_boshkovska	1
tsd:sebastien_vial	3
tsd:thomas_razafimbahoaka	21
tsd:romain_gallerne	21
tsd:Adam_ElHorri	21
tsd:g_d	3
tsd:Tom_Gimat	21
tsd:elliott_mazerand	1
tsd:Lord_laurent	21
tsd:otya_kikadidi-zoula	21
tsd:Adam_Elhorri	21
tsd:Ibrahim_Simpara	21
tsd:mohamed_dahmoun	21
tsd:rafik_amara	21
tsd:Loreena_Barrere	21
tsd:laurencia_dovi-late	1
tsd:Bomboclaat	21
tsd:ayman_benazzouz	21
tsd:amaury_rebillard	21
tsd:Tai_Nguyen	21
tsd:chaymae_chouari	21
tsd:jules_cassan	21
tsd:loris_gp	3
tsd:romain_campillo	21
tsd:Bamboclaat	3
tsd:alice_dupont	21
tsd:Thomas_Geoffroy	21
tsd:loris_gpExplorer2	1
tsd:rayan_malandain	21

Figure 3 : Résultat de la requête de connexité personalFriend.

```
SELECT ?s (count( ?o) as ?nbConnexe)
WHERE { ?s (tsd :socialFriend|^ tsd :socialFriend)+ ?o.
FILTER ( ?s!= ?o)} GROUP BY ?s
```

Il y a plusieurs composantes connexe dans le graphe : une à 29 sommets et une à 4 sommets. Le graphe n'est donc, globalement, pas connexe.

s	nbConnexe
tsd:charles_langlois	28
tsd:Ibrahim_simpara	28
tsd:marta_boshkovska	28
tsd:sebastien_vial	3
tsd:thomas_razafimbahoaka	28
tsd:Adam_ElHorri	28
tsd:g_d	28
tsd:tsd:Lord_laurent	28
tsd:elliott_mazerand	3
tsd:Lord_laurent	28
tsd:otya_kikadidi-zoula	28
tsd:Adam_Elhorri	28
tsd:Ibrahim_Simpara	28
tsd:mohamed_dahmoun	28
tsd:rafik_amara	28
tsd:Loreena_Barrere	28
tsd:laurencia_dovi-late	28
tsd:sebastienCortexpyramide	3
tsd:Bomboclaat	28
tsd:ayman_benazzouz	28
tsd:amaury_rebillard	28
tsd:Tai_Nguyen	28
tsd:chaymae_chouari	28
tsd:jules_cassan	28
tsd:Tom	28
tsd:loris_gp	3
tsd:ibrahim_simpara	28
tsd:romain_campillo	28
tsd:alice_dupont	28
tsd:Wemby	28
tsd:Thomas_Geoffroy	28
tsd:g_n	28
tsd:rayan_malandain	28

Figure 4 : Résultat de la requête de connexité socialFriend.

### 3.2.4 Donner la liste de vos voisins dans le graphe

```
SELECT DISTINCT (?o as ?voisins)
WHERE {tsd :romain_gallerne
(tsd :personalFriend|^ tsd :personalFriend) ?o.}
```

```
SELECT DISTINCT (?o as ?voisins)
WHERE {tsd :romain_gallerne
(tsd :socialFriend|^ tsd :socialFriend) ?o.}
```

### 3.2.5 Calculez le degré de chaque noeud (nombre de voisins)

```
SELECT ?s (count( ?o) as ?nbVoisins)
WHERE { ?s (tsd :personalFriend|^ tsd :personalFriend) ?o.
FILTER ( ?s!= ?o)} GROUP BY ?s
```

s	nbVoisins
tsd:charles_langlois	1
tsd:marta_boshkovska	2
tsd:sebastien_vial	3
tsd:thomas_razafimbahoaka	3
tsd:romain_gallerne	1
tsd:Adam_ElHorri	6
tsd:g_d	1
tsd:Tom_Gimat	1
tsd:Lord_laurent	4
tsd:elliott_mazerand	1
tsd:otya_kikadidi-zoula	11
tsd:Adam_Elhorri	1
tsd:Ibrahim_Simpara	4
tsd:mohamed_dahmoun	2
tsd:rafik_amara	7
tsd:Loreena_Barrere	9
tsd:laurencia_dovi-late	2
tsd:ayman_benazzouz	3
tsd:Bamboclaat	1
tsd:amaury_rebillard	2
tsd:Tai_Nguyen	7
tsd:chaymae_chouari	2
tsd:jules_cassan	7
tsd:loris_gp	3
tsd:romain_campillo	1
tsd:alice_dupont	2
tsd:Bamboclaat	1
tsd:Thomas_Geoffroy	10
tsd:loris_gpExplorer2	1
tsd:rayan_malandain	5

Figure 5 : Résultat de la requête du degré de chaque noeud (personalFriend).

```
SELECT ?s (count( ?o) as ?nbVoisins)
WHERE { ?s (tsd :socialFriend|^ tsd :socialFriend) ?o.
FILTER ( ?s!= ?o)} GROUP BY ?s
```

s	nbVoisins
tsd:charles_langlois	2
tsd:Ibrahim_simpara	1
tsd:marta_boshkovska	11
tsd:sebastien_vial	1
tsd:thomas_razafimbahoaka	8
tsd:Adam_ElHorri	10
tsd:g_d	1
tsd:tsd:Lord_laurent	1
tsd:Lord_laurent	3
tsd:elliott_mazerand	3
tsd:otya_kikadidi-zoula	13
tsd:Adam_Elhorri	1
tsd:mohamed_dahmoun	2
tsd:Ibrahim_Simpara	8
tsd:rafik_amara	9
tsd:Loreena_Barrere	14
tsd:laurencia_dovi-late	4
tsd:sebastienCortexpyramide	1
tsd:Bomboclaat	8
tsd:ayman_benazzouz	3
tsd:amaury_rebillard	9
tsd:Tai_Nguyen	10
tsd:chaymae_chouari	5
tsd:jules_cassan	8
tsd:Tom	8
tsd:loris_gp	1
tsd:romain_campillo	3
tsd:ibrahim_simpara	1
tsd:alice_dupont	1
tsd:Wemby	7
tsd:Thomas_Geoffroy	14
tsd:g_n	2
tsd:rayan_malandain	5

Figure 6 : Résultat de la requête du degré de chaque noeud (socialFriend).

### 3.2.6 Combien de triangles y-a-t il dans le graphe ?

```
SELECT (count(*) as ?nbTriangle) WHERE
{ ?s1 tsd :personalFriend ?s2. ?s2 tsd :personalFriend ?s3.
?s3 tsd :personalFriend ?s1. FILTER(STR(?s1) < STR(?s2))
FILTER(STR(?s1) < STR(?s3)) FILTER(?s2 != ?s3)}
```

Grâce à cette requête, on remarque qu'il y a 10 triangles dans le graphe des personal-Friend.

```
SELECT (count(*) as ?nbTriangle) WHERE
{ ?s1 tsd :socialFriend ?s2. ?s2 tsd :socialFriend ?s3.
?s3 tsd :socialFriend ?s1. FILTER(STR(?s1) < STR(?s2))
FILTER(STR(?s1) < STR(?s3)) FILTER(?s2 != ?s3)}
```

Grâce à cette requête, on remarque qu'il y a 22 triangles dans ce graphe.



### 3.2.7 Combien de triangles peuvent encore se former dans le graphe ?

```
SELECT (count(*) as ?nbTriangle) WHERE
{ ?s1 tsd :personalFriend ?s2. ?s2 tsd :socialFriend ?s3.
  FILTER(STR( ?s1) < STR( ?s2))
  FILTER(STR( ?s1) < STR( ?s3)) FILTER( ?s2 != ?s3)}
```

Grâce à cette requête, on remarque qu'il y a  $48-10 = 38$  triangles à former dans le graphe des personalFriend.

```
SELECT (count(*) as ?nbTriangle) WHERE
{ ?s1 tsd :socialFriend ?s2. ?s2 tsd :socialFriend ?s3.
  FILTER(STR( ?s1) < STR( ?s2))
  FILTER(STR( ?s1) < STR( ?s3)) FILTER( ?s2 != ?s3)}
```

Grâce à cette requête, on remarque qu'il y a  $100-22 = 78$  triangles à former dans ce graphe.

### 3.2.8 Donner la liste de toutes les cliques de cardinalité 4 dans le graphe

Une clique de cardinalité 4 est une composante fortement connexe de taille 4. On cherche donc 4 éléments distincts tous reliés entre eux. Voici les deux requêtes SPARQL ainsi que leurs résultats :

```
SELECT ?p1 ?p2 ?p3 ?p4 WHERE{
  ?p1 tsd :personalFriend ?p2. ?p1 tsd :personalFriend ?p3.
  ?p1 tsd :personalFriend ?p4. ?p2 tsd :personalFriend ?p3.
  ?p2 tsd :personalFriend ?p4. ?p3 tsd :socialFriend ?p4.
  FILTER(STR( ?p1)<STR( ?p2)) FILTER(STR( ?p2)<STR( ?p3))
  FILTER(STR( ?p3)<STR( ?p4))}
```

```
SELECT ?p1 ?p2 ?p3 ?p4 WHERE{
  ?p1 tsd :socialFriend ?p2. ?p1 tsd :socialFriend ?p3.
  ?p1 tsd :socialFriend ?p4. ?p2 tsd :socialFriend ?p3.
  ?p2 tsd :socialFriend ?p4. ?p3 tsd :socialFriend ?p4.
  FILTER(STR( ?p1)<STR( ?p2)) FILTER(STR( ?p2)<STR( ?p3))
  FILTER(STR( ?p3)<STR( ?p4))}
```

p1	p2	p3	p4
tsd:Lord_laurent	tsd:Loreena_Barrere	tsd:Thomas_Geoffroy	tsd:otya_kikadidi-zoula
tsd:Loreena_Barrere	tsd:Thomas_Geoffroy	tsd:jules_cassan	tsd:otya_kikadidi-zoula
tsd:Tai_Nguyen	tsd:Thomas_Geoffroy	tsd:jules_cassan	tsd:otya_kikadidi-zoula

  

p1	p2	p3	p4
tsd:Bomboclaat	tsd:Loreena_Barrere	tsd:Tai_Nguyen	tsd:otya_kikadidi-zoula
tsd:Bomboclaat	tsd:Loreena_Barrere	tsd:Tai_Nguyen	tsd:Thomas_Geoffroy
tsd:Bomboclaat	tsd:Loreena_Barrere	tsd:Thomas_Geoffroy	tsd:otya_kikadidi-zoula
tsd:Lord_laurent	tsd:Loreena_Barrere	tsd:Thomas_Geoffroy	tsd:otya_kikadidi-zoula
tsd:Loreena_Barrere	tsd:Thomas_Geoffroy	tsd:jules_cassan	tsd:otya_kikadidi-zoula
tsd:Tai_Nguyen	tsd:Thomas_Geoffroy	tsd:jules_cassan	tsd:otya_kikadidi-zoula
tsd:Bomboclaat	tsd:Tai_Nguyen	tsd:Thomas_Geoffroy	tsd:otya_kikadidi-zoula
tsd:Loreena_Barrere	tsd:Tai_Nguyen	tsd:jules_cassan	tsd:otya_kikadidi-zoula
tsd:Loreena_Barrere	tsd:Tai_Nguyen	tsd:Thomas_Geoffroy	tsd:jules_cassan
tsd:Loreena_Barrere	tsd:Tai_Nguyen	tsd:Thomas_Geoffroy	tsd:otya_kikadidi-zoula

Figure 7 : Au dessus, le résultat pour la requête avec personalFriend et au dessous le résultat de la requête pour socialFriend.

### 3.2.9 Donner la taille de chaque composante connexe du graphe

Pour cette question, on va simplement compter le nombre d'éléments dans chaque composante connexe du graphe.

```
SELECT ?s (count( ?o) as ?nbConnexe)
WHERE { ?s (tsd :personalFriend|^ tsd :personalFriend)+ ?o.
FILTER ( ?s != ?o)} GROUP BY ?s
```

```
SELECT ?s (count( ?o) as ?nbConnexe)
WHERE { ?s (tsd :socialFriend|^ tsd :socialFriend)+ ?o.
FILTER ( ?s != ?o)} GROUP BY ?s
```

s	nbConnexe
tsd:charles_langlois	21
tsd:marta_boshkovska	1
tsd:sebastien_vial	3
tsd:thomas_razafimbahoaka	21
tsd:romain_gallerne	21
tsd:Adam_ElHorri	21
tsd:g_d	3
tsd:Tom_Gimat	21
tsd:elliott_mazerand	1
tsd:Lord_laurent	21
tsd:otya_kikadidi-zoula	21
tsd:Adam_Elhorri	21
tsd:Ibrahim_Simpara	21
tsd:mohamed_dahmoun	21
tsd:rafik_amara	21
tsd:Loreena_Barrere	21
tsd:laurencia_dovi-late	1
tsd:Bomboclaat	21
tsd:ayman_benazzouz	21
tsd:amaury_rebillard	21
tsd:Tai_Nguyen	21
tsd:chaymae_chouari	21
tsd:jules_cassan	21
tsd:loris_gp	3
tsd:romain_campillo	21
tsd:Bamboclaat	3
tsd:alice_dupont	21
tsd:Thomas_Geoffroy	21
tsd:loris_gpExplorer2	1
tsd:rayan_malandain	21

Figure 8 : Résultat de la requête de connexité personalFriend.

On remarque donc qu'il y a plusieurs composantes connexes dans le graphe : une à 22 sommets, une à deux sommets et une à quatre sommets.

s	nbConnexe
tsd:charles_langlois	28
tsd:Ibrahim_simpara	28
tsd:marta_boshkovska	28
tsd:sebastien_vial	3
tsd:thomas_razafimbahoaka	28
tsd:Adam_ElHorri	28
tsd:g_d	28
tsd:tsd:Lord_laurent	28
tsd:elliott_mazerand	3
tsd:Lord_laurent	28
tsd:otya_kikadidi-zoula	28
tsd:Adam_Elhorri	28
tsd:Ibrahim_Simpara	28
tsd:mohamed_dahmoun	28
tsd:rafik_amara	28
tsd:Loreena_Barrere	28
tsd:laurencia_dovi-late	28
tsd:sebastienCortexpyramide	3
tsd:Bomboclaat	28
tsd:ayman_benazzouz	28
tsd:amaury_rebillard	28
tsd:Tai_Nguyen	28
tsd:chaymae_chouari	28
tsd:jules_cassan	28
tsd:Tom	28
tsd:loris_gp	3
tsd:ibrahim_simpara	28
tsd:romain_campillo	28
tsd:alice_dupont	28
tsd:Wemby	28
tsd:Thomas_Geoffroy	28
tsd:g_n	28
tsd:rayan_malandain	28

Figure 9 : Résultat de la requête de connexité socialFriend.

Il y a plusieurs composantes connexe dans le graphe : une à 29 sommets et une à 4 sommets.