Nom :	Prénom :	*page 1*

Modèles Avancés pour les Bases de Données

MABD – MI005

Examen réparti du 6 janvier 2012

Version CORRIGEE

Exercice 1. DTD — 5 pts

Question 1. On considère la DTD *A.dtd* suivante. Les éléments non définis sont vides.

<!ELEMENT A ((B |(C, D?) | E), ((D,(F,G)|B) | ((I,J)*, H+))*)>

a) Donnez le plus petit document XML conforme à cette DTD

<A><B/></A> ou <A><C/></A> ou <A><E/></A>

b) Pour chaque document D1 et D2 suivant, indiquez s'il est conforme à la DTD *A.dtd* ? S'il n'est pas conforme, expliquez pourquoi.

D1 : <A><C/><F/><G/><B/><I/><J/><H/></A>

Non, doit contenir un D avant le F

D2 : <A><C/><D/><B/><H/><I/><J/><H/></A>

Oui, conforme

Question 2. On considère la DTD suivante :

```
<!DOCTYPE Etudiants [     <!ELEMENT Etudiants (Etudiant+)>
                          <!ELEMENT Etudiant (Post*, Amis)>
                          <!ATTLIST Etudiant Nom CDATA #REQUIRED>
                          <!ELEMENT Post (#PCDATA)>
                          <!ELEMENT Amis (Ami*)>
                          <!ELEMENT Ami (Nom)>
                          <!ELEMENT Nom (#PCDATA)>
                    ]>
```

On veut utiliser le concept IDREF au lieu de la structure en sous-élément pour représenter les Amis. Modifier la DTD en conséquence, en veillant à ce qu'elle soit le plus simple possible, sans perdre d'information.
Remarque : les amis d'un étudiant ne sont pas tous étudiants.

```
<!DOCTYPE Etudiants[
<!ELEMENT Etudiants (Etudiant+, Ami *)>
<!ELEMENT Etudiant ( Post*)>
<!ATTLIST Etudiant Nom CDATA #REQUIRED
                   Amis IDREFS #REQUIRED>
<!ELEMENT Post (#PCDATA)>
<!ELEMENT Ami  EMPTY>
<!ATTLIST Ami Nom ID  #REQUIRED>
]>
```

Question 3. On considère le schéma XML ci-dessous. Ce schéma est-il satisfait par exactement le même ensemble de documents XML qui sont conformes à la DTD Etudiants de la question 2 ? Si la réponse est NON, expliquez pourquoi.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="Etudiants">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="Etudiant">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="Post" type="xsd:string"/>
                <xsd:element name="Amis">
```

```
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="Ami">
                              <xsd:complexType>
                                <xsd:sequence>
                                  <xsd:element name="Nom" type="xsd:string"/>
                                </xsd:sequence>
                              </xsd:complexType>
                            </xsd:element>
                        </xsd:sequence>
                      </xsd:complexType>
                    </xsd:element>
                  </xsd:sequence>
                  <xsd:attribute name="Name" type="xsd:string" use="required"/>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
    </xsd:schema>
```

Rép : Non, dans la DTD on peut avoir plusieurs éléments Etudiant dans Etudiants (Etudiant +), alors que dans le schéma, on a maxoccurs = 1 par défaut..

Exercice 2 : Xquery et Xpath 10 pts

Soient « cat.xml », « price.xml » et « ord.xml » trois fichiers XML décrivant respectivement les catalogues des produits par département, la liste des prix des produits et les différentes ventes des articles à une date donnée.

cat.xml

```
<catalog>
   <product dept="WMN">
      <number>557</number>
      <name language="en">Linen Shirt</name>
      <colorChoices>beige sage</colorChoices>
   </product>
   <product dept="ACC">
      <number>563</number>
      <name language="en">Ten-Gallon Hat</name>
   </product>
   <product dept="ACC">
      <number>443</number>
      <name language="en">Golf Umbrella</name>
   </product>
   <product dept="MEN">
      <number>784</number>
      <name language="en">Rugby Shirt</name>
      <colorChoices>blue/white blue/red</colorChoices>
      <desc>Our <i>best-selling</i> shirt!</desc>
    </product>
</catalog>
```

price.xml

```
<prices>
  <priceList effDate="2004-11-15">
     <prod num="557">
```

```
        <price currency="USD">29.99</price>
        <discount type="CLR">10.00</discount>
    </prod>
    <prod num="563">
        <price currency="USD">69.99</price>
    </prod>
    <prod num="443">
      <price currency="USD">39.99</price>
      <discount type="CLR">3.99</discount>
    </prod>
    <prod num="784">
        <price currency="USD">25.99</price>
    </prod>
  </priceList>
</prices>
```

ord.xml

```
<order num="00299432" date="2004-09-15" cust="0221A">
   <item dept="WMN" num="557" quantity="1" color="beige"/>
   <item dept="ACC"  num="563" quantity="1"/>
   <item dept="ACC"  num="443" quantity="2"/>
   <item dept="MEN"  num="784" quantity="1" color="blue/white"/>
   <item dept="MEN"  num="784" quantity="1" color="blue/red"/>
   <item dept="WMN" num="557" quantity="1" color="sage"/>
</order>
```

A. XPATH (4 pts)

A.1) Evaluer les expressions XPATH suivantes sur les documents et donner le résultat obtenu.

```
document("ord.xml")//order/*[exists(@color)]/@*
```

Retourne uniquement les attributs des éléments « item » (nœud fils de order) qui au moins un attribut « color »
dept="WMN" num="557" quantity="1" color="beige"
dept="MEN" num="784" quantity="1" color="blue/white"
dept="MEN" num="784" quantity="1" color="blue/red"
dept="WMN" num="557" quantity="1" color="sage"

```
document("cat.xml")//product[* except (number | name) ]/@dept
```

Retourne les départements des produits ayant au moins un élément autre que number et name

dept="WMN" dept="MEN"

```
document("cat.xml")//product[position() mod 2 = 0]/number
```

Remarque: l'opérateur mod calcule le modulo.

Retourne uniquement le numéro des produits qui se trouvent en positions paires 2ème 4ème etc.
<number>563</number><number>784</number>

```
document("cat.xml")//colorChoices/parent::*[child::desc]/name
```

Retourne le nom des produits pères de colorchoices qui un fils « desc ».
<name language="en">Rugby Shirt</name>

A.2) Exprimer avec une seule expression XPATH les requêtes suivantes :

Parmi les produits du département ACC ayant leur numéro inférieur à 700, sélectionner uniquement le deuxième produit.

```
doc("cat.xml")//product[number < 700 and @dept = "ACC"][2]
```

Les noms des produits du catalogue qui précèdent le produit numéro 443 (ses précédents frères).

```
document("cat.xml")//product/preceding-sibling::*[number=443]/name
```

Les prix des produits du catalogue ayant au moins un choix de couleur ‘bleu’.

```
doc("price.xml")//prod[@num=(doc("cat.xml")//product[contains(colorChoices,"blue")]/number)] /price
```

B. XQUERY (6 pts)

Ecrire les requêtes XQUERY permettant d’obtenir les informations suivantes :

1) Les noms des produits du département ACC triés suivant leurs noms. Le résultat de la requête doit être formaté comme suit :

```
<ul type="square">
     <li>Golf Umbrella</li>
     <li>Ten-Gallon Hat</li>
</ul>
```

```
<ul type="square">
{
for $product in doc("cat.xml")/catalog/product
where $product/@dept='ACC'
order by $product/name
return <li>{data($product/name)}</li>
}
</ul>
```

2) Les différents numéros des produits. Les numéros des produits du département ACC sont encadrés par <acc> </acc> et les autres par <other></other>. Le résultat de la requête doit être :

```
<results>
     <other>557</other>
     <acc>563</acc>
     <acc>443</acc>
     <other>784</other>
</results>
```

```
<results>
{
for $prod in (doc("cat.xml")/catalog/product)
return if ($prod/@dept = 'ACC')
then <acc>{data($prod/number)}</acc>
else <other>{data($prod/number)}</other>

}
</results>
```

3) Le numéro et le nom du produit le moins cher. Le résultat de la requête doit être :

```
<result> <product num="784" name="Rugby Shirt" minprice="25.99"/> </result>
```

```
<order>
{

let $prices:=document("price.xml")//prod/price
let $m:=min($prices)
for $np in document("price.xml")//prod[price=$m]/@num,
$pr in document("cat.xml")//product[number=$np]
return <product num="{$np}" name="{$pr/name}" minprice="{$m}" />

}
</order>
```

4) Les noms des produits du département “ACC” ou “WMN” dont le numéro de produit est supérieur à 100, ayant au moins une couleur et dont le nom commence par la lettre L. Le résultat de la requête doit être :

```
<result> <name language="en">Linen Shirt</name> </result>
```

```
<result>
```

```
{
for $prod in document("cat.xml")//product
let $prodDept := $prod/@dept
where
$prod/number > 100 and
starts-with($prod/name, "L") and
exists($prod/colorChoices) and
($prodDept="ACC" or $prodDept="WMN")
return $prod/name
}
</result>
```

5) Le numéro, le nom, le prix de chacun des articles vendus et la quantité totale de ces articles regroupés par leur numéro. Le résultat doit être trié par numéro croissant. Le résultat de la requête doit être :

```
<order>
     <item num="443" name="Golf Umbrella" price="39.99" qte="2" />
     <item num="557" name="Linen Shirt" price="29.99" qte="2" />
     <item num="563" name="Ten-Gallon Hat" price="69.99" qte="1" />
     <item num="784" name="Rugby Shirt" price="25.99" qte="2" />
</order>
```

```
<order>
{
for  $item in distinct-values(doc("ord.xml")//item/@num)
let $it:=doc("ord.xml")//item[@num=$item],
$product:=doc("cat.xml")//product[number=$item],
$price:=doc("price.xml")//prod[@num=$item]
order by $item
return
<item num="{$item}" name="{$product/name}"
qte="{sum($it/@quantity)}"
price="{$price/price}"/>
}
</order>
```

6) Evaluer l'expression XQUERY suivante et donner le résultat obtenu.

```
<result>
{
for $d in distinct-values(doc("ord.xml")//item/@dept)
let $items := doc("ord.xml")//item[@dept = $d]
order by $d
return
  <department code="{$d}">
  {
    for $n in distinct-values($items/@num)
    let $qte := $items[@num =$n ]/@quantity
    let $pr := doc("price.xml")//prod[@num =$n ]/price
    let $dc := doc("price.xml")//prod[@num =$n]/discount
    order by $n
    return
    if (exists($dc)) then
      <item num="{$n}"  price="{xs:decimal($pr)}"  qte=" {sum($qte)"
       discount="{xs:decimal($dc)}" totalPrice="{(xs:decimal($pr)*sum($qte))-$dc}" />
    else
      <item  num="{$n}" price="{xs:decimal($pr)} "qte=" {sum($qte)} "discount="0"
        totalPrice="{(xs:decimal($pr)*sum($qte))}" />
  }
  </department>
 }
</result>
```

Le numéro, le prix unitaire, la quantité, la remise et le prix total des articles vendus par chaque département (totalPrice=[qte* price]-discount).
<result>

```
    <department code="ACC">
            <item num="443" price="39.99 " qte="2 " discount="3.99" totalPrice="75.99"/>
            <item num="563" price="69.99 " qte="1 " discount="0" totalPrice="69.99"/>
    </department>
    <department code="MEN">
            <item num="784" price="25.99 " qte="2 " discount="0" totalPrice="51.98"/>
    </department>
    <department code="WMN">
            <item num="557" price="29.99 " qte="2 " discount="10" totalPrice="49.98"/>
    </department>
</result>
```

Exercice 3 : XML Schema — 5 pts

Soit le document *sport.xml* décrivant brièvement des équipes de football et de basketball et les rencontres entre équipes. Le document se décompose en deux parties. Tout d'abord, l'annuaire recense les équipes par sport et par pays. Puis, l'historique des matchs récapitule, pour chaque coupe, les rencontres entre équipes. Il y a deux équipes par match, une équipe peut jouer plusieurs matchs.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<sport xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="sport.xsd">
<!-- annuaire des équipes-->
  <foot>
    <p nom="France">
      <e>OM</e> <e>TFC</e> <e>OL</e> <e>PSG</e> <e>SL</e>
    </p>
    <p nom="Angleterre">
      <e>Liverpool</e> <e>Arsenal</e> <e>Chelsea</e>
    </p>
    <p nom="Italie">
      <e>Milan AC</e> <e>Juventus</e> <e>AS Roma</e>
    </p>
  </foot>
  <basket>
    <p nom="France">
      <e>ASVEL</e> <e>PL</e> <e>Elan</e> <e>MSB</e>
    </p>
    <p nom="Angleterre">
      <e>Trafford</e> <e>C.Jets</e> <e>M.Magic</e>
    </p>
  </basket>
<!--historique des matchs entre équipes-->
  <coupeFoot>
    <match> <e>TFC</e> <e>OM</e> </match>
    <match> <e>Juventus</e> <e>PSG</e> </match>
    <match> <e>Liverpool</e> <e>Milan AC</e> </match>
    <match> <e>OM</e> <e>Juventus</e> </match>
  </coupeFoot>
  <coupeBasket>
    <match> <e>C.Jets</e> <e>MSB</e> </match>
    <match> <e>ASVEL</e> <e>Elan</e> </match>
    <match> <e>ASVEL</e> <e>M.Magic</e> </match>
  </coupeBasket>
</sport>
```

Le document *sport.xsd* est le schéma XML du document *sport.xml*. On donne une trame partielle du schéma *sport.xsd* :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="sport">
```

```
            <xs:complexType>
              <xs:sequence>
                <xs:element name="foot">
                    ...
                </xs:element>
                ...
              </xs:sequence>
            </xs:complexType>
                ...
          </xs:element>
           ...
        </xs:schema>
```

Question 1. On veut définir une contrainte pour vérifier qu'une équipe jouant un match de la coupe de foot est bien recensée dans l'annuaire comme étant une équipe de foot. Par exemple, un match de foot entre PL et OM est incohérent. Définir les éléments key et keyref exprimant cette contrainte et préciser quel est leur père.

```
Les éléments key et keyref ont pour père:     <xs:element name="sport">

Le plus petit ancêtre commun permettant d'atteindre les équipes de l'annuaire et les
équipes jouant un match est l'élément sport.

Erreur à éviter : ne pas définir la clé dans l'élément foot, car ensuite on ne pourra
pas l'utiliser pour la keyref dans coupeFoot.


    <xs:key name="cléFoot">
      <xs:selector xpath="foot/p/e"/>
      <xs:field xpath="."/>
    </xs:key>

    <xs:keyref name="matchFoot" refer="cléFoot">
      <xs:selector xpath="coupeFoot/match/e"/>
      <xs:field xpath="."/>
    </xs:keyref>
```

Question 2. Définir une contrainte pour vérifier l'unicité globale du nom d'une équipe recensée dans l'annuaire, tous sports et tous pays confondus. Définir l'élément et préciser quel est son père.

```
L'element unique a pour père <xs:element name="sport">

    <xs:unique name="equipeUnique">
      <xs:selector xpath="*/p/e"/>
      <xs:field xpath="."/>
    </xs:unique>


Autre possibilité pour le sélecteur :  <xs:selector xpath="foot|basket/p/e"/>


Réponse erronnée
<xs:selector xpath=".//e"/>
Erreur car cela inclut à tort les éléments e des équipes jouant des matchs.
```

Question 3. Définir le schéma d'un match entre exactement deux équipes.

```
<xs:element name="match">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="e" minOccurs="2" maxOccurs="2"/>
    </xs:sequence>
  </xs:complexType>
```

```
</xs:element>
```

Question 4. Définir le type *typeEquipePoint* de l'element *e* contenu dans l'élément *match*, de telle sorte que l'élément *e* ait un attribut *points* dont la valeur est le nombre de points marqués par l'équipe pendant le match. Exemple de contenu valide :

```
<match> <e points="0">TFC</e> <e points="3">OM</e> </match>
```

```
  <xs:complexType name="typeEquipePoint">
     <xs:simpleContent>
       <xs:extension base="xs:string">
             <xs:attribute name="points" type="xs:string"/>
       </xs:extension>
     </xs:simpleContent>
  </xs:complexType>


Remarque : le type est ensuite utilisé pour définir un élément local :

<xs:element name="match">
    <xs:complexType>
      <xs:sequence>
     <xs:element name="e" type="typeEquipePoint" minOccurs="2" maxOccurs="2">
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>


Le type e est définit localement dans le contenu de l'élément match.
IL n'est pas en conflit avec l'autre type e contenu dans l'élément p pour l'annuaire
des équipes.
Rappel : en XML Schema on peut avoir plusieurs éléments de même nom mais de type
différent.
```