| Nom : | Prénom : | *page 1* |
|---|---|---|

# MLBDA – 4I801- Examen réparti du 7 janvier 2015

**Version CORRIGEE**

| Ex1 | Ex2 | Ex3 | Ex4 | Ex5 | Total |
|---|---|---|---|---|---|
| | | | | | |

**Seuls les documents de cours et de TD sont autorisés** – Durée : 2h.

**Répondre aux questions sur la feuille du sujet** dans les cadres appropriés. Utiliser le dos de la feuille précédente si la réponse déborde du cadre. Le barème est donné à titre indicatif. La qualité de la rédaction sera prise en compte. Ecrire à l'encre bleue ou noire. Ne pas dégrafer le sujet. Eteindre et ranger tout téléphone et autre appareil électronique.

## Exercice 1. Xschema — 5 pts

On considère le fichier *formation.xml* (annexe 1) décrivant l'emploi du temps d'une formation universitaire. On souhaite définir le schéma XML validant ce fichier en complétant le fichier *formation.xsd* (annexe 2).

**Question 1 (1 pt)**. Complétez la définition de l'élément `semestre` (voir annexe 1), sachant que le contenu d'un élément `semestre` est de la forme Si avec i ∈ {1, 2}.

```
<xs:element name="semestre">




</ xs:element>
```

```
<xs:element name="semestre">
 <xs:simpleType>
      <xs:restriction base ="xs :string">
            <xs :pattern value = "S1 | S2  "/>
      </xs :restriction>
</xs :simpleType>
</ xs:element>
```

On peut aussi mettre pattern value = "S[1-2]" ou enumeration value ="S1" (2 éléments enumeration pour les 2 valeurs)

**Question 2 (1 pt)**. Complétez la définition de l'élément `creneau` du fichier *formation.xsd* (voir annexe 1) sachant qu'un créneau est caractérisé par les informations suivantes :

- Le moment où a lieu le créneau (élément `semestre` ou élément `le`)
- Le jour de la semaine du créneau
- L'heure du début et de fin du créneau (éléments `de` et `a`)
- La salle
- Un ou deux intervenants
- Une matière

```
<xs:element name="creneau">
```

</ xs:element>

```
<xs:complexType>
  <xs:sequence>
        <xs:choice>
              <xs:element name="le" type="xs:date"/>
              <xs:element ref="frm:semestre"/>
        </ xs:choice>
        <xs:element name="jour" type="frm:jour−de−la−semaine"/>
        <xs:element name="de" type="xs:time"/>    ** string est aussi accepté
        <xs:element name="a" type="xs:time"/>     **ici aussi
        <xs:element name="salle" type="xs:string" />
        <xs:element name="intervenant" type="xs:string" maxOccurs="2" />
        <xs:element name="matiere" type="xs:string"/>
    </ xs:sequence>
</xs:complexType>
```

**Question 3 (1 pt)**. Expliquez pourquoi ce schéma n'aurait pas pu être décrit par une DTD (donnez une raison autre que l'utilisation de types).

Ce fichier contient deux éléments `matière` (un élément `matière` dans `créneau` et un élément `matiere` dans `matieres`) qui ne sont pas du même type.

**Question 4 (2 pts)**. On souhaite préciser la contrainte suivante : *tout créneau doit se référer à une matière qui existe*. Définissez cette contrainte en précisant à quels endroits vous ajoutez les déclarations nécessaires.

Dans la definition de l'élément `matieres` (à la fin, entre </xs:complexType> et </xs:element>

```
<xs:key name="cleMatiere">
<xs:selector xpath="frm:matiere"/>
<xs:field xpath="frm:sigle"/>
</ xs:key>
```

Dans la définition de l'élément formation entre </xs:complexType> et </xs:element>

```
<xs:keyref name="referenceMatiere" refer="frm:cleMatiere">
<xs:selector xpath="frm:creneaux/frm:creneau"/>
<xs:field xpath="frm:matiere"/>
</ xs:keyref>
```

## Exercice 2. XPath — 4 pts

On considère le fichier *"sports.xml"* (annexe 3) qui décrit les épreuves des sports des jeux olympiques d'hiver 2014. Exprimez en XPath les requêtes suivantes :

**Question 1 (0,5pt).** Le nom des sports qui n'ont eu aucune épreuve le "*23-02"*.

```
//sport[not(epreuve[date='23-02'])]/@nom
```

**Question 2 (0,5pt).** Le type de la 3e épreuve de chaque sport (s'il y en a).

```
//sport/epreuve[3]/@type
```

**Question 3 (1pt).** Les noms des sports avec au moins 2 épreuves pour les hommes ou 2 pour les femmes.

```
//sport[count(epreuve[@category="Femmes"]) > 1 or
      count(epreuve[@category="Hommes"]) > 1 ]/@nom
```

**Question 4 (1pt).** Le type des épreuves en '*Ski de fond'* qui précèdent les *'50 km libre'* dans le document.

```
//sport[@nom="Ski de fond"]/epreuve[@type="50km libre"]/preceding-sibling::*/@type
```

**Question 5 (1pt).** Expliquez en une phrase en français ce que renvoie la requête XPath suivante :

```
//sport/epreuve/date[. = preceding::date]
```

Les dates où il y a eu au moins deux épreuves

Pour éviter les doublons, il faudrait écrire :

```
//sport/epreuve/date[. = preceding::date and not(. = following::date)]
```

## Exercice 3. XQuery — 6 pts

On considère les fichiers *"sports.xml"* (annexe 3) et *" gagnants.xml"* (annexe 4) contenant la liste des gagnants de médailles. **Ce fichier contient uniquement les athlètes ayant gagné une médaille au moins**.
Exprimez en XQuery les requêtes suivantes. Pour chaque requête, on donne la DTD du résultat qu'on veut obtenir et un exemple de résultat en XML.

**Question 1 (1,5 pt).** Le nombre d'athlètes par pays :

**DTD du résultat :**
```
<!ELEMENT resultat (pays)+>
<!ELEMENT pays EMPTY>
<!ATTLIST pays nom CDATA>
<!ATTLIST pays nb CDATA>
```

**Exemple**
```
<resultat>
  <pays nom="Pays-Bas" nb="4"/>
  <pays nom="Corée du Sud" nb="1"/>
  ...
</resultat>
```

```
<resultat> {




} </resultat>
```

```
for  $pays in distinct-values(doc("athletes.xml")//pays/text())
let $ath := doc("athletes.xml")//athlete[pays/text()=$pays]/@id
let $nbr := count($ath)
return <pays nom="{$pays}" medailles="{$nbr}">
```

**Question 2 (1,5 pt).** Le(s) plus jeune(s) athlète(s) par pays :

**DTD du résultat**
```
<!ELEMENT resultat (pays)+>
<!ELEMENT pays (jeune)+>
<!ATTLIST pays nom CDATA>
<!ATTLIST pays age-min CDATA>
<!ELEMENT jeune EMPTY>
<!ATTLIST jeune nom CDATA>
<!ATTLIST jeune prenom CDATA>
```

**Exemple**
```
<resultat>
 <pays nom="Pays-Bas" age-min="27">
  <jeune nom="Smeekens" prenom="Jan"/>
  <jeune nom="Stoch" prenom="Kamil"/>
 </pays>
 <pays nom="Norvège" age-min="23">
  <jeune nom="Weng" prenom="Heidi"/>
 </pays>
 ...
</resultat>
```

```
<resultat> {
for …
```

```
return <pays …




} </resultat>
```

```
for $pays in distinct-values(doc("athletes.xml")//pays)
let $min := min(doc("athletes.xml")//athlete[pays/text()=$pays]/age)
let $jeune := doc("athletes.xml")//athlete[age=$min]
return <pays nom="{$pays}" age-min="{$min}"> {
 for $a in $jeune
 return <plus-jeune nom="{$a/nom/text()}" prenom="{$a/prenom/text()}" />
}
</pays>
```

**Question 3 (1,5 pt).** Les épreuves qui ont eu lieu à une date à laquelle *"Kamil Stoch"* a gagné une médaille (attention : *Kamil Stoch* a gagné plusieurs médailles à des dates différentes):

**DTD du résultat**

```
<!ELEMENT resultat (epreuve)*>
<!ELEMENT epreuve EMPTY>
<!ATTLIST epreuve type CDATA>
<!ATTLIST epreuve date CDATA>
```

**Exemple**

```
<resultat>
  <epreuve type="Skiathlon 30 km"
           date="9-02" />
  <epreuve type="Grand tremplin"
           date="15-02" />
  ...
</resultat>
```

```
<resultat> {




} </resultat>
```

**for** $d **in** doc("sports.xml")//sport/epreuve[descendant::*/@aid=

```
                    doc("athletes.xml")//athlete[nom/text()="Stoch" and
            prenom/text()="Kamil"]/@id]/date,
   $e in doc("sports.xml")//sport/epreuve[date/text()=$d/text()]/@type
return <epreuve type="{$e}" date="{$d}" />
```

**Question 4 (1,5 pt).** Les athlètes compatriotes du gagnant (médaille d'or) de l'épreuve *"50 km libre"* du *"Ski de fond" :*

**DTD du résultat**

```
<!ELEMENT resultat (athlete)*>
<!ELEMENT athlete EMPTY>
<!ATTLIST athlete nom CDATA>
<!ATTLIST athlete prenom CDATA>
```

**Exemple**

```
<resultat>
  <athlete nom="Fatkulina" prenom="Olga"/>
</resultat>
```

```
<resultat> {









} </resultat>
```

```
for $gagnant in doc("sports.xml")//sport[@nom="Ski de fond"]
                        /epreuve[@type="Sprint"]/medaille-or/@aid
for $pays in doc("athletes.xml")//athlete[@id=$gagnant]/pays/text()
for $a in doc("athletes.xml")//athlete[pays/text()=$pays and @id!=$gagnant]
return <athlete nom="{$a/nom/text()}" prenom="{$a/prenom/text()}" />
```

## Exercice 4. SPARQL — 4 pts

Considérons les triplets suivants donnés sous forme factorisée :

```
@prefix : <http://dbpedia.org/resource/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

#singers
:Shania_Twain
    a foaf:Singer ;
    rdfs:label "Shania Twain"@en ;
    :birthDate "1965-08-28" ;
    :placeOfBirth "Windsor"@en ;
    :genre :country_pop , :country_rock , :rock , :pop ;
    :instruments "vocals" , :guitar .
:Whitney_Houston
    a foaf:Singer ;
    rdfs:label "Whitney Houston"@en ;
    :birthDate "1963-08-09" ;
    :placeOfBirth "Newark"@en ;
    :deathDate "2012-02-11" ;
    :genre :pop , :sould, :rnb ;
    :instruments "vocals" , :piano .
:Mariah_Carey
    a foaf:Singer ;
    rdfs:label "Mariah Carey"@en ;
    :birthDate "1969-03-27";
    :placeOfBirth "Long Island"@en ;
    :genre :pop , :soul, :rnb ;
    :instruments "vocals" , :guitar .
:Jay-Z
    a foaf:Singer ;
    rdfs:label "Jay-Z"@en ;
    :birthDate "1969-12-04" ;
    :placeOfBirth "Brooklyn"@en ;
    :genre :hiphop ;
    :instruments "vocals" .

#actors
:Emilia_Clarke
    a foaf:Actor ;
    rdfs:label "Emilia Clarke"@en ;
    :birthDate "1987-05-01";
    :placeOfBirth "London"@en ;

#songs
:Emilia_Clarke
    :sang :fisherman .
:Whitney_Houston
    :sang :believe , :run .
:Mariah_Carey
     :sang :believe , :odds, :myall  .
:Shania_Twain
    :sang :kaching .
:Jay-Z
    :sang :empire .
```

Exprimer les requêtes suivantes en SPARQL.
Remarque : ne pas écrire la déclaration des préfixes dans les requêtes

**Question 1 (1 pt).** Les chanteurs de pop et de rock qui jouent de la guitare.
Le résultat de la requête est

| singer |
|---|
| :Shania_Twain |

```
prefix : <http://dbpedia.org/resource/>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix foaf: <http://xmlns.com/foaf/0.1/>
# les chanteurs de pop et de rock qui jouent de la  guitare.
select ?singer
{
?singer :genre :pop . ?singer :instruments :guitar   .  ?singer :genre
:rock .
}
```

**Question 2 (1 pt).** Les duos des chanteurs ayant chanté ensemble. (Une réponse sans doublon donne un bonus)

Le résultat de la requête est :

| singer | other |
|---|---|
| :Mariah_Carey | :Whitney_Houston |

```
select ?singer ?other
{
?singer a foaf:Singer . ?singer :sang ?song . ?other :sang ?song
Filter (?other!=?singer && str(?other) >str(?singer))
}
```

**Question 3 (1 pt).** Les chanteurs en vie ayant réalisé un duo avec des chanteurs qui sont décédés. Retourner le nom du chanteur en vie et de celui décédé.

Le résultat de la requête est :

| singer | collab |
|---|---|
| :Mariah_Carey | :Whitney_Houston |

```
select ?singer ?collab
{
?singer a  foaf:Singer . ?collab a foaf:Singer. ?collab :deathDate ?dc.
?singer :sang ?s . ?collab :sang ?s
OPTIONAL {?singer :deathDate ?ds} Filter (!bound(?ds))
}
```

**Question 4 (1,5 pt).** Le chanteur qui chante le plus grand nombre de chansons avec le nombre de ces chansons. (on suppose que le nombre total de chansons est différent pour chacun).

Le résultat de la requête est

| singer | total-songs |
|---|---|
| :Mariah_Carey | 3 |

```
select ?singer (count(?song) as ?total_songs)
{
?singer a foaf:Singer . ?singer :sang ?song
}
group by ?singer
order by DESC(?total_songs )
limit 1
```

## Exercice 5. Intégration de données 1 pt

Il existe deux grands types d'architecture pour l'intégration de données : les entrepôts de données où les données intégrées sont matérialisées et les médiateurs, où l'intégration est virtuelle. Expliquez pourquoi les entrepôts de données sont mieux adaptés que les médiateurs pour les applications de type OLAP (applications d'aide à la décision).

Les requêtes OLAP utilisent de très grandes quantités de données, font des opérations coûteuses (group by, agrégats), et ont besoin d'utiliser des données historisées.

L'utilisation d'un entrepôt permet d'optimiser les requêtes en mettant des index, en créant des résumés de données et en stockant les historiques. Cela permet aussi de ne pas dégrader les performances des BD opérationnelles.

Le médiateur a de très mauvaises performances, surtout pour les requêtes utilisant de grandes quantités de données.