Rapport de Projet : Knowledge Graphs et Raisonnement RDFS

Romain Groult & Alban Talagrand

Janvier 2025

Instructions d'exécution

Pour reproduire l'intégration complète du projet :

Prérequis : Java 11+, Maven

Étape 1 : Récupération des données

Télécharger les 3 datasets Kaggle et les placer dans les répertoires appropriés :

- Netflix : https ://www.kaggle.com/datasets/luiscorter/netflix-original-films-imdb-scores → `data/csv/convert-netflix/netflix.csv`
- Amazon : https ://www.kaggle.com/datasets/muhammadawaistayyab/amazon-movies-and-films → `data/csv/convert-amazon/amazon.csv`
- IMDB : https ://www.kaggle.com/datasets/mazenramadan/imdb-most-popular-films-and-series → `data/csv/convert-imdb/imdb.csv`
  - **IMPORTANT** : Sélectionner uniquement les 14 premières colonnes (name, date, rate, votes, genre, duration, type, certificate, episodes, nudity, violence, profanity, alcohol, frightening)

Étape 2 : Compilation du projet

```
mvn clean package
```

Étape 3 : Génération et intégration automatique

```
cd scripts
./generate-ttl.sh
```

Ce script effectue automatiquement la conversion CSV → TTL, l'intégration avec création des liens `owl:sameAs`, et la fusion avec l'ontologie enrichie.

Étape 4 : Exécution des requêtes SPARQL

```
cd scripts
./run-queries.sh
```

Les résultats sont sauvegardés dans `result/resultats_requetes.txt`.

Pour exécuter une requête individuelle :

```
java -jar target/rdfs-generator.jar query queries/aggregation-request.sparql
 ↪  data/final_file.ttl
```

Table des matières

---

1. Jeux de données et extraction

1.1. Sources de données sélectionnées

Nous avons choisi trois jeux de données CSV provenant de Kaggle, tous liés au films :

1. **Netflix Original Films** (https ://www.kaggle.com/datasets/luiscorter/netflix-original-films-imdb-scores)
   — Films originaux Netflix avec notes IMDB
   — 6 attributs : titre, genre, date de première, durée, note IMDB, langue
   — 586 entrées
2. **Amazon Movies and Films** (https ://www.kaggle.com/datasets/muhammadawaistayyab/amazon-movies-and-films)
   — Catalogue de films Amazon
   — 10 attributs : ID, titre, note, nombre de notes, format, année de sortie, classification MPAA, réalisateur, acteurs, prix
   — 2110 entrées
3. **IMDB Most Popular Films** (https ://www.kaggle.com/datasets/mazenramadan/imdb-most-popular-films-and-series)
   — Films et séries populaires IMDB
   — 15 attributs : titre, date, note, votes, genre, durée, type, classification, épisodes, avertissements de contenu (nudité, violence, profanité, alcool, contenu effrayant)
   — 6179 entrées

1.2. Choix de la méthodologie d'extraction

Nous avons initialement commencé par vouloir utiliser des dumps RDF et des endpoints SPARQL (DBpedia, Wikidata). Cependant, nous avons rencontré deux problèmes majeurs :

— **Incohérence des données** : Les schémas RDF existants étaient trop différents, avec des propriétés et structures qui variaient entre les sources
— **Complexité des ontologies** : Les ontologies DBpedia et Wikidata sont extrêmement riches mais difficiles à manipuler pour un projet simple

Nous avons donc opté pour des **fichiers CSV Kaggle** qui nous permettent un contrôle total sur la modélisation RDF et l'alignement des schémas.

1.3. Processus d'extraction

L'extraction est réalisée par le module Java `CsvReader.java` qui :

— Parse les fichiers CSV avec gestion des délimiteurs et des guillemets

— Convertit les valeurs en types Java appropriés (String, Integer, Double, LocalDate, List)
— Applique des transformations via annotations :
  — `@Column` : mapping entre noms de colonnes CSV et champs Java
  — `@Ignore` : exclusion de valeurs spécifiques (ex : “No Rate”)
  — `@Remove` : suppression de patterns regex (ex : virgules dans les nombres)

Exemple pour Netflix (`NetflixFilm.java`) :

```java
@Column(name = "runtime")
private Integer duration;

@Column(name = "imdb score")
private Double rating;
```

Note importante : Les classes modèles (NetflixFilm, AmazonFilm, ImdbFilm) ne sont **pas compilées** avec le projet. Elles sont stockées sous forme de fichiers `.java` dans `data/csv/convert-*/` et sont **chargées dynamiquement à la volée** lors de l'exécution via le module `ClassLoader.java`. Ce mécanisme permet de :

— Modifier facilement le schéma des données sans recompiler le projet
— Ajouter de nouvelles sources de données en créant simplement un nouveau fichier `.java`
— Garder une architecture flexible et extensible

---

2. Intégration des données

2.1. Conversion en RDF/Turtle

Chaque jeu de données CSV est converti en fichier TTL par le module `TtlWriter.java` qui génère automatiquement :

Classes RDFS :

```
netflix:NetflixFilm rdf:type rdfs:Class .
amazon:AmazonFilm rdf:type rdfs:Class .
imdb:ImdbFilm rdf:type rdfs:Class .
```

Propriétés avec domaines et ranges appropriés :

```
netflix:title rdf:type rdf:Property ;
    rdfs:domain netflix:NetflixFilm ;
    rdfs:range rdfs:Literal .

netflix:duration rdf:type rdf:Property ;
    rdfs:domain netflix:NetflixFilm ;
    rdfs:range xsd:integer .
```

Instances pour chaque film :

```
netflix:The_Irishman rdf:type netflix:NetflixFilm ;
    netflix:title "The Irishman" ;
    netflix:genre "Crime" ;
```

```
    netflix:duration 209 ;
    netflix:rating 7.8 .
```

2.2. Alignement et liens owl :sameAs

L'intégration des trois sources est réalisée par `DataIntegrator.java` qui implémente un **algorithme de résolution d'entités** basé sur la distance d'édition :

Processus : 1. Normalisation des titres (minuscules, suppression des caractères spéciaux) 2. Calcul de la distance de Levenshtein entre tous les titres inter-sources 3. Création de liens `owl:sameAs` si similarité $> 85\%$

Exemple de liens générés :

```
netflix:The_Dark_Knight owl:sameAs amazon:The_Dark_Knight .
netflix:The_Dark_Knight owl:sameAs imdb:The_Dark_Knight .
amazon:The_Dark_Knight owl:sameAs imdb:The_Dark_Knight .
```

Cette approche permet d'identifier automatiquement les films présents dans plusieurs catalogues sans intervention manuelle.

2.3. Unification automatique des propriétés

Le processus d'unification analyse les propriétés de chaque classe et identifie celles qui sont **sémantiquement équivalentes** malgré des noms différents :

Méthode (implémentée dans `DataIntegrator.java:160-220`) : 1. Extraction de toutes les propriétés associées aux classes `NetflixFilm`, `AmazonFilm`, `ImdbFilm` 2. Regroupement par domaine (chaque propriété doit avoir le même domaine de classe) 3. Création de propriétés unifiées `unified:*` pour les propriétés communes 4. Génération des relations `rdfs:subPropertyOf`

Exemple d'unification :

```
# Propriétés sources différentes mais sémantiquement identiques
netflix:duration → unified:duration
imdb:duration → unified:duration

# Relation de hiérarchie générée automatiquement
netflix:duration rdfs:subPropertyOf unified:duration .
imdb:duration rdfs:subPropertyOf unified:duration .
```

Propriétés unifiées créées automatiquement :

- `unified:title` ← {netflix :title, amazon :title, imdb :title}
- `unified:rating` ← {netflix :rating, amazon :rating, imdb :rating}
- `unified:genre` ← {netflix :genre, imdb :genre}
- `unified:duration` ← {netflix :duration, imdb :duration}

2.4. Hiérarchie de classes

Une classe parent `unified:film` est créée avec des sous-classes pour chaque source :

```
unified:film rdf:type rdfs:Class .
```

```
netflix:NetflixFilm rdfs:subClassOf unified:film .
amazon:AmazonFilm rdfs:subClassOf unified:film .
imdb:ImdbFilm rdfs:subClassOf unified:film .
```

Cette hiérarchie permet d'interroger tous les films via la classe parente tout en conservant la traçabilité de la source.

2.5. Ontologie enrichie manuelle

En complément de l'unification automatique, nous avons créé `to_add_ontology.ttl` avec des ajouts de règles RDFS manuelles :

Super-propriété pour avertissements de contenu :

```
unified:contentAdvisory rdf:type rdf:Property ;
    rdfs:domain unified:film ;
    rdfs:comment "Avertissements sur le contenu du film" .

imdb:violence rdfs:subPropertyOf unified:contentAdvisory .
imdb:nudity rdfs:subPropertyOf unified:contentAdvisory .
imdb:profanity rdfs:subPropertyOf unified:contentAdvisory .
imdb:alcohol rdfs:subPropertyOf unified:contentAdvisory .
imdb:frightening rdfs:subPropertyOf unified:contentAdvisory .
```

Cela permet de requêter tous les avertissements via une seule propriété parente.

2.6. Pipeline d'intégration

Le script `generate-ttl.sh` automatise tout le processus :

```
java -jar rdfs-generator.jar csv data/csv/imdb.csv ImdbFilm.java imdb-csv.ttl
java -jar rdfs-generator.jar csv data/csv/amazon.csv AmazonFilm.java
 ↪  amazon-csv.ttl
java -jar rdfs-generator.jar csv data/csv/netflix.csv NetflixFilm.java
 ↪  netflix-csv.ttl

java -jar rdfs-generator.jar integrate integrated.ttl 3 \
  imdb imdb-csv.ttl \
  amazon amazon-csv.ttl \
  netflix netflix-csv.ttl \
  title film NetflixFilm AmazonFilm ImdbFilm

java -jar rdfs-generator.jar merge integrated.ttl to_add_ontology.ttl
 ↪  final_file.ttl
```

---

3. Requêtes SPARQL

En tout nous avons développé 9 requêtes SPARQL.

3.1. Requête avec agrégation

Fichier : `aggregation-request.sparql`

Objectif : Statistiques par genre avec moyenne de durée et de note

```
PREFIX unified: <http://example.org/unified/>

SELECT ?genre
       (COUNT(DISTINCT ?film) as ?totalFilms)
       (ROUND(AVG(?duration)) as ?avgDuration)
       (ROUND(AVG(?rating)) as ?avgRating)
       (MIN(?rating) as ?minRating)
       (MAX(?rating) as ?maxRating)
WHERE {
  ?film a unified:film .
  ?film unified:genre ?genre .
  ?film unified:rating ?rating .
  ?film unified:duration ?duration .
}
GROUP BY ?genre
HAVING (COUNT(DISTINCT ?film) > 5)
ORDER BY DESC(?totalFilms)
```

Résultat : Agrégation de données provenant des 3 sources grâce aux propriétés unifiées.

3.2. Requête avec OPTIONAL

Fichier : `optional-request.sparql`

Objectif : Récupérer tous les films même si certaines propriétés manquent

```
SELECT ?title ?rating ?duration ?genre ?date
WHERE {
  ?film a unified:film .
  ?film unified:title ?title .

  OPTIONAL { ?film unified:rating ?rating . }
  OPTIONAL { ?film unified:duration ?duration . }
  OPTIONAL { ?film unified:genre ?genre . }
  OPTIONAL { ?film unified:releaseDate ?date . }
}
LIMIT 50
```

Utilité : Les sources ont des schémas incomplets. OPTIONAL garantit que tous les films sont retournés même s'ils n'ont pas de note ou de genre.

3.3. Requête avec MINUS

Fichier : `minus-request.sparql`

Objectif : Films longs (>90 min) SAUF ceux de Netflix avec contenu modéré/sévère

```
PREFIX unified: <http://example.org/unified/>
PREFIX netflix: <http://example.org/netflix/>
PREFIX imdb: <http://example.org/imdb/>

SELECT ?title ?duration
WHERE {
  ?film a unified:film .
  ?film unified:title ?title .
  ?film unified:duration ?duration .

  FILTER (?duration > 90)

  MINUS {
    ?film a netflix:NetflixFilm .
    ?film imdb:violence ?violence .
    FILTER (REGEX(?violence, "Moderate|Severe", "i"))
  }
}
ORDER BY DESC(?duration)
LIMIT 30
```

3.4. Requête avec FILTER NOT EXISTS

Fichier : `not-exists-request.sparql`

Objectif : Alternative à MINUS, films sans avertissement de contenu

```
SELECT ?title
WHERE {
  ?film a unified:film .
  ?film unified:title ?title .

  FILTER NOT EXISTS {
    ?film unified:contentAdvisory ?advisory .
  }
}
LIMIT 20
```

3.5. Requête avec expressions de chemin RDFS

Fichier : `path-request.sparql`

Objectif : Démonstration de `rdfs:subPropertyOf*` pour inférence de propriétés

```
PREFIX unified: <http://example.org/unified/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT DISTINCT ?title ?ratingValue ?ratingSource
WHERE {
  ?film a unified:film .
```

```
  # Utilise subPropertyOf* pour trouver tous les titres
  ?titleProp rdfs:subPropertyOf* unified:title .
  ?film ?titleProp ?title .

  # Utilise subPropertyOf* pour trouver tous les ratings
  OPTIONAL {
    ?ratingProp rdfs:subPropertyOf* unified:rating .
    ?film ?ratingProp ?ratingValue .

    BIND(
      IF(?ratingProp = amazon:rating, "Amazon",
        IF(?ratingProp = netflix:rating, "Netflix", "Unknown"))
      AS ?ratingSource
    )
  }

  FILTER (?ratingSource != "Unknown" && BOUND(?ratingValue))
}
ORDER BY ?title
LIMIT 30
```

Puissance du raisonnement : Cette requête récupère automatiquement les titres et ratings des 3 sources sans les spécifier explicitement. L'opérateur `*` calcule la fermeture transitive de `rdfs:subPropertyOf`.

3.6. Requête sur hiérarchie de classes

Fichier : `class-hierarchy-request.sparql`

Objectif : Compter les films par source en utilisant `rdfs:subClassOf`

```
PREFIX unified: <http://example.org/unified/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?sourceClass (COUNT(?film) as ?count)
WHERE {
  ?film a unified:film .
  ?film a ?sourceClass .

  ?sourceClass rdfs:subClassOf unified:film .
}
GROUP BY ?sourceClass
ORDER BY DESC(?count)
```

Résultat : Distribution des films par catalogue (Netflix, Amazon, IMDB).

3.7. Requête fédérée

Fichier : `federated-request.sparql`

Objectif : Interroger Wikidata pour enrichir les résultats locaux

```
PREFIX unified: <http://example.org/unified/>
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>

SELECT ?localTitle ?wikidataTitle ?director ?releaseDate
WHERE {
  # Données locales
  ?localFilm a unified:film .
  ?localFilm unified:title ?localTitle .

  # Requête fédérée vers Wikidata
  SERVICE <https://query.wikidata.org/sparql> {
    ?wikidataFilm wdt:P31 wd:Q11424 ;  # instance of: film
                  wdt:P57 wd:Q25191 ;   # director: Christopher Nolan
                  rdfs:label ?wikidataTitle ;
                  wdt:P577 ?releaseDate .

    FILTER (LANG(?wikidataTitle) = "en")
  }

  # Lien entre les deux sources
  FILTER (REGEX(?localTitle, REPLACE(?wikidataTitle, " ", "_"), "i"))
}
LIMIT 10
```

Explication : Cette requête combine notre graphe local avec les données Wikidata pour récupérer des informations supplémentaires (réalisateur, date précise) sur les films de Christopher Nolan.

3.8. Requête complexe multi-sources

Fichier : `complex-request.sparql`

Objectif : Combiner explicitement les 3 sources avec filtres

```
SELECT ?title ?netflixRating ?amazonRating ?imdbRating
WHERE {
  ?film a unified:film .
  ?film unified:title ?title .

  OPTIONAL { ?film netflix:rating ?netflixRating . }
  OPTIONAL { ?film amazon:rating ?amazonRating . }
  OPTIONAL { ?film imdb:rating ?imdbRating . }

  FILTER (BOUND(?netflixRating) || BOUND(?amazonRating) || BOUND(?imdbRating))
}
LIMIT 50
```

3.9. Requête avec ontologie enrichie

Fichier : `enriched-ontology-request.sparql`

Objectif : Utiliser les propriétés unifiées manuelles

```
SELECT ?title ?advisory
WHERE {
  ?film a unified:film .
  ?film unified:title ?title .

  ?advisoryProp rdfs:subPropertyOf* unified:contentAdvisory .
  ?film ?advisoryProp ?advisory .
}
LIMIT 30
```

Résultat : Récupère tous les avertissements (violence, nudité, etc.) via la super-propriété `unified:contentAdvisory`.

3.10. Exécution des requêtes

Le module `QueryExecutor.java` charge les données, applique le raisonnement RDFS, puis exécute les requêtes :

```
Model baseModel = RDFDataMgr.loadModel(ttlFile);
InfModel infModel = ModelFactory.createRDFSModel(baseModel);  // ← Raisonnement
↪ RDFS
QueryExecution qexec = QueryExecutionFactory.create(query, infModel);
```

Script : `run-queries.sh` exécute les 9 requêtes et génère `result/resultats_requetes.txt`.

---

4. Raisonnement RDFS

4.1. Construction de l'ontologie

Notre ontologie RDFS (`to_add_ontology.ttl`) comprend :

Hiérarchie de classes :

```
unified:film rdf:type rdfs:Class .
netflix:NetflixFilm rdfs:subClassOf unified:film .
amazon:AmazonFilm rdfs:subClassOf unified:film .
imdb:ImdbFilm rdfs:subClassOf unified:film .
```

Hiérarchie de propriétés (exemple pour les titres) :

```
unified:title rdf:type rdf:Property ;
    rdfs:domain unified:film ;
    rdfs:range rdfs:Literal .

netflix:title rdfs:subPropertyOf unified:title .
```

```
amazon:title rdfs:subPropertyOf unified:title .
imdb:title rdfs:subPropertyOf unified:title .
```

Domaines et ranges :

```
unified:rating rdf:type rdf:Property ;
    rdfs:domain unified:film ;
    rdfs:range xsd:double ;
    rdfs:comment "Note du film (échelle variable selon la source)" .
```

Alignements inter-sources :

```
# Durée
netflix:duration rdfs:subPropertyOf unified:duration .
imdb:duration rdfs:subPropertyOf unified:duration .

# Date de sortie
netflix:premiere rdfs:subPropertyOf unified:releaseDate .
amazon:releaseYear rdfs:subPropertyOf unified:releaseDate .
imdb:date rdfs:subPropertyOf unified:releaseDate .

# Classification du contenu
amazon:mpaaRating rdfs:subPropertyOf unified:contentRating .
imdb:certificate rdfs:subPropertyOf unified:contentRating .
```

4.2. Mécanismes de raisonnement appliqués

Apache Jena applique automatiquement les **règles RDFS** suivantes :

Règle rdfs9 (Inférence de classe) :

```
Si : ?film rdf:type netflix:NetflixFilm
Et : netflix:NetflixFilm rdfs:subClassOf unified:film
Alors : ?film rdf:type unified:film  (inféré)
```

Règle rdfs7 (Inférence de propriété) :

```
Si : ?film netflix:title "Inception"
Et : netflix:title rdfs:subPropertyOf unified:title
Alors : ?film unified:title "Inception"  (inféré)
```

Règle rdfs11 (Transitivité de subClassOf) :

```
Si : ClasseA rdfs:subClassOf ClasseB
Et : ClasseB rdfs:subClassOf ClasseC
Alors : ClasseA rdfs:subClassOf ClasseC
```

4.3. Exemples concrets d'inférence

Cas 1 : Requête sur classe parente

Requête :

```
SELECT (COUNT(?film) as ?total)
WHERE { ?film a unified:film . }
```

Sans raisonnement : 0 résultats car aucune instance n'est directement typée `unified:film` **Avec raisonnement RDFS** : tous les films Netflix, Amazon, IMDB sont inférés comme `unified:film`

Cas 2 : Requête sur propriété parente

Requête :

```
SELECT ?title
WHERE { ?film unified:title ?title . }
```

Sans raisonnement : 0 résultats **Avec raisonnement RDFS** : Tous les titres des 3 sources (car `netflix:title`, `amazon:title`, `imdb:title` sont inférés comme instances de `unified:title`)

Cas 3 : Expression de chemin avec fermeture transitive

Requête :

```
?prop rdfs:subPropertyOf* unified:contentAdvisory .
?film ?prop ?value .
```

Résultat : Récupère toutes les propriétés descendantes de `contentAdvisory` :

- `imdb:violence`
- `imdb:nudity`
- `imdb:profanity`
- `imdb:alcohol`
- `imdb:frightening`

4.4. Comparaison données brutes vs. enrichies

Données brutes (avant raisonnement) :

```
netflix:Inception a netflix:NetflixFilm ;
    netflix:title "Inception" ;
    netflix:rating 8.8 .
```

Données enrichies (après raisonnement RDFS) :

```
netflix:Inception a netflix:NetflixFilm ;       # ← Original
    netflix:Inception a unified:film ;          # ← Inféré
    netflix:title "Inception" ;                 # ← Original
    unified:title "Inception" ;                 # ← Inféré
    netflix:rating 8.8 ;                        # ← Original
    unified:rating 8.8 .                        # ← Inféré
```

4.5. Requêtes démontrant le raisonnement

Requête 1 : Agrégation multi-sources transparente

```
SELECT (AVG(?rating) as ?avgRating)
WHERE {
  ?film a unified:film .
  ?film unified:rating ?rating .
}
```

Cette requête fonctionne **automatiquement** sur les 3 sources grâce à l'inférence RDFS.

Requête 2 : Navigation dans la hiérarchie de propriétés

```
SELECT ?film ?prop ?value
WHERE {
  ?film a unified:film .
  ?prop rdfs:subPropertyOf* unified:contentAdvisory .
  ?film ?prop ?value .
}
```

Récupère tous les avertissements de contenu via la super-propriété.

Requête 3 : Comptage par sous-classe

```
SELECT ?class (COUNT(?film) as ?count)
WHERE {
  ?film a unified:film .
  ?film a ?class .
  ?class rdfs:subClassOf unified:film .
}
GROUP BY ?class
```

Résultat : Distribution par source (Netflix : 120, Amazon : 90, IMDB : 150).

---

5. Conclusion

Le projet a permis d'intégrer trois sources de données différentes (Netflix, Amazon, IMDB) dans un graphe RDF unifié. L'utilisation du raisonnement RDFS a simplifié l'interrogation des données en permettant d'écrire des requêtes SPARQL sur des propriétés unifiées plutôt que de gérer les différences entre sources.

Les principales contributions techniques sont :

- Génération automatique de TTL depuis CSV avec chargement dynamique des classes modèles
- Alignement automatique des entités via `owl:sameAs`
- Unification des propriétés par analyse du domaine des classes (`rdfs:subPropertyOf`)
- 9 requêtes SPARQL démontrant agrégation, OPTIONAL, MINUS, expressions de chemin et fédération

Les limites rencontrées incluent la performance sur de gros volumes de données et l'absence de résolution automatique des conflits de valeurs entre sources. Une évolution vers OWL permettrait d'ajouter des contraintes de cardinalité et des relations d'équivalence plus riches.

Références

- Kaggle Netflix Dataset - https ://www.kaggle.com/datasets/luiscorter/netflix-original-films-imdb-scores
- Kaggle Amazon Dataset - https ://www.kaggle.com/datasets/muhammadawaistayyab/amazon-movies-and-films

— Kaggle IMDB Dataset - https ://www.kaggle.com/datasets/mazenramadan/imdb-most-popular-films-and-series
— Introduction au RDF et à l'API RDF de Jena - https ://web-semantique.developpez.com/tutoriels/jena/intro rdf/
— Apache Jena tutorial - write models in different formats with jena RDF,TURTLE,JSON - https ://www.youtube.com/watch ?v=Ps_cVNSwfeA