INF344 2019–2020

TP Programmer avec le web sémantique

Due date: Thursday 2 July 2020, 12:00
Requested files: dvptAvecRdflib.py (Download)
Nombre maximal de fichiers: 30
Type of work: Individual work

Objectif: aborder la manipulation de connaissances exprimées en RDF

Pour ce TP, vous allez utiliser diverses techniques d'accès et de manipulation de données RDF. Ce TP utilise Python.

Nous allons principalement nous intéresser au site http://dbpedia.org/sparql. Le but va être d'extraire des informations de ce point d'accès sparql et de manipuler ces données en Python.

A la fin du TP, ou, au plus tard le 28/6 à minuit, vous devrez avoir actualisé votre version du fichier dvptAvecRdflib.py sur le moodle (vérifiez ce que donne l'évaluation automatique). Les dépôts tardifs seront pénalisés.

ATTENTION:

lors des requêtes répétées à un même site et compte tenu du nombre que vous êtes, respectez un délai d'au moins 2 secondes entre 2 appels: pensez-y dans votre code, par exemple en utilisant la méthode sleep entre 2 requêtes ; le non-respect de cette règle risque d'entrainer le blocage de nos accès

Eléments fournis

- un modèle de code de base nommé dvptAvecRdflib.py (dans Moodle).

Tâches

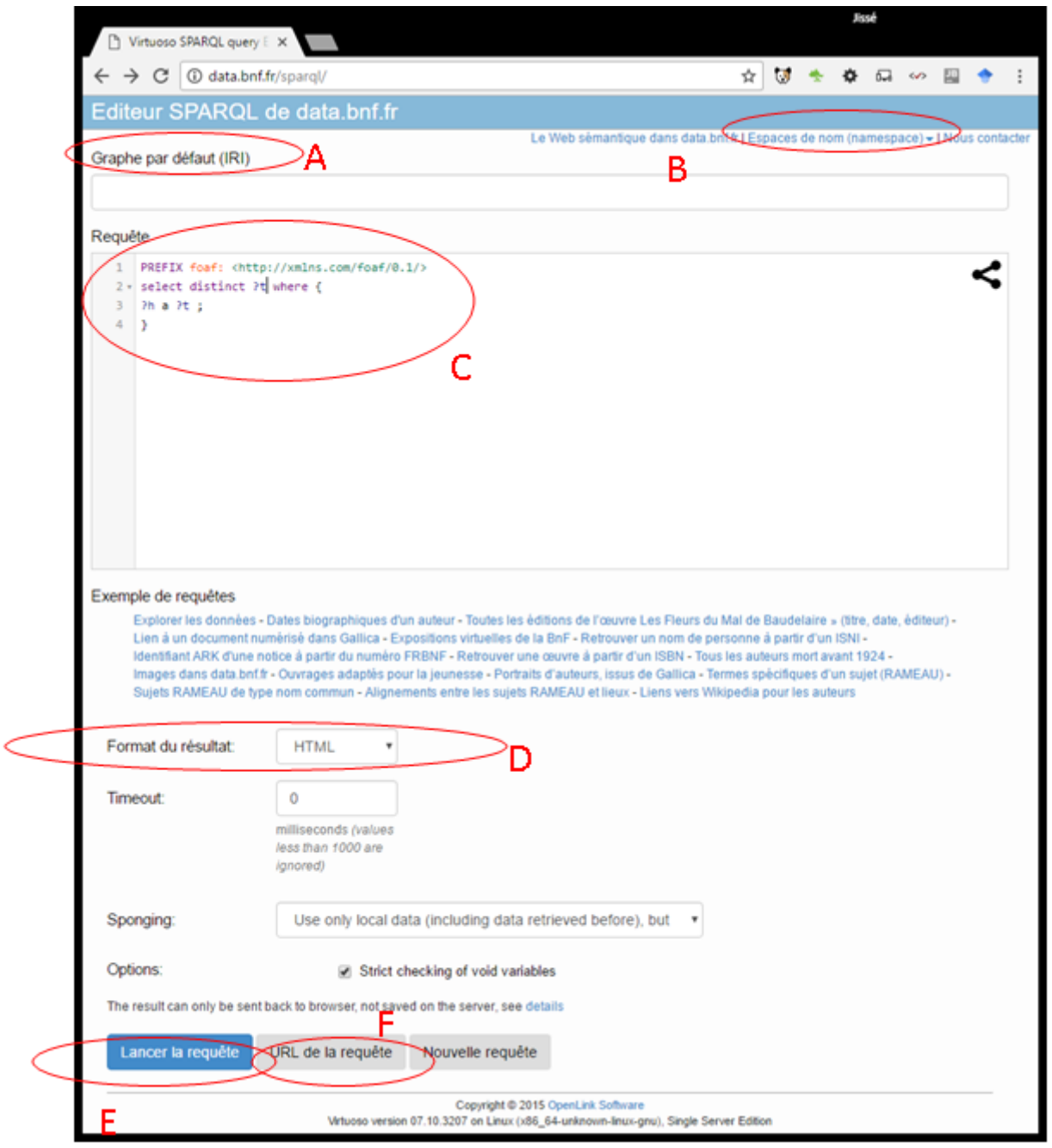
Modifiez le modèle dvptAvecRdflib.py en le complétant avec votre code. Il y a une méthode stepx à compléter par étape x décrite ci-après. L'étape 3 comporte une question dont la réponse est évaluée manuellement et vaut 2 points.
Vous pourrez vous familiariser interactivement avec l'accès aux données de DBpedia et tester interactivement des requêtes en vue de les intégrer dans votre programme Python. Nous allons aborder l'accès par programme –Python- à ces même données et différentes possibilités d'exploiter les données obtenues.

Découverte interactive du point d'accès SPARQL de DBPedia

Le point d'accès SPARQL de DBpedia est à l'adresse :

http://dbpedia.org/sparql

Il s'agit à la fois d'une page de formulaire qui vous permet de tester des requêtes SPARQL et aussi du point d'accès qui permet d'envoyer des requêtes depuis un programme. La version actuelle de l'interface peut être légérement différente de celle de la figure.



Dans la capture d'écran ci-dessus, les zones suivantes sont mises en évidence :

A La zone 'Graphe par défaut' permet d'indiquer un graphe de connaissances qui servira par défaut lorsque aucun graphe n'est indiqué dans la requête

B Ce lien permet d'afficher une liste de préfixes prédéfinis pour vous, afin de rendre les requêtes plus compactes et lisibles

C Zone d'édition des requêtes SPARQL proprement dite

D Cette liste donne les formats possibles pour le résultat des requêtes

E Exécution de la requête

F Devrait permettre la récupération de l'URL associée à la requête.

On voit en bas de la fenêtre que le graphe de connaissances de DBpedia est rendu accessible via le logiciel Virtuoso.

En cours de TP, vous pourrez utilement vous référer à:

SPARQL in 11 minutes:

La référence de SPARQL: https://www.w3.org/TR/sparql11-query/

La référence du module sparqlwrapper: https://rdflib.github.io/sparqlwrapper/

La documentation de rdflib: https://rdflib.readthedocs.io/en/stable/

Sauf indications contraires, nous recommandons le format JSON pour le résultat de sparqlwrapper.

Etape 1 Trouver les types de données présents dans dbpedia (notée 2 points)

Commencez par compléter la méthode step1 par du code.

Vous allez interroger http://dbpedia.org/sparql à l'aide de sparqlwrapper (module importé dans le modèle de code)
La requête:

select distinct ?t

from <http://dbpedia.org>

where { ?h a ?t }

limit 10

permet de trouver de sélectionner des triplets qui relient une entité à un type et d'obtenir comme résultats les 10 premiers types distincts.

(le prédicat ‘a’ est équivalent à ‘rdf:type’ ou en développant le préfixe http://www.w3.org/1999/02/22-rdf-syntax-ns#type, si comme c'est usuel le préfixe rdf: a été associé à la chaîne http://www.w3.org/1999/02/22-rdf-syntax-ns#)

La fin de step1 doit sauver dans un fichier texte de nom rdfsparql.step1 un objet dictionnaire python sérialisé en json contenant une clé "typelist" dont la valeur est la liste des 10 premiers types trouvés.

Ce principe d'objet json avec des clés sera utilisé pour sauver les résultats de toutes les étapes ultérieures.

Etape 2 Trouver des personnes, les compter, trouver les propriétés qui les décrivent (notée 4 points)

Nous allons nous intéresser aux entités de type foaf:Person.

Les résultats de cette étape seront introduites dans un dictionnaire python qui sera sauvé à la fin de l'étape.

Cherchez la valeur usuelle du préfixe foaf:, par exemple en consultant le site prefix.cc et donnez la version du type où le préfixe foaf: est remplacé par sa valeur.

foaf:Person. Enregistrez dans le dictionnaire la valeur développée du préfixe avec la clé "prefix".

Nous allons compter les personnes décrites dans dbpedia.

Pour cela, écrire une requête SPARQL qui commence par définir le préfixe foaf: , puis, en s'inspirant du modèle de la requête suivante

select (count(?e) as ?c) where { ?e ?p [] }

écrire une requête permettra de compter le nombre de personnes décrites dans DBpedia. Enregistrez la requête sous la clé "query" et le résultat sous la clé "personcount".

Pour les prochaines requêtes, pour alléger la lecture du sujet, nous omettrons les déclarations de préfixes ; vous pourrez, par exemple, chercher les préfixes sur le site prefix.cc, afin de les définir en tête de vos requêtes..

Nous allons maintenant chercher 10 uris de personnes décrites sur le modèle de la première requête qui serons enregistrés dans une table sous la clé "firstten" dans le dictionnaire, puis nous chercherons dix uris de personnes à partir de la centième, en exploitant le mot-clé OFFSET en plus du mot-clé LIMIT; ces dix-là seront enregistrés dans une table sous la clé "tenothers".

Dans l'interface interactive du point d'accès sparql de dbpedia, vous pouvez cliquer sur les URI obtenues pour être redirigé vers la description associée à cette uri en format HTML.

Notez l’url apparaissant dans la barre de navigation du navigateur et l'introduire dans le résultat avec la clé "urlhtml".

Sauvez le dictionnaire des résultats dans le fichier rdfsparql.step2.

Etape 3 Prédicats décrivant des personnes (5 points +2 pour la réponse textuelle)

Nous allons chercher les propriétés –prédicats- qui sont utilisées pour décrire des personnes dans dbpedia

Combien de prédicats sont utilisés ? On sauvera la requête sparql permettant d'obtenir ce nombre sous la clé "rqcount" et le nombre de prédicats sous la clé "predicatescount". Une requête avec le mot-clé count peut, dans certaines conditions, générer un timeout; on pourra contourner ce problème en récupérant tous les prédicats distincts en une ou plusieurs fois (avec LIMIT et OFFSET), en les rangeant dans une table et en regardant la taille de la table.

ATTENTION: certains d'entre vous ont rencontrés des problèmes sur ce compte; nous suggèrons que vous ne passiez pas trop de temps dessus; nous allons analyser la source de ce problème sporadique

24/6/2020: modification: pour éviter les problèmes de timeout, vous allez de voir faire l'hypothèse que les 1000 premières personnes sont représentatives des prédicats utilisés pour décrire les personnes et trouver les prédicats de ces milles personnes; pour cela, il est nécessaire

représentatives des prédicats utilisés pour décrire les personnes et trouver les prédicats de ces mêmes personnes, pour cela, il est nécessaire d'utiliser une requête imbriquée sur le modèle suivant

SELECT ... WHERE { {select ?pers where { ?pers a foaf:Person} limit 1000 } ?pers ... }

où les ... sont remplacés par ce qui vous permet de compter les prédicats associés aux personnes.

Quels prédicats sont utilisés? Enregistrez les 20 premiers trouvés sous la clé "predicates", la liste des prédicats obtenus et sous la clé rqpredicates la requête permettant d'obtenir cette liste.

Dans l'interface interactive, ou en copiant dans la barre de recherche d'un navigateur, cliquez sur le prédicat:

http://xmlns.com/foaf/0.1/familyName

En navigant dans la page web que vous obtenez, vous devez constater pourquoi il est intéressant de rendre une URI déréférençable ? (c'est-à-dire que d'une façon ou d'une autre, elle envoie sur une page web, ce qui n'est pas obligatoire)

Par curiosité, vous pouvez en consulter d'autres.

Maintenant, nous allons compter les utilisations de la propriété <http://dbpedia.org/ontology/deathPlace> par les objets de type foaf:Person. Pour cela, on comptera les entités qui utilisent cette propriété, sur le modèle suivant qu'on modifiera pour ne garder que les entités de type personne:

select (count(?s) as ?c) where

{ ?s <http://dbpedia.org/ontology/deathPlace> ?o }

Enregistrez dans le dictionnaire de résultat votre requête sous la clé "rqplacecount" et le résultat de la requête sous la clé "placecount".

Sous la clé "interpretation", mettez un bref texte donnant deux interprétations possibles de la comparaison entre le nombre de personnes trouvé précédemment ("personcount" à l'étape 2) et le nombre obtenu dans "placecount" (cette réponse ne fait pas partie de l'évaluation automatique).

Sauvez votre dictionnaire de résultat dans le fichier rdfsparql.step3.

Etape 4 Trouver une entité par son label et des entités liées (2 points)

Trouvez l'uri de l'entité de type foaf:Person et ayant un prédicat dont la valeur est "Jules Verne"@fr.

Enregistrez l'uri commençant par http dans la clé "julesverne" de votre dictionnaire de résultat.

Enregistrez le prédicat qui lie "Jules Verne" à cette uri dans la clé "jvpredicate" de votre dictionnaire de résultat.

Trouvez les uris des entités de type foaf:Document liées à l'entité précédente. Sauvez ces uris dans une liste sous la clé "jvdoc".

Etape 5 Obtenir la description d'une entité (5 points)

Avec l'uri trouvée à l'étape précédente, trouver quels prédicats lui sont associés? Enregistrez la liste des prédicats sous la clé "jvpredicates"

Nous allons maintenant récupérer des triplets descriptifs de cette uri.

Une requête cherchant tous les triplets ?pers ?p ?o, où ?pers serait l'uri correspondant à Jules Verne renverrait un contenu trop volumineux. On fera donc la récupération de tous ces triplets décrivant Jules Verne en faisant plusieurs appels avec LIMIT 100 et en jouant sur l'OFFSET.

Comptez les triplets trouvés et mettez ce décompte sous la clé "triplecount".

Injectez le résultat de cette requête dans un graphe créé avec rdflib. Dans un premier temps, on forcera les deux premiers éléments d'un triplet à être de type

```
URIRef et le troisième de type Literal (une gestion rigoureuse nécessite de distinguer les types possibles pour ce
troisième élément).
```

Sauvez ce graphe par sérialisation au format n3 (fonction serialize de rdflib) dans le fichier julesverne.n3

Sauvez le dictionnaire avec la clé jvpredicates dans le fichier rdfsparql.step5.

Nous savons maintenant récupérer dans un graphe rdflib local un ensemble de triplets décrivant l'entité liée à "Jules Verne".

Vous pourriez maintenant utiliser rdflib pour explorer cet ensemble de triplets.

Requested files

dvptAvecRdflib.py

```python
import isodate
from SPARQLWrapper import SPARQLWrapper, JSON, N3, TURTLE
from rdflib import Graph, URIRef, Literal
import json

class RdfDev:
    def __init__(self):
        """__init__
        Initialise la session de collecte
        :return: Object of class Collecte
        """
        # NE PAS MODIFIER
        self.basename = "rdfsparql.step"

    def rdfdev(self):
        """collectes
        Plusieurs étapes de collectes. VOTRE CODE VA VENIR CI-DESSOUS
        COMPLETER les méthodes stepX.
        """
        self.step1()
        self.step2()
        self.step3()
        self.step4()
        self.step5()

    def step1(self):
        stepfilename = self.basename+"1"
        result = { "typelist": []}
        # votre code ici
        with open(stepfilename, "w", encoding="utf-8") as resfile:
            json.dump(result, resfile)


    def step2(self):
        stepfilename = self.basename+"2"
        result = {}
        # votre code ici
        with open(stepfilename, "w", encoding="utf-8") as resfile:
            json.dump(result, resfile)

    def step3(self):
        stepfilename = self.basename+"3"
        result = {}
        # votre code ici
        with open(stepfilename, "w", encoding="utf-8") as resfile:
            json.dump(result, resfile)

    def step4(self):
        stepfilename = self.basename+"4"
        result = {}
        # votre code ici
        with open(stepfilename, "w", encoding="utf-8") as resfile:
            json.dump(result, resfile)

    def step5(self):
        stepfilename = self.basename+"5"
        result = {}
        # votre code ici
        with open(stepfilename, "w", encoding="utf-8") as resfile:
            json.dump(result, resfile)


if __name__ == "__main__":
    testeur = RdfDev()
    testeur.rdfdev()

```

VPL

◄ Explication du TP (+ rappels sur la désambiguation)

Aller à...

Programmer pour le web sémantique ►