Master MIAGE – 2009/2010

PXML

vendredi 26 mars 2010

durée 2h documents autorisés

examen

Exercice 1 : Subtilités XPATH

Question 1 : Expliquez la signification des expressions suivantes. Parmi ces expressions lesquelles donneront le même résultat ?

1. `/item/livre[@titre="pxml" and position()=last()]`
2. `/item/livre[@titre="pxml"] [position()=last()]`
3. `/item/livre[position()=last()] [@titre="pxml"]`

Donnez un exemple pour lequel les trois expressions ne donneront pas toutes le même résultat.

Les expressions 1 et 3 retournent le dernier livre à condition que le titre de celui-ci soit `pxml`*, ces 2 expressions sont équivalentes. L'expression 2 retourne le dernier des livres dont le titre est* `pxml`*. Sur le fragment* `xml` *ci-dessous*

```
<item>
    <livre titre="pxml"/>
    <livre titre="lmxp"/>
</item>
```

les expressions 1 et 3 retournent une séquence vide contrairement à l'expression 2.

Question 2 : Quelle différence y a-t-il entre l'expression `//livre[titre="edition"]` et l'expression `//livre[titre=edition]` ? Donnez un exemple pour lequel le résultat est identique.

L'expression `//livre[titre="edition"]` *retourne tous les éléments* `livre` *qui ont un sous-élément* `titre` ***dont le contenu est égal au texte*** `"edition"`.
L'expression `//livre[titre=edition]` *retourne tous les éléments* `livre` *qui ont un sous-élément* `titre` ***et un sous-élément*** `edition` *dont les contenus sont égaux. Le résultat de ces deux expressions est malgré tout identique pour le fragment* `xml` *suivant :*

```
<livre>
    <titre>edition</titre>
    <edition>edition</edition>
</livre>
```

les expressions auraient d'ailleurs retourné aussi le même résultat (vide) pour le fragment `xml` *suivant :* `<livre/>`.

Question 3 : Même question pour `//livre[1]` et `/descendant::livre[1]` mais en donnant un document pour lequel on n'obtient pas le même résultat.

Dans le premier cas, la notation étendue correspondante est `/descendant-or-self::node()/livre[1]`. *On a donc les premiers fils éléments* `livre` *d'un des éléments du document. Dans le second cas on obtient le premier élément* `livre` *rencontré dans le document. Sur le fichier suivant,* `//livre[1]` *retourne 2 éléments* `livre` *(celui dont le contenu est* `un` *et celui dont le contenu est* `deux.un` *alors que l'expression* `/descendant::livre[1]` *ne retourne que l'élément* `livre` *dont le contenu est* `un`.

```
<bib>
    <livre>un</livre>
        <etagere>
            <livre>deux.un</livre>
            <livre>deux.deux</livre>
        </etagere>
</bib>
```

Exercice 2 : Immo-XSLT

On considère le fichier `maisons.xml` , donné en annexe, et qui contient les descriptions de plusieurs maisons (pièces, décoration, superficie...).

Question 1 : Écrivez une transformation `XSLT` qui, à partir de ce fichier `XML`, calcule, pour chaque maison, sa superficie totale. La sortie du programme sera un texte (`<xsl:output method="text"/>`) qui contient ces informations, sous la forme suivante :

```
Maison 1:
   superficie totale : 95 m2
Maison 2:
   superficie totale : 28 m2
Maison 3:
   superficie totale : 57.5 m2
```

```
<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="2.0">
    <xsl:output method="text"/>

    <xsl:template match="/">
        <xsl:apply-templates select="//maison"/>
    </xsl:template>

    <xsl:template match="//maison">
        Maison <xsl:value-of select="@id"/> :
        superficie totale : <xsl:value-of select="sum(.//@surface-m2)"/> m2
    </xsl:template>

</xsl:stylesheet>
```

Exercice 3 : Pas d'orchidée pour XQUERY

On considère les trois fichiers `xml` suivants, donnés en annexe.

`plant_catalog.xml` est un catalogue de plantes ;

`plant_families.xml` qui indique à quelle famille appartiennent certaines plantes ;

`plant_order.xml` est une commande de plantes.

Question 1 : Donnez un programme `XQUERY` qui produit à partir des fichiers `plant_catalog.xml` et `plant_families.xml.` un document `XML` en ajoutant dans chaque élément `PLANT` apparaissant dans `plant_catalog.xml` un élément `FAMILY` qui donne le nom de la famille à laquelle appartient la plante comme dans l'exemple ci-dessous :

```
<PLANT>
      <COMMON>Bloodroot</COMMON>
      <BOTANICAL>Sanguinaria canadensis</BOTANICAL>
      <ZONE>4</ZONE>
      <LIGHT>Mostly Shady</LIGHT>
      <PRICE>$2.44</PRICE>
      <AVAILABILITY>031599</AVAILABILITY>
      <FAMILY>Papaveraceae</FAMILY>
</PLANT>
```

```
<CATALOG>
{
    for $p in doc("plant_catalog.xml")//PLANT, $f in doc("plant_families.xml")//FAMILY
    where $f/SPECIES = $p/BOTANICAL
    return
    <PLANT>
    {
        $p/*
    }
        <FAMILY>{$f/NAME/text()}</FAMILY>
    </PLANT>
}
</CATALOG>
```

Question 2 : Donnez un programme `XQUERY` qui classe et regroupe les éléments `PLANT` du fichier `plant_catalog.xml` en fonction du contenu de leur élément `LIGHT` comme dans l'exemple ci-dessous :

```
<CATALOG>
   <LIGHT>
      <EXPOSURE>Mostly Shady</EXPOSURE>
      <PLANT>
         <COMMON>Bloodroot</COMMON>
         <BOTANICAL>Sanguinaria canadensis</BOTANICAL>
         <ZONE>4</ZONE>
         <PRICE>$2.44</PRICE>
         <AVAILABILITY>031599</AVAILABILITY>
      </PLANT>
      <PLANT>
         <COMMON>Columbine</COMMON>
         <BOTANICAL>Aquilegia canadensis</BOTANICAL>
```

```
            <ZONE>3</ZONE>
            <PRICE>$9.37</PRICE>
            <AVAILABILITY>030699</AVAILABILITY>
         </PLANT>
         (...)
         <EXPOSURE>Mostly Sunny</EXPOSURE>
         <PLANT>
            <COMMON>Marsh Marigold</COMMON>
            <BOTANICAL>Caltha palustris</BOTANICAL>
            <ZONE>4</ZONE>
            <PRICE>$6.81</PRICE>
            <AVAILABILITY>051799</AVAILABILITY>
         </PLANT>
      </LIGHT>
      (...)
   </CATALOG>
```

```
<CATALOG>
{
    for $l in distinct-values(doc("plant_catalog.xml")//LIGHT)
    return
    <LIGHT>
        <EXPOSURE>{$l}</EXPOSURE>
    {
        for $p in doc("plant_catalog.xml")//PLANT
        where $p/LIGHT=$l
        return
        <PLANT>
        {
            $p/*[not(name()="LIGHT")]
        }
        </PLANT>
    }
    </LIGHT>
}
</CATALOG>
```

Question 3 : Donnez un programme `XQUERY` qui réalise les 2 opérations des questions 1 et 2 en classant en outre les éléments `LIGHT` par ordre alphabétique du contenu des éléments `EXPOSURE` et en classant les éléments `PLANT` par ordre alphabétique du contenu des éléments `COMMON`.

```
<CATALOG>
{
    for $l in distinct-values(doc("plant_catalog.xml")//LIGHT)
    order by $l
    return
    <LIGHT>
        <EXPOSURE>{$l}</EXPOSURE>
```

```
        {
            for $p in doc("plant_catalog.xml")//PLANT ,
                $f in doc("plant_families.xml")//FAMILY
            where  $p/LIGHT=$l and $f/SPECIES = $p/BOTANICAL
            order by $p/COMMON
            return
            <PLANT>
            {
                $p/*[not(name()="LIGHT")]
            }
                <FAMILY>{$f/NAME/text()}</FAMILY>
            </PLANT>
        }
    </LIGHT>
}
</CATALOG>
```

Question 4 : Donnez un programme `XQUERY` qui calcule le montant total de la commande décrite dans `plant_order.xml` en donnant le résultat dans un élément `PRICE` comme dans l'exemple : `<PRICE>663.2</PRICE>`

```
<PRICE>
{
    sum(
        for $p1 in doc("plant_order.xml")//PLANT,
            $p2 in doc("plant_catalog.xml")//PLANT
        where $p1/COMMON=$p2/COMMON
        return number($p1/QUANTITY) * number(substring-after($p2/PRICE , "$"))
    )
 }
</PRICE>
```
