Université Paris Cité **Bases de données spécialisées**
M2 Info et Maths-Info **Année 2025-2026**

TP - Séance 2 b

RDF, CSV, SPARQL et Intégration de données RDF

1 Conversion de CSV en RDF

Un des outils les plus utilisés pour convertir des données CSV en RDF est **Tarql** (un outil interne à Jena, appelé `jena-csv`, existait jusqu'à la version 3.9 mais il a été retiré dans la version 3.10).

Téléchargez la dernière version de Tarql ici (sans sources) : `https://github.com/tarql/tarql/releases`. Décompressez l'archive dans un repertoire `Tarql` et ajoutez le chemin vers `Tarql/bin` à la variable d'environnement PATH.

Tarql est un outil très flexible car il permet d'interroger des données CSV directement avec SPARQL. Par exemple si le fichier `test.csv` contient :

```
id,name,quantity,description,available
1,widget,3,for framing the blivets,false
2,blivet,2,needed for widgets,true
3,"like, wow",4,testing the CSV parsing,true
```

Tarql permet d'executer la requête SPARQL suivante directement sur ces données CSV :

```
# test.rq
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
SELECT ?name ?quantity ?available
WHERE {FILTER(xsd:boolean(?available) = true)}
```

Cela s'effectue grâce à la commande :

```
tarql test.rq test.csv
```

qui donnera le résultat :

```
--------------------------------------
| name        | quantity | available |
======================================
| "blivet"    | "2"      | "true"    |
| "like, wow" | "4"      | "true"    |
--------------------------------------
```

Utiliser SPARQL pour interroger une table CSV permet en particulier de générer du RDF en utilisant les requêtes de type `CONSTRUCT`. Cela permet notamment de convertir une table CSV en RDF, selon la règle de conversion désirée. Par exemple la requête suivante :

```
# test1.rq
base <http://example.com/id/>
prefix ex:<http://example.com/>
prefix xsd:<http://www.w3.org/2001/XMLSchema#>
```

```
construct {
  ?iri  ex:id ?i ;
        ex:name ?n ;
        ex:quantity ?q ;
        ex:available ?a .
}
where {
  bind(iri(?id) as ?iri)
  bind(xsd:integer(?id) as ?i)
  bind(xsd:string(?name) as ?n)
  bind(xsd:integer(?quantity) as ?q)
  bind(xsd:boolean(?available) as ?a)
}
```

exécutée en Tarql sur les données `test.csv`, renvoie l'ensemble de triplets RDF :

```
<http://example.com/id/1>
        ex:id         1 ;
        ex:name       "widget" ;
        ex:quantity   3 ;
        ex:available  false .

<http://example.com/id/2>
        ex:id         2 ;
        ex:name       "blivet" ;
        ex:quantity   2 ;
        ex:available  true .

<http://example.com/id/3>
        ex:id         3 ;
        ex:name       "like, wow" ;
        ex:quantity   4 ;
        ex:available  true .
```

Pour identifier chaque ligne de la table comme une ressource RDF, nous avons utilisé la fonction SPARQL `iri()` qui génère un IRI à partir de l'URI de base (`BASE`) et du paramètre. En particulier nous avons utilisé l'id de ligne pour rendre les IRI de ces ressources uniques.

Une autre option (surtout quand un `id` n'est pas disponible dans le CSV) est d'utiliser la fonction SPARQL `UUID()` qui génère à chaque invocation un IRI universellement unique.

Encore une autre possibilité est de ne pas donner d'IRI aux ressources RDF qui représentent les lignes : ces ressources peuvent être des noeuds anonymes. Un nouveau noeud anonyme peut être généré en SPARQL par la fonction `bnode()`.

Consultez la documentation Tarql pour plus d'informations sur son fonctionnement.

Exercice 1 Le fichier `mosaicBookStore.csv` décrit les livres en stock dans la librairie MOSAIC Bookstore. Ces données sont pour le moment stockées dans un tableur, il s'agit de les représenter en utilisant le modèle de données RDF. En s'inspirant de l'exemple de traduction ci-dessus, écrire une requête SPARQL pour la conversion de ces données en RDF, l'executer en Tarql et sauvegarder le résultat en format Turtle dans le fichier `mosaicBookStore.ttl`.

Les données RDF générées doivent utiliser le namespace `mbr: <http://mosaicBookstore.com/mbr/>`. Elles doivent contenir une ressource pour chaque livre (dont l'URI dans le namespace `mbr` peut être construit à partir de l'ISBN par exemple), ainsi qu'une ressource pour chaque auteur (dont l'URI dans le namespace `mbr` eut être généré en utilisant le nom et le prénom de l'auteur).

2 Exploration de DBPedia

Exercice 2 L'url `http://dbpedia.org/sparql` correspond à un point d'accès SPARQL (SPARQL endpoint) aux données de DBPedia. Vous allez utiliser celui-ci pour explorer les données RDF de DBpedia.

Une des difficultés (qui est aussi un avantage) de travailler avec les données RDF est que la structure de données n'est pas documentée séparément (comme c'est le cas dans les bases de données relationnelles), mais on peut la découvrir en interrogeant les données mêmes.

Supposez par exemple de vouloir rechercher dans DBPedia des informations relatives aux écrivains américains. On peut suivre la démarche suivante :

- Commencez par chercher un écrivain particulier, par exemple "Paul Auster". En DBPedia il y a une façon simple de repérer l'URI d'une ressource : cherchez Paul Auster dans Wikipedia ; on le trouve à l'adresse `https://fr.wikipedia.org/wiki/Paul_Auster`, cela nous permet de déduire que l'URI de la ressource associée à Paul Auster en DBPedia est `http://dbpedia.org/resource/Paul_Auster`. Notez que le préfixe `dbr:<http://dbpedia.org/resource/>` peut être utilisé dans l'éditeur de requêtes de DBPedia sans devoir le declarer.
- Dans DBPedia quelles sont les propriétés de l'écrivain Paul Auster ? Ecrivez une requête SPARQL permettant d'obtenir la liste des URI de ces propriétés, ordonnée par ordre alphabétique.

  En analysant le résultat notez quelles propriétés nous permettent de savoir quand et où Paul Auster est né.
- Ecrivez une requête SPARQL permettant de savoir quand et où Paul Auster est né.
- Ecrivez une requête SPARQL permettant de savoir si Paul Auster est mort.
- Ecrivez une requête permettant de trouver les classes dont Paul Auster est instance. Parmi celles-ci, laquelle définie dans l'ontologie DBpedia (préfixe `dbo: <http://dbpedia.org/ontology/>`) vous semble la plus appropriée pour déterminer que Paul Auster est un écrivain ?
- En utilisant cette classe, écrivez une requête permettant de déterminer le nombre d'écrivains présents dans DBpedia. Combien en avez-vous trouvé ? (hint : utilisez la fonction d'agrégation COUNT)
- Modifiez la requête précédente pour ne retrouver que les écrivains nés aux Etats-Unis. Combien en avez-vous trouvé ?
- Écrire une requête SPARQL qui permet de retrouver, pour chaque écrivain Américain, son URI et son nom foaf (`foaf:name`) s'il existe.

3 Créer les liens entre vos données et DBpedia

Nous allons maintenant relier le jeu de données RDF de la librairie MOSAIC Bookstore avec DBPedia.

Pour ce faire il faut enrichir les données RDF de la librairie MOSAIC Bookstore afin de rajouter des triplets permettant de créer un lien avec les ressources DBpedia. Par exemple, si dans votre jeu de données Paul Auster est identifié par l'URI `mbr:PaulAuster`, le triplet permettant de le lier à sa représentation dans DBpedia sera :

```
mbr:PaulAuster owl:sameAs dbr:Paul_Auster
```

Le prefixe `owl: <http://www.w3.org/2002/07/owl#>` identifie le schema de l'ontologie prédéfinie OWL, plus expressive que RDFS.

Écrire une requête CONSTRUCT qui génère un fichier `mosaicLinked.ttl` contenant tous les triplets de la forme :

```
uri_auth owl:sameAs uri_auth_dbpedia
```

où `uri_auth` est l'URI d'un auteur de notre jeu de donnée `mosaicBookStore.ttl`, et `uri_auth_dbpedia` est l'URI du même auteur en DBPedia. Pour chercher l'auteur en DBPedia, utilisez SERVICE dans votre requête CONSTRUCT, qui devra etre donc une requete féderée. La recherche en DBPedia sera faite par `foaf:name`. Vous pouvez obtenir le `foaf:name` à chercher par concatenation du prénom et du nom disponibles dans `mosaicBookStore.ttl`, séparés d'un espace (cette concatenation donne dans la plupart des cas le `foaf:name` de l'auteur en DBPedia).

Exécutez la requete CONSTRUCT depuis l'outil `sparql` de Jena pour générer les triplets `owl:sameAs` dans un fichier `mosaicLinked.ttl`

Ecrivez ensuite une requête fédérée qui calcule pour chaque livre la date de naissance de son auteur. Pour ce faire il faut interroger `mosaicBookStore.ttl`, `mosaicLinked.ttl` et DBPedia.