Corrigé rapide de la première session de MIF04 pour l'année 2020-2021

Énoncé: http://emmanuel.coquery.pages.univ-lyon1.fr/mif04/mif04-examen-2020-2021-s1.pdf

Exercice 1

Q1.1

L'information cherchée est l'ensemble des paires titre-année d'édition.

```
SELECT ?titre, ?annee WHERE {
    ?notice <http://www.w3.org/2000/01/rdf-schema#label> ?titre.
    ?notice <http://biblio.org/prop/edition> ?edition.
    ?edition <http://biblio.org/prop/annee> ?annee.
}
```

Q1.2

Attention, il faut bien générer un document conforme à la syntaxe RDF en XML comme dans l'exemple. Il faut bien utiliser les préfixes xml pour définir correctement les noms des éléments qui représente les triplets.

Remarques:

Ici on intègre directement les préfixes dans la syntaxe XML générée. En réalité, il faudrait les définir avec des déclarations `namespace` XQuery. Cela n'étant pas au centre de l'exercice, les déclarations telles qu'effectuées ci-dessous ont été comptées justes.

Les triplets sont à générer pour chaque notice, chaque édition et chaque auteur. Il peut y avoir des doublons dans la génération.

Les 4 premières lignes n'étaient pas demandées et sont juste placées pour rappel (cf énoncé).

```
let $rdf := "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
    $rdfs := "http://www.w3.org/2000/01/rdf-schema#",
    $biblio := "http://biblio.org/"
return
<rdf:RDF xmlns:rdf="{$rdf}"
         xmlns:rdfs="{$rdfs}"
         xmlns:bibrel="{concat($biblio,"prop/")}">
    {
        for $no in //notice
        return
        <rdf:Description ref:about="{concat($biblio,"/notice/",$no/@id)}">
            <rdfs:label>{$no/titre/text()}</rdfs:label>
            {
```

```
                for $aut in $no/auteur
                return
                <bibrel:auteur rdf:resource="
{concat($biblio,"/auteur/",$aut/@id)}"/>
            }
            {
                for $ed in $no/edition
                return
                <bibrel:edition rdf:resource="
{concat($biblio,"/edition/",$ed/@id)}"/>
            }
        </rdf:Description>
    }
    {
        for $aut in //auteur
        return
        <rdf:Description ref:about="{concat($biblio,"/auteur/",$aut/@id)}">
            <bibrel:nom>{$aut/nom/text()}</bibrel:nom>
            <bibrel:prenom>{$aut/prenom/text()}</bibrel:prenom>
        </rdf:Description>
    }
    {
        for $ed in //edition
        return
        <rdf:Description ref:about="{concat($biblio,"/edition",$ed/@id)}">
            <bibrel:annee>{$ed/annee/text()}</bibrel:annee>
            <bibrel:editeur>{$ed/editeur/text()}</bibrel:editeur>
        </red:Description>
    }
<rdf:RDF>
```

Exercice 2

Attention: il faut prendre en compte les cas suivants:

- reduce n'est pas appelé si on a une seule vidéo dans une certaine journée
- il y aura des rereduce pour certaines vidées, la valeur émise par le map et renvoyées par le reduce doivent avoir la même forme (le même type, les mêmes champs, etc). Cette orme correspond à celle des éléments dans le tableau des valeurs passées en argument à reduce.
- les tableaux ne peuvent pas être émis directement, il faut les encapsuler dans des objets

Remarque: le `finalize` est ici accessoire dans le sens où il enlève juste de l'information non demandée (le nombre de vues).

```
let maptk = function() {
    for(let i = 0; i < this.vues.length; i++) { // ou for ... of ou
forEach, etc
        let v = this.vues[i];
        emit(v.jour, { "top": [ { "_id": this._id, "nombre": v.nombre,
```

```
"titre": this.titre } ] })
    }
}

let reducetk = function (jour /* key */, plus_vues /* values */)
    /* comparateur qui place en premier les vidéos avec le plus de vues
(ordre par nombre inverse) */
    let cmp_videos = function(v1, v2) {
        return v2.nombre - v1.nombre;
    }
    /* aggrégation de videos de plus_vues dans un unique tableau */
    let vues = [];
    for(let i = 0; i < plus_vues.length; i++) {
        for(let j = 0; j < plus_vues[i].top.length) {
            vues.push(plus_vues[i].top[j]);
        }
    }
    /* tri */
    vues.sort(cmp_videos);
    /* extraction des 10 première vidéos */
    let res_t = [];
    for(let i = 0; i < 10 && i < vues.length; i++) {
        res_t.push(vues[i]);
    }
    return { "top": res_t };
}

let finalizetk = function(jour, top_vues) {
    let res = { "top": [] };
    for (let i = 0; i < 10 && i < top_vues.top.length; i++) {
        res.top.push({"_id": top_vues.top[i]._id, "titre":
top_vues.top[i].titre});
    }
}
```

On peut faire plus succint pour `reducetk` avec un peu de connaissance de Javascript:

```
let reducetk = function (jour, plus_vues) {
  return {
    top: plus_vues
      .map((t) => t.top)
      .flat()
      .sort((v1, v2) => v2.nombre - v1.nombre)
      .slice(0, 10),
  };
};
```

Exercice 3

Pour les deux première questions, il faut compter les triplets déduits. Pour être efficace, on peut regarder les règles ainsi que la partie du graphe qui concerne les triplets RDFS. On voit alors que si on applique les règles dans l'ordre SP, Se, NJ, Domain, Range, SC on a pas besoin de reboucler sur les règles précédentes.

On obtient le décompte suivant:

- SP: 1 triplet
- Se: 5 triplets
- NJ: 2 triplets
- Domain: 1 (ext) + 2 (nbJoueurs) + 3 (serie), attention aux doublons et il faut prendre en compte l'effet de SP et NJ
- Range: 0 (car villainous type jeu a déjà été déduit)
- SC: 1 (attention aux triplets déjà déduits)

Remarque: les questions 1 et 2 ne demandaient pas de lister les triplets, même si les écrire au brouillon où les dessiner pouvait s'avérer utile pour y répondre.

Question 3.1

3 types (via Domain).

Question 3.2

15 triplets

Question 3.3

Il n'y a pas de règle qui force directement la symétrie ou la transitivité, donc si on prend un graphe où la relation compatible initial n'est pas symétrique, ni transitive, elle ne le sera pas forcément dans le graphe saturé. Un exemple simple, prendre deux triplets dans le graphe initial:

```
jj:a jp:compatible jj:b.
jj:b jp:compatible jj:c.
```

Le graphe saturé est identique au graphe initial et la relation n'est ni symétrique, ni transitive.

Question 3.4

S'il n'y a pas de triplet jp:compatible dans le graphe initial, les seuls triplets jp:compatible présents sont déduits par Se. Or la règle est symétrique dans son application, on obtient donc dans ce cas un graphe saturé où jp:compatible est symétrique. En revanche, rien n'assure la transitivité, par exemple

```
jj:a jp:serie js:s.
jj:b jp:serie js:s.
jj:b jp:serie js:t.
jj:c jp:serie js:t.
```

produit un graphe saturé dans lequel on a

```
jj:a jp:compatible jj:b.
jj:b jp:compatible jj:c.
```

mais pas

```
jj:a jp:compatible jj:c.
```

Exercice 4

Remarque: cet exercice étant particulièrement difficile, il a été compté comme bonus dans la note finale (comme le montre le calcul de la note dans tomuss).

Pour réussir cet exercice, il fallait comprendre le lien entre les données relationnelles et leur représentation XML calculée par les vues. Il fallait ensuite refaire de nouvelles requêtes (similaires aux vues) produisant le même XML que celui produit par XQuery.

Question 4.1

Remarque: on veut 1 résultat par ligne de la vue V1. En regardant les contraintes de clé primaire et étrangère de la base et le `GROUP BY` de V1, on peut en déduire qu'il faut 1 résultat pour chaque valeur de `S.C`. L'attribut `c` du XML de `V1` correspond exactement à cette valeur, donc on produit bien 1 élément XML ayant pour racine une élément `res1` pour chaque valeur de `S.C`.

Attention, comme S est jointe avec T dans V1, il faut supprimer les tuples de S qui n'ont pas de correspondance dans `T`. Ici, on le fait avec `EXISTS`, mais il est également possible de reproduire la jointure de `V1` si on oublie pas le `GROUP BY` (le `EXISTS` est probablement plus efficace ici car on utilise pas les valeurs de T).

La requête XQuery récupère l'élément `y` qui correspond au champ `S.D`. Là aussi, une seule valeur car `S.C` est clé primaire de `S`. Remarque: la requête XQuery conserve le tag `y` autour de `S.D`

En appliquant la mise en forme XML de la requête XQuery, on obtient:

```
SELECT XMLElement(name "res1",
                  XMLAttributes(S.C as "h"),
                  XMLForest(S.D as "y"))
       as result
FROM S
WHERE EXISTS (SELECT 1
              FROM T
              WHERE S.C = T.C)
```

Question 4.2

On veut un document par valeur de `R.A` (avec un raisonnement similaire à la question précédente), sauf pour les tuples pour lesquels ont ne peut pas faire de jointure avec `S` et `T`.

La boucle `for` XQuery itère sur les paires de `x` (donc les valeurs de `S.C`) ayant des attributs `c` différents, (donc sur des valeurs de `S.C` différentes, mais ayant le même `S.A`) et tels qu'on puisse leur trouver un `z` (c'est-à-dire un `T.F`) identique. Cette boucle est à imiter en faisant des groupes de tuples partageant le même `A` et en utilisant `XMLAgg` pour produire sous-résultat `p` par tuple du groupe.

Les valeurs à produire dans `p1` et `p2` viennent de `y`, c'est-à-dire de `S.D`

On obtient:

```
SELECT XMLElement(name "res2",
                  XMLAgg(XMLElement(name "p",
                                    XMLForest(S1.D as "p1", S2.D as
"p2"))))
       as result
FROM R JOIN S S1 ON R.A = S1.A
       JOIN S S2 ON R.A = S2.A
WHERE S1.C <> S2.C
  AND EXISTS (SELECT 1
              FROM T T1, T T2
              WHERE T1.C = S1.C
                AND T2.C = S2.C
                AND T1.F = T2.F)
GROUP BY R.A
```