Traitement Sémantique des données TD 1 : RDF

(TD en binômes ou monômes)

1 Le réseau des participants à l'UE

Notre premier exercice consiste en créer un graphe RDF décrivant les liens entre les participants au module. Le résultat sera mis à disposition et interrogeable lors des séances suivantes.

1) Saisissez vos données ici :

https://notes.inria.fr/LnHBdHhUQlG74P_65iAcjQ

en utilisant le format Turtle comme dans l'exemple suivant.

Note : vous pouvez mettre des données fictives si vous le souhaitez.

```
  # namespaces

@prefix tsd: <http://www.umontpellier.fr/traitementsemantiquedesdonnees#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

  # description d'un participant

tsd:prenom_nom   rdf:type   foaf:Person .
tsd:prenom_nom   foaf:mail   "prenom.nom@etud.umontpellier.fr" .
tsd:prenom_nom   foaf:name   "prénom" .
tsd:prenom_nom   foaf:familyname   "nom" .
```

2) Ensuite, référencez vos collègues à l'aide des propriétés suivantes :

```
  # liens

tsd:prenom_nom   tsd:socialFriend   tsd:prenom1_nom1 .
tsd:prenom_nom   tsd:personalFriend   tsd:prenom2_nom2 .
```

- `tsd:personalFriend` indique pour une personne dont vous avez le numéro de portable
- `tsd:socialFriend` indique une personne avec laquelle vous êtes amis dans un réseau social (e.g. Facebook) ou professionnel (e.g. LinkedIn).
- Notez que les deux relations ne sont pas mutuellement exclusives, et peuvent être vraies en même temps.
- Enfin, elles sont définies comme des sous-propriétés de `foaf:knows`, qui est une relation symétrique.

2 Modélisation RDF

Pour chaque donnée (JSON, Relationnelle, Textuelle) fournie dans la suite, proposer un graphe RDF intégrant les informations présentes dans les les données. *On ne vous demande pas de spécifier la procédure d'intégration de ces données.* On vous demande juste de modéliser le graphe souhaité en fin d'intégration.

Voici les aspects importants à prendre en compte pour la résolution de l'exercice.

1. Respect de la syntaxe RDF.
2. Pertinence de la modélisation proposée au regard du problème d'intégration de données.
3. Correcte utilisation des préfixes, littéraux (avec datatypes), et de vocabulaires existants.

2.1 Données JSON

```
{
  "patientID": P12345,
  "name": "Emily",
  "device": "SmartRing",
  "heartRateBPM": "78",
  "stepsTakenToday": "8500"
}
```

```
{
  "patientID": "P12345",
  "name": "Emily",
  "devices": {
    "SmartWatch": {
      "heartRate": {
        "currentBPM": 78
      },
      "stepsTaken": {
        "today": 8500
      }
    }
  }
}
```

2.2 Données Relationnelles

PatientID	Name	Age	Condition
1	Alice	30	Hypertension
2	Bob	25	Allergy
3	Charlie	40	Diabetes

2.3 Données Textuelles

"Le patient Jean Dupont, identifié par JDP123, a été diagnostiqué avec un diabète de type 2 et de l'hypertension. Il a 58 ans. John est actuellement sous prescription de Metformine. Sa dernière lecture de pression sanguine enregistrée était de 140/90 mmHg, et son niveau de sucre dans le sang à jeun était de 180 mg/dL."

3 Jena : une bibliothèque Java pour RDF

Implémenter les modèles proposées dans l'exercice 2 en Jena.

Une classe d'exemple est proposée au lien suivant.

https://gitlab.etu.umontpellier.fr/p00000013857/jena-models

3.1 Courte Liste des Principales Méthodes du Modèle Jena

La documentation de Jena est disponilble ici https://jena.apache.org/documentation/.

La classe `Model` d'Apache Jena fournit plusieurs méthodes pour manipuler et interroger des données RDF, dont vous pouvez vous servir.

— `add(Statement s)` : Ajoute une déclaration au modèle.
— `add(Resource s, Property p, RDFNode o)` : Ajoute une déclaration avec sujet, prédicat et objet au modèle.
— `createResource()` : Crée une nouvelle ressource anonyme.
— `createResource(String uri)` : Crée une nouvelle ressource avec l'URI donnée.
— `createProperty(String uri)` : Crée une propriété avec l'URI donnée.
— `remove(Statement s)` : Supprime une déclaration du modèle.
— `listStatements()` : Retourne un itérateur sur toutes les déclarations du modèle.
— `listResourcesWithProperty(Property p)` : Liste toutes les ressources avec une propriété donnée.
— `getResource(String uri)` : Retourne une ressource avec l'URI donnée.
— `getProperty(String uri)` : Retourne une propriété avec l'URI donnée.
— `createLiteral(String v)` : Crée un littéral avec une chaîne de caractères.
— `createTypedLiteral(Object value)` : Crée un littéral typé avec la valeur donnée.
— `read(String url)` : Lit un modèle RDF à partir d'une URL.
— `read(InputStream in, String base)` : Lit un modèle RDF à partir d'un flux d'entrée.
— `write(OutputStream out)` : Écrit le modèle dans un flux de sortie au format RDF/XML.
— `write(OutputStream out, String lang)` : Écrit le modèle dans un flux de sortie dans un format spécifié.
— `query(String queryString)` : Exécute une requête SPARQL sur le modèle.
— `createResource(Class<?> view)` : Crée une ressource et la retourne sous forme d'interface de vue.
— `contains(Resource s, Property p, RDFNode o)` : Vérifie si le modèle contient une déclaration spécifique.
— `setNsPrefix(String prefix, String uri)` : Définit un préfixe d'espace de noms dans le modèle.