Stockage d'arbres XML et Interrogation XPATH et SQL

Information plus ou moins structurée

- une table : Stockage statique assez plat
- un arbre XML : Stockage dynamique en profondeur
- Lien entre les deux modes : les clés étrangères

Exemple d'arbres XML

```
<commande id="1">
    <lc id="1">c1 : ma première ligne</lc>
    <lc id="2">c1 :ma deuxième ligne</lc>
</commande>

<commande id="2">
    <lc id="1">c2 : ma première ligne</lc>
    <lc id="2">c2 : ma deuxième ligne</lc>
</commande>
```


Stockage de l'arbre dans Oracle

```
CREATE TABLE Commandes (
idCom number,
comm XMLTYPE, CONSTRAINT pk_commande PRIMARY KEY(idCom)
);
```


Insertion d'un arbre dans la base

- Oracle a créé la fonction XMLTYPE pour gérer les arbres XML.

```
INSERT INTO commandes VALUES
  (1,XMLTYPE ('<commande id="1">...</commande>'));

INSERT INTO commandes VALUES
  (2,XMLTYPE ('<commande id="2">...</commande>'));
```

- La fonction XMLTYPE convertit les balises en un contenu de type XMLTYPE

- Elle génère une erreur si l'information de la colonne comm n'est pas un arbre XML

La fonction UPDATEXML

- Une requête de mise à jour de lignes

```
UPDATE commandes SET comm =
    UPDATEXML(comm,
  '/commande/lc[@id="1"]/text()', 'two')
```

- Résultat
  Tous les arbres XML de la colonne Comm auront le texte des nœuds lc, dont l'attribut id vaut 1, remplacé par le texte 'two'. Dans notre cas, les deux lignes seront changées

La fonction UPDATEXML

- Une requête de mise à jour de lignes

```
UPDATE commandes SET comm =
    UPDATEXML(comm,
'/commande/lc[@id="1"]/text()', 'two')
WHERE idComm = 1
```

- Résultat
  L'arbre XML de la colonne Comm de la ligne dont la clé vaut 1 aura le texte des nœuds lc, dont l'attribut id vaut 1, remplacé par le texte 'two'.

Sélection d'un arbre dans la base

- La commande suivante permet d'obtenir le contenu entier d'un arbre XML

```
SELECT EXTRACT (comm,'/')
FROM commandes
```


Sélection d'un sous arbre dans la base

- La commande suivante permet d'obtenir les contenus de sous arbres d'une colonne XMLTYPE

```
SELECT EXTRACT
(comm,'commande/lc') FROM
commandes
```


Sélection d'un sous arbre dans la base

- Les commandes suivantes permettent d'obtenir une partie des arbres d'une colonne XMLTYPE

```
SELECT EXTRACT(comm, 'commande[lc/@id =
"1"]') FROM commandes
```

- Résultat

  Toutes les lignes de la table sont sélectionnées. Pour chacune, on affiche un arbre vide ou tout l'arbre s'il existe un noeud lc dont l'attribut id vaut 1

Sélection d'un sous arbre dans la base

- Les commandes suivantes permettent d'obtenir une partie des arbres d'une colonne XMLTYPE

```
SELECT EXTRACT(comm, 'commande[lc/@id =
"1"]') FROM commandes  where idComm=1
```

- Résultat

  On sélectionne la ligne dont la clé vaut 1.

  On affiche un arbre vide ou tout l'arbre s'il existe un noeud lc dont l'attribut id vaut 1

e**X**tensible **M**arkup **L**anguage : Le concept

- Un fichier html est un fichier à balise qui contient de l'information et la façon d'afficher cette information à l'aide d'un browser (ie, netscape, mozilla, ...).

- *Le concept XML vise essentiellement*
  - *à séparer l'information de sa présentation*
  - *à pouvoir accéder facilement à toute ou partie de cette information*

XML : Un exemple de fichier XML

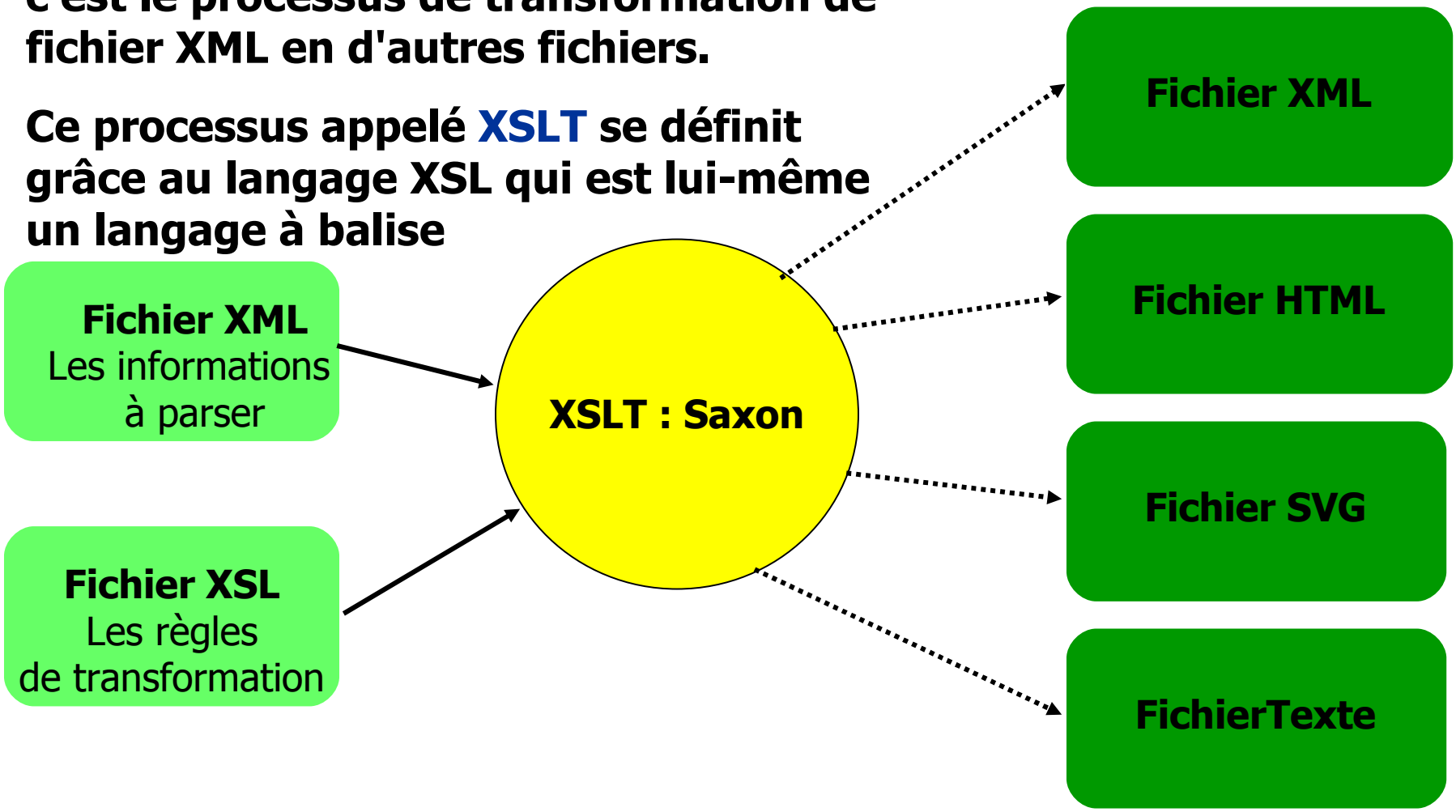
```
<etudiant prenom="Maude" nom="Eme" annee="ing2">
    <ou adresse="1 rue du bo" ville="Paris" cp="75000" />
    <matiere libelle="info">
        <appreciation>tu peux mieux faire</appreciation>
        <note epreuve="ds n° 1" valeur="10.5" />
        <note epreuve="ds n° 2" valeur="11" />
        <note epreuve="ds n° 3" valeur="12" />
    </matiere>
    <matiere libelle="comptabilité générale">
        <appreciation>belle prestation</appreciation>
        <note epreuve="ds" valeur="15" />
    </matiere>
</etudiant>
```

Vers une transformation
en un fichier HTML

XSL : e**X**tensible **S**tylesheet **L**anguage

Ce qui nous intéresse dans ce cours, c'est le processus de transformation de fichier XML en d'autres fichiers.

Ce processus appelé XSLT se définit grâce au langage XSL qui est lui-même un langage à balise



XSL : Un exemple de source XSL

```
<!– L'entête -->
<?xml version="1.0"
   encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL
   /Transform">
<xsl:output method="html" indent="yes"
   encoding="iso-8859-1" />
```


XSL : Un exemple de source XSL

```
<!- le patron du nœud etudiant -->
<xsl:template match="etudiant">
<html>
    <head>
    </head>
    <body>
    <h1><xsl:value-of select="@prenom" /><xsl:text>
    </xsl:text><xsl:value-of select="@nom" /></h1>
    <xsl:apply-templates select="ou" />
    </body>
</html>
</xsl:template>
```


XSL : Un exemple de source XSL

```
<!- le patron du nœud ou -->
<xsl:template match="ou">
<h2>
<xsl:value-of select="@adresse" /><br />
<xsl:value-of select="@cp" /><xsl:text>
    </xsl:text><xsl:value-of select="@ville" />
</h2>
</xsl:template>

<!- fin de la feuille xsl-->
</xsl:stylesheet>
```


XSLT : Un exemple de transformation

- Le fichier ci-dessous est la transformation XSLT des fichiers étudiés précédemment.

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
    charset=iso-8859-1">
  </head>
  <body>
    <h1>Maude Eme</h1>
    <h2>1 rue du bo<br>75000 Paris
    </h2>
  </body>
</html>
```

Vers le fichier XML

Vers le fichier XSL

XML : **D**ocument **T**ype **D**efinition

- Deux fichiers d'étudiants avec leur adresse et leurs notes dans les matières diffèrent dans leurs valeurs (nom, prénom, adresse et notes) mais pas dans leur sémantique

- On a deux façons de décrire cette sémantique
  - À travers une DTD : Document Type Definition.
  - A travers un schéma : description sous forme de classes. Un fichier XML associé à ce schéma est une instance.

- Nous nous intéresserons à une représentation sous forme de *DTD*.

XML : Un exemple de DTD (1)

```
<!ELEMENT etudiant(ou, matiere*)>
<!ATTLIST etudiant
    nom CDATA #REQUIRED
    prenom CDATA #REQUIRED
    annee (ing1|ing2|ing3|autre) "autre" #REQUIRED
    age CDATA #IMPLIED
    >
<!ATTLIST ou
   adresse CDATA #REQUIRED
   ville CDATA #REQUIRED
   cp CDATA #REQUIRED
   >
```


XML : Un exemple de DTD (2)

```
<!ELEMENT matiere(appreciation,note*) >
<!ATTLIST matiere
    libelle CDATA #REQUIRED
    >
<!ELEMENT appreciation (#PCDATA| attention)+ >
<!ELEMENT attention (#PCDATA)>
<!ATTLIST note
    epreuve CDATA #REQUIRED
    valeur CDATA #IMPLIED
    >
```


Le langage XSL

- Un source XSL est un fichier XML composé :
  - D'une entête
  - De patrons de nœuds composés eux-mêmes
    - De paramètres et variables locales
    - De structures conditionnelles
    - De structures d'aiguillages
    - De structures itératives portant sur les nœuds fils
  - De paramètres globaux

XSL : les paramètres et les variables

- On peut définir des paramètres avec l'instruction
  - <xsl:param name="nomParam">…</xsl:param>

- On peut définir des variables locales à un noeud avec l'instruction
  - <xsl:variable name="nomVar">…</xsl:variable>

- On peut récupérer le contenu d'une variable ou d'un paramètre avec l'expression $nomElement où nomElement est le libéllé de la variable ou du paramètre.

XSL : les attributs

- On peut accéder à un attribut d'un nœud (ou élément) avec l'expression @nomAttribut.
- On peut définir des listes d'attributs

```
<xsl:attribute-set name="nomListeAtt">
    <xsl:attribut name="nomAtt1">…</xsl:attribut>
    …
    <xsl:attribut name="nomAttn">…</xsl:attribut>
</xsl:attribute-set>
```

- Une liste d'attributs sert au processeur XSLT à fabriquer les attributs d'une balise.
- La syntaxe est

```
<nomBalise  xsl:use-attribute-set="nomListeAtt" ..>
```


XSL : Patron de nœud

- L'instruction xsl:template permet de définir pour un nœud ce que le processeur XSLT devra faire quand il devra traiter ce nœud.

- La syntaxe est

```
<xsl:template match="NomDuNoeud">
....
</xsl:template>
```

- Si un patron de nœud n'est pas défini, il vaut par défaut :

```
<xsl:template match="nomDuNoeud">
    <xsl:apply-templates />
</xsl:template>
```


XSL : L'appel d'un patron de nœud

- L'instruction xsl:apply-template permet de préciser au processeur XSLT quel patron de noeud il faut appliquer à un endroit précis du fichier XML où ce nœud peut être un fils du nœud courant.

- La syntaxe est
  - <xsl:apply-templates select="NomDuNoeudFils" />
  - <xsl:apply-templates /> pour tous les nœuds fils.

XSL : L'appel d'un patron de nœud

- L'instruction xsl:apply-template permet au concepteur de la feuille XSL de gérer les appels des fils du nœud courant.
- En particulier, des nœuds fils peuvent être dans un certain ordre dans le fichier XML et être traités (ou ne pas être traités) dans un autre ordre.

XSL : La boucle

- L'instruction xsl:for-each permet de préciser au processeur XSLT dans le traitement du nœud courant de traiter des nœuds fils particuliers sans nécessairement faire appel au patron de ces nœuds fils.

- La syntaxe est

  <xsl:for-each select="..." >

  ...

  </xsl:for-each>

- On peut imposer un ordre pour les différentes itérations de la boucle avec l'instruction

  <xsl:sort select="..." />.

XSL : La clause select

- Cette clause est utilisée dans les instructions xsl:apply-template et xsl:for-each. Elle porte sur des nœuds.
- On peut sélectionner ces nœuds avec une condition en utilisant la syntaxe suivante

select="nomDuNoeud[condition]"

XSL : Les tests

- L'instruction xsl:if permet de préciser au processeur XSLT un traitement conditionnel.

- La syntaxe est

```
<xsl:if test="…" />
…
</xsl:if>
```


XSL : Les tests

- La condition définie entre guillemets peut porter
  - Sur l'existence ou non de nœuds fils, l'existence ou non de variables, de paramètres et d'attributs.
  - Sur le contenu de de nœuds fils, l'existence ou non de variables, de paramètres et d'attributs.

XSL : Les tests

- Lors de l'exécution du processeur XSLT, on appelera contexte l'arbre défini par le nœud courant.

- La fonction count(node-set) renvoie le nombre de nœuds de l'ensemble de nœuds node-set passé en argument.

- La fonction position() renvoie la place du nœud courant dans le contexte de son père.

XSL : Les tests

- La fonction last() renvoie la taille du contexte du père du nœud courant.
- Les opérateurs booléens sont or, and et not(…)
- Les opérateurs de comparaison =, < , >, !=
- La fonction node() indique tous les nœuds fils du nœud courant

XSL : Les tests

```
<!- on teste si le nœud courant est le premier fils
   de son père-->
<xsl:if test="position() = 1">

<!- on teste si le nœud courant est le dernier fils
   de son père-->
<xsl:if test="position() = last()">

<!- on teste si la variable v1 est inférieure v2 -->
<xsl:if test="$v1 &lt; $v2">
```


XSL : Les tests

```
<!- on teste si le nœud courant à un fils
   nommé theFils-->
<xsl:if test="theFils">

<!- on teste si le nœud courant à un attribut
   nommé theAttribut-->
<xsl:if test="@theAttribut">
```


XSL : Les tests

```
<!- on teste si le nœud courant à un attribut
   nommé theAttribut qui vaut val -->
<xsl:if test="@theAttribut = 'val' ">

<!- on teste si le nœud courant à une
   variable ou un paramètre nommé
   theParVal qui vaut val -->
<xsl:if test="$theParVal = 'val' ">
```


XSL : Les aiguillages

- L'instruction <xsl:choose> permet de gérer des branchements conditionnels à plusieurs branchements
- La syntaxe est

```
<xls:choose>
   <xsl:when test="...">...</xsl:when>
     ...
     <xsl:otherwise>...</xsl:when>
</xls:choose>
```