Examen MIF04 - Gestion de données pour le Web - session 1 - 18 janvier 2022
Durée : 2h
Documents autorisés
Le barème est donné à titre indicatif

Exercice 1: SPARQL vers SQL (6 points)

On considère une base de données relationnelle ayant le schéma correspondant aux définitions SQL suivantes :

```
1 CREATE TABLE triplet(
2     sujet TEXT,
3     predicat TEXT,
4     objet TEXT,
5     PRIMARY KEY (sujet, predicat, objet)
6 );
```

La table `triplet` est utilisée pour stocker le contenu d'un *store* RDF nommé $data_{RDF}$. Les ressources et les littéraux sont stockés sous forme de texte en utilisant la syntaxe N-Triple :

- Les ressources sont représentées par leur URI entre les symboles < et >, par exemple `<http://xmlns.com/foaf/0.1/knows>`.
- Les littéraux sont représentés sous forme d'une chaîne de caractère délimitée par le caractère " (ce qui est différent de la délimitation des chaînes de caractères en SQL qui utilise '). Dans cet exercice, on se limitera au ressources non typées (i.e. on utilisera pas de notation de spécification de type comme par exemple `^^xsd:integer`).

Question 1.1: Écrire une requête SQL permettant d'obtenir les résultats de la requête SPARQL suivante si elle avait été exécutée sur les données de $data_{RDF}$.

```
1   PREFIX foaf: <http://xmlns.com/foaf/0.1/> .
2
3   SELECT ?a WHERE {
4     ?a foaf:name "Toto" .
5   }
```

$$\text{(Domain)}\ \frac{S\ P\ O \qquad P\ \texttt{rdfs:domain}\ T}{S\ \texttt{rdf:type}\ T}$$

$$\text{(Range)}\ \frac{S\ P\ O \qquad P\ \texttt{rdfs:range}\ T}{O\ \texttt{rdf:type}\ T}$$

$$\text{(SC)}\ \frac{X\ \texttt{rdf:type}\ C \qquad C\ \texttt{rdfs:subClassOf}\ D}{X\ \texttt{rdf:type}\ D}$$

$$\text{(SP)}\ \frac{S\ P\ O \qquad P\ \texttt{rdfs:subPropertyOf}\ Q}{S\ Q\ O}$$

`rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#`
`rdfs: http://www.w3.org/2000/01/rdf-schema#`

FIGURE 1 – Quelques règles de déduction RDFS

Question 1.2: Ecrire une requête SQL permettant d'obtenir les résultats de la requête SPARQL suivante si elle avait été exécutée sur les données de $data_{RDF}$. Les noms des attributs dans le résultat de la requête SQL doivent correspondre au nom des variables de la requête SPARQL (sans le `?`).

```
1  PREFIX foaf: <http://xmlns.com/foaf/0.1/> .
2
3  SELECT ?a, ?b WHERE {
4    ?a foaf:knows ?c .
5    ?b foaf:knows ?c .
6    ?c foaf:name "Toto" .
7    FILTER (?a != ?b).
8  }
```

Question 1.3: Ecrire une requête SQL permettant d'obtenir les résultats de la requête SPARQL suivante si elle avait été exécutée sur les données de $data_{RDF}$. Si, dans un résutat, certaines variables ne sont pas liées, la requête SQL leur attribuera la valeur `NULL`. Les noms des attributs dans le résultat de la requête SQL doivent correspondre au nom des variables de la requête SPARQL (sans le `?`).

```
1  PREFIX foaf: <http://xmlns.com/foaf/0.1/> .
2
3  SELECT ?a, ?b, ?c WHERE {
4    ?a foaf:knows ?b .
5    OPTIONAL { ?b foaf:knows ?c . }
6  }
```

Question 1.4: Ecrire une requête SQL permettant d'obtenir les triplets déductibles en une étape à partir de $data_{RDF}$, en utilisant la règle (SP) de la figure 1. Le résultat de cette requête aura pour attributs `sujet`, `predicat` et `objet`. On ne souhaite pas obtenir les triplets déjà présents dans $data_{RDF}$.

Exercice 2: Agrégation et logs (8 points)

On considère une collection MongoDB, nommée `logs` contenant les logs d'un centre de données. Les éléments de cette collection ont le type suivant :

```
< _id: mongoId, ts: timestamp, rack: int,
  node: string, app: string, level: int, message: string >
```

La signification des champs est la suivante : `ts` est le timestamp d'émission du log ; `rack` est le numéro de l'armoire du serveur ou le processus s'exécute ; `node` est le serveur en question ; `app` est l'application exécutée ; `level` est le niveau de gravité (0 pour `FATAL`, 1 pour `ERROR`, 2 pour `WARNING`, 3 pour `INFO` et 4 pour `DEBUG`) ; enfin `message` est le message du log.

On suppose l'existance de l'opérateur `$dateTrunc: { date:` d`, unit:` u`}` où d est une expression qui renvoie une date ou un timestamp et u est une nité parmi `year`, `quarter`, `week`, `month`, `day`, `hour`, `minute` et `second`. L'expression ainsi construite renvoie la date tronquée à l'unité fournie. On peut ainsi récupérer le jour correspondant à un timestamp. On peut par exemple l'utiliser dans `$project` ou `$group`.

Question 2.1: Écrire un pipeline d'agrégation Mongo qui donne pour chaque rack le timestamp du log le plus récent.

Question 2.2: Écrire un pipeline d'agrégation Mongo qui donne les applications n'ayant pas de log de niveau `FATAL` ou `ERROR`.

Question 2.3: Écrire un pipeline d'agrégation Mongo qui donne le tableau des applications sur chaque rack (sans doublon).

Question 2.4: Écrire un pipeline d'agrégation Mongo qui donne pour chaque jour le rack ayant le plus grand nombre de logs de niveau `FATAL`, ainsi que le nombre d'applications produisant des log de niveau `FATAL` sur ce rack. *Indication :* il pourra éventuellement être utile de calculer le nombre de logs de niveau `FATAL` et le nombre d'applications ayant des logs `FATAL` pour chaque rack et chaque jour.

En MongoDB, on peut appeler le framework d'aggrégation via `db.collection.aggregate([...])`
Voici quelques *stages* utilisables dans le pipeline :

$project spec : ensemble de spécifications de champs. Permet d'ajouter, supprimer ou changer la valeurs de certains champs.

$match spec : conditions sur les champs. Filtre les éléments du pipeline à la façon d'une requête MongoDB classique.

$lookup spec :

```
{
  from: <collection to join>,
  localField: <field from the input documents>,
  foreignField: <field from the documents of the "from" collection>,
  as: <output array field>
}
```

Effectue une jointure avec la collection spécifiée, mais en combinant chaque docuemnt de la collection courante avec le tableau des documents correspondants.

$unwind spec : chemin de champs avec $ au début. Produit un document pour chaque valeur du tableau contenu dans le champ spécifié en recopiant le reste du document.

$group spec :

```
{
 _id: <expression>, // Group By Expression
 <field1>: { <accumulator1> : <expression1> },
 ...
}
```

Regroupe les documents par valeur décrite dans `_id` et calcule pour chaque champ la valeur aggrégée décrite par l'accumulateur auquel on passe les valeurs calculées par l'expression. Parmi les accumulateurs possibles, on trouve `$sum`, `$avg`, `$min`, `$max` ou encore `$push` (qui construit un tableau à partir des valeurs reçues).

$sort spec : ensemble de champs avec le ordre de tri (1 ou -1)

FIGURE 2 – Aggrégation pipeline en MongoDB

Exercice 3: Transformation XML pour ingestion dans une base relationnelle (6 points)

On dispose d'un outil de chargement de données vers une base de données relationnelle. Cet outil lit des données dans un fichier XML et les insère dans les tables spécifiées. Le format d'entrée de l'outil est défini par la DTD suivante :

```
<!DOCTYPE tables[
<!ELEMENT tables(table+)>
<!ELEMENT table (r*)>
<!ATTLIST table name CDATA #REQUIRED>
<!ELEMENT r (v+)>
<!ELEMENT v (#PCDATA)>
<!ATTLIST v a CDATA #REQUIRED>
]>
```

Les éléments `r` correspondent à des n-uplets à insérer dans une table. Les éléments `v` contiennent des valeurs (le nom de l'attribut SQL est spécifié par `a`). Par exemple le document suivant :

```
<tables>
  <table name="x">
    <r><v a="t">1</v><v a="u">2</v></r>
    <r><v a="t">3</v><v a="u">4</v></r>
  </table>
</tables>
```

va permettre de d'insérer dans la table `x(t,u)` les n-uplets `(1,2)` et `(3,4)`.

On considère maintenant un document XML `source` ayant la DTD suivante :

```
<!DOCTYPE equipes[
<!ELEMENT equipes (equipe+)>
<!ELEMENT equipe (nom,ville,(joueur|joueurref)+)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT joueur (nom,age)>
<!ATTLIST joueur id ID #REQUIRED
                 saison CDATA #REQUIRED>
<!ELEMENT age (#PCDATA)>
<!ELEMENT joueurref EMPTY>
<!ATTLIST joueurref ref IDREF #REQUIRED
                    saison CDATA #REQUIRED>
]>
```

On considère que si un joueur (ou une référence vers un joueur) apparait dans une équipe, il est membre de cette équipe pour la saison indiquée.

On considère enfin le schéma relationnel correspondant aux déclarations SQL suivantes :

```
1  CREATE TABLE equipe(nom TEXT PRIMARY KEY, ville TEXT);
2  CREATE TABLE joueur(nom TEXT PRIMARY KEY, age INTEGER);
3  CREATE TABLE membre(
4    equipe TEXT REFERENCES equipe(nom),
5    joueur TEXT REFERENCES joueur(nom),
6    saison TEXT,
7    PRIMARY KEY (equipe,joueur,saison)
8  );
```

Question 3.1: Écrire une requête XQuery permettant, à partit d'un fichier `source` de générer un fichier d'entrée pour l'outil de chargement des données afin d'insérer les données du document dans un base ayant le schéma spécifié précédement.