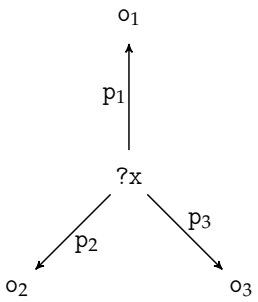
1 Implementation d'un mini moteur de requêtes RDF★ (RDF Star).

L'objectif du projet est d'implementer l'une des approches pour la persistence des données RDF vues en cours (hexastores, column-stores, graphes) ainsi que les procedures nécessaires pour l'évaluation de requêtes SPARQL de type "étoile".

Une requête SPARQL en étoile est composée de n patrons de triplet, tous partageant la même variable au sujet. Ces sont des requêtes de la forme `SELECT ?x WHERE { ?x` p_1 o_1 `. ?x` p_2 o_2 `... ?x` p_n o_n `}` dont la représentation graphique est la suivante.



Voici des interrogations possibles grâce aux requêtes étoile.

Q_1 : *donner les artistes qui sont en même temps des écrivains et des peintres*

Q_2 : *quels sont les artistes nés en 1904 ayant vécu à Paris ?*

Q_3 : *qui sont les amis d'Alice qui aiment le cinéma ?*

Les deux premières interrogations s'expriment par des requêtes étoile avec trois patrons de triplet. Pour Q_1, il suffit d'imposer p_1 `= type,` o_1 `= artiste,` o_2 `= peintre,` o_3 `= écrivain`. Similairement, pour Q_2, nous aurons p_1 `= type,` o_1 `= artiste,` p_2 `= lives_in,` p_3 `= birth_year,` o_2 `= Paris,` o_3 `= 1900`. Enfin, pour pour Q_3, nous aurons la requête `SELECT ?x WHERE {?x friendOf Alice . ?x likes Movies}`.

Bien entendu le système résultant du projet doit être capable de gérer des requêtes étoile avec n branches, et n'importe quelle valeur pour les constantes p_i et o_i.

L'évaluation des requêtes étoile constitue le socle pour n'importe quel système d'évaluation de requêtes SPARQL. Nous identifions cinque étapes fondamentales.

1. Chargement de la base de triplets et création d'un dictionnaire pour les ressources.
2. Creation des indexes.
3. Lecture des requêtes en entrée.
4. Choix de l'ordre d'évaluation des patrons de triplet.
5. Accès aux données et visualisation des résultats.

Nous allons maintenant détailler chaque étape.

Le dictionnaire (10 octobre)

Le dictionnaire associe un entier à chaque ressource de la base RDF, afin d'en permettre un stockage compact. Par exemple, on voit les triplets `<Bob, knows, Bob>` et `<1,2,1>` indistinctement avec la correspondence {(1,Bob) , (2,knows)}.

L'index (17 octobre)

L'index permet une évaluation efficace des requêtes et est adapté au système de persistence choisi. Dans l'implementation vous utiliserez des HasMap, des arrays, ou des listes d'adjacence. Clairement, chaque structure répond à un besoin particulier et a ses avantages et ses inconvénients (espace vs temps d'accès vs maintenance).

L'accès aux données (17/24 octobre)

1) L'accès aux données se fait par l'index. Une fois retrouvé l'ensemble des valeurs pour la variable `?x` pour le premier patron de triplet, cela nous servira pour filtrer ultérieurement les valeurs récupérées pour le deuxième patron, et ainsi de suite.

L'optimisation de requête (24 octobre)

Afin de choisir la meilleure stratégie d'évaluation d'une requête, le système doit prévoir des statistiques sur les données, permettant d'estimer la sélectivité des patrons de triplet de la requête. Il sera ainsi primordial de choisir un ordre d'évaluation des triplets qu'on estime minimiser le nombre de résultat intermédiaires (et donc le coût d'évaluation).

Travail démandé : implementer dictionnaire, indexation, optimisation, et accès aux données du moteur RDF★.

Consignes :

1. Le projet se fait en binômes (pas de monômes, ni de trinômes).
2. Nous choissons de representer nos données en mémoire vive (ram), mais rien nous empêche de prévoir un système de persistance en mémoire secondaire comme extension possible du travail.
3. Le langage de programmation est imposé : Java.
4. Pour la lecture des fichiers à partir du langage Java, vous trouverez un exemple dans l'ENT, ainsi que des données et des requêtes RDF pour tester votre prototype.
5. Il est également important de fournir un .jar exécutable en ligne de commande avec les options suivantes : en indiquant le répertoire contenant les documents à interroger, les requêtes à évaluer, et la sortie.

```
java -jar rdfstar
-queries "/chemin/vers/requetes"
-data "/chemin/vers/donnees"
-output "/chemin/vers/dossier/sortie"
-verbose
-export_results
-export_stats
-workload_time
```

6. Les temps d'executions doivent êtres visualisées dans le terminal si besoin (option `verbose`) et exportés dans un fichier `csv` contenu dans le répertoire indiqué (option `output`).
7. Ce répertoire contiendra également les statistiques sur les requêtes (option `export_stats`) et notamment l'ordre d'évaluation choisi et l'estimation de sélectivité de chaque requête.
8. Les résultats des requêtes doivent être exportés dans un fichier `cvs` séparé, si demandé (option `export_results`).
9. L'option `workload_time` sert à visualiser le temps total d'évaluation de l'*ensemble des requêtes*.

Extensions :

— Implémenter une procédure générique simple permettant d'évaluer n'importe quelle requête SPARQL.
— Utiliser Watdiv pour générer des requêtes de test adaptées au point précédent.