

XML

Didier Le Foll

Sommaire

Généralités

Dialectes XML

Anatomie d'un document XML

Document « bien formé »

Processeurs (parsers) XML

La DTD (Document Type Definition)

Document valide

Les espaces de noms (namespace)

Les schémas (XML Schema)

Affichage de documents XML

- XML et CSS
- Conversion en HTML avec XSLT

Le langage XQuery

XML et bases de données

Généralités

- Le langage **XML** (*eXtensible Markup Language*) a été créé en 1998 par un groupe de travail du **W3C** (organisme de normalisation du Web).
- Il est une **simplification du SGML**, langage de balisage utilisé dans l'édition.
- Le but était de définir un **langage simple de description de tout type de document**, de telle sorte qu'ils puissent être facilement traités par ordinateur et facilement échangés.

Généralités (2)

- La norme XML 1.0 a été publiée en 1998.
- La norme XML 1.1 a été publiée en 2004.
Elle apporte des améliorations dans le support des différentes versions d'Unicode.
- XML 1.0 est actuellement la version la plus répandue.

Dialectes XML

- XML est une **syntaxe générique et extensible**. Il permet de structurer une grande variété de contenus, car son “langage” (vocabulaire et grammaire) peut être redéfini.
- Chaque secteur peut donc développer son langage de balisage spécifique, à base de XML (« **dialectes** » XML). Il en existe des centaines.
- Les plus connus : XHTML, XSLT, SOAP, SVG, MathML, RSS, Atom, XAML, OOXML.

Caractéristiques générales

- Ecrit en **mode texte** (lisible par un humain aussi bien que par un ordinateur).
- **Balises en chevrons**, avec obligation de fermeture : `< >` et `</ >`
- Les **mots clés** entre les chevrons ne sont **pas définis par la norme XML**, mais par le dialecte.
- **Structure arborescente** : un document XML a toujours un élément « racine ».

Anatomie d'un document XML(1)

Un document XML est composé de deux parties :
le *prologue* et l'*élément document*.

- Le **prologue** : il contient obligatoirement (1ère ligne) la **déclaration XML**. Ex: `<?xml version="1.0"?>`
Il peut aussi contenir d'autres éléments, notamment :
 - une **DTD** (*Document Type Definition*) qui définit le type et la structure du document,
 - une ou plusieurs **instructions de traitement** fournissant des informations pour l'application qui traite le fichier XML.
- L'**élément document** (ou *élément racine*): il peut contenir d'autres éléments emboîtés, qui constituent la structure arborescente du document XML.

Anatomie d'un document XML(2)

Anatomie d'un élément :

- il commence par une **balise (tag) d'ouverture**, ex :
`<livre>` et se termine par une **balise de fermeture**, ex :
`</livre>`, sauf les éléments dits « vides » (ex: `<hr />`).
Le **nom** des balises de début et de fin (ici `livre`)
spécifient le « **type** » de l'élément (sensible à la casse).
- il peut contenir :
 - d'autres éléments (hiérarchie récursive), dits « **sous-éléments** »,
 - du **texte** (données caractère),
 - un contenu **mixte** texte et sous-éléments,
 - des sections CDATA (texte contenant n'importe quel caractère),
 - des références d'entités, des instructions de traitement, des commentaires.

Anatomie d'un document XML(3)

- un élément peut être porteur d'**attributs** : dans la balise d'ouverture ou bien dans un élément vide, peut apparaître une liste d'attributs, sous la forme *nom_attr="valeur"*. Ex :

```
<livre categorie="fiction" format="poche">
```

Ex. pour un élément vide :

```
<couverture image="couverture.gif"/>
```

La valeur d'un attribut est un texte sans élément (ni autres nœuds). Un attribut est unique : la répétition d'un attribut de même nom sur le même élément provoquera une erreur. L'ordre des attributs n'est pas significatif.

Exemple fictif

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- '''Commentaire''' -->
<élément-document xmlns="http://exemple.org/" xml:lang="fr">
  <élément>Texte</élément>
  <élément>élément répété</élément>
  <élément>
    <élément>Hiérarchie récursive</élément>
  </élément>
  <élément>Texte avec<élément>élément</élément>mêlé</élément>
  <élément/><!-- élément vide -->
  <élément attribut="valeur"></élément>
</élément-document>
```

Document « bien formé »

- Un document est « bien formé » s'il respecte certaines règles de syntaxe, dont :
 - il a un et un seul élément racine. Tous les autres éléments sont emboîtés dans celui-ci,
 - les éléments doivent être correctement emboîtés,
 - chaque élément doit avoir une balise de début et de fin (sauf éléments vides, mais avec /> en fin),
 - le nom placé dans la balise de début doit correspondre exactement à celui de la balise de fin (attention! sensibilité à la casse).

Processeurs (*parsers*) XML

- Ce sont les « *parseurs* » XML qui vérifient le caractère « bien formé » d'un document. Si ce n'est pas le cas, le « *parseur* » s'arrête en erreur. Les plus courants sont :
 - *MSXML* parseur de Microsoft, accessible notamment en JavaScript sur le navigateur *Internet Explorer*,
 - *libxml2*, le parseur installé sous les principales distributions de Linux (utilisé aussi par PHP),
 - *Xerces*, le parseur d'Apache,
 - *expat*, le parseur installé sur *Mozilla Firefox*.

La DTD (*Document Type Definition*)

- La DTD liste les contraintes qui spécifient quels sont les éléments autorisés, comment ils peuvent s'imbriquer entre eux, les attributs autorisés pour chaque élément, etc...
- La **déclaration de DTD** est placée dans le prologue (n'importe où après la première ligne).
- La DTD est généralement **externe**, c'est à dire dans un autre fichier. La déclaration de DTD reprend le nom de l'élément racine, et donne le nom de l'URL ou du fichier contenant la DTD. Ex :

```
<!DOCTYPE inventaire SYSTEM "livres.dtd">
```

Ici, *inventaire* indique que l'élément racine doit être de type *inventaire*. Le mot suivant doit être SYSTEM ou PUBLIC. Le fichier contenant la DTD est *livres.dtd*.

Syntaxe de la DTD (1)

Une DTD contient principalement :

- Des déclarations de **types d'éléments**. Ex:

```
<!ELEMENT titre (#PCDATA)>
```

C'est une déclaration d'un élément nommé *titre*, qui n'est autorisé à contenir que des données caractères (aucun élément enfant n'est autorisé).

Un élément peut être de type: texte (mot-clé #PCDATA, cf. ex. ci-dessus), élément pur (il contient des éléments enfants), mixte (éléments enfants+texte), vide (mot-clé EMPTY), ou libre (mot-clé ANY).

- Des déclarations de **listes d'attributs** : Ex:

```
<!ATTLIST livre editeur CDATA #REQUIRED lang CDATA #IMPLIED>
```

C'est une déclaration de liste d'attributs pour un élément nommé *livre* :
attribut *editeur* de type "texte", obligatoire, attribut *lang*, "texte", facultatif.

Un attribut peut être de type : texte (mot-clé CDATA, cf. ex. ci-dessus), énumération (ex: `lang (fr|en|de)`), identifiant (mot-clé ID), référence(s) d'identifiant (mot-clé IDREF ou IDREFS),...

Syntaxe de la DTD (2)

- Les déclarations d'**éléments contenant d'autres éléments**.

Ex:

```
<!ELEMENT livre (titre, auteur+, pages)>
```

C'est une déclaration d'un élément nommé *livre*, qui n'est autorisé à contenir que des éléments (dans l'ordre) *titre*, *auteur* (opérateur + : un ou plusieurs) et *pages*.

Les opérateurs peuvent être : « , » (mise en séquence), « | » (choix), « ? » (0 ou 1 occurrence), « * » (0 à N occurrences), « + » (1 à N occurrences).

Syntaxe de la DTD (3)

- Exemple complet de fichier DTD:

```
<!ELEMENT inventaire (livre*)>
<!ELEMENT livre (titre, auteur+, pages)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT auteur (#PCDATA)>
<!ELEMENT pages (#PCDATA)>
<!ATTLIST livre
    isbn ID #REQUIRED
    editeur CDATA #REQUIRED
    lang CDATA #IMPLIED>
```


Document valide

- Un document XML **valide** est un document bien formé qui respecte en plus les deux exigences suivantes :
 - le prologue du document comporte une **déclaration de type de document**, qui indique une DTD. Ex :

```
<!DOCTYPE inventaire SYSTEM "livres.dtd">
```
 - le reste du document se conforme à la structure définie dans la DTD.

Exemple de document valide

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE inventaire SYSTEM "livres.dtd">
<!--Nom de fichier : Inventaire.xml -->
<inventaire>
  <livre isbn="2-212-11690-X" editeur="Eyrolles"
    lang="fr">
    <titre>Systèmes de bases de données</titre>
    <auteur>Thomas Conolly</auteur>
    <auteur>Carolyn Begg</auteur>
    <pages>1429</pages>
  </livre>
  <livre isbn="1-56592-691-9" editeur="O'Reilly"
    lang="en">
    <titre>Building Oracle XML Applications</titre>
    <auteur>Steve Muench</auteur>
    <pages>792</pages>
  </livre>
</inventaire>
```

Les espaces de noms (*namespace*) (1)

- Les espaces de noms permettent de **mélanger plusieurs vocabulaires** (dialectes) au sein d'un même document. Le but est de ne pas ré-inventer un vocabulaire pour chaque besoin, mais de réutiliser les vocabulaires existants.
- Chaque élément ou attribut appartient à un espace de noms qui détermine le vocabulaire dont il est issu. Cette appartenance est marquée par la présence dans le nom d'un **préfixe** associé à l'espace de noms. Ex :

`<dc:title>` L'espace de noms est ici «dc» (*Dublin Core*)

- La **déclaration d'espace de noms** se fait, dans un élément, par un pseudo-attribut de forme *xmlns:préfixe* dont la valeur est une URL. La portée de la déclaration est celle de l'élément. Ex:

```
<inventaire xmlns:dc="http://purl.org/dc/elements/1.1/">
```

Les espaces de noms (*namespace*) (2)

- Un **espace de noms par défaut** est associé au préfixe vide. Son utilisation permet d'alléger l'écriture des documents XML en évitant de mettre un préfixe aux éléments les plus fréquents.

Ce « *namespace* » par défaut est spécifié par un pseudo-attribut de nom *xmlns*. Dans la portée de ce « *namespace* », les éléments ou attributs dont le nom n'est pas préfixé font partie de l'espace de noms par défaut. Ex :

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:mml="http://www.w3.org/1998/Math/MathML">
<head><title>MathML</title></head>
<body><h1>Racine de 2 en MathML</h1>
      <mml:math>
        <mml:msqrt><mml:mn>2</mml:mn></mml:msqrt>
      </mml:math>
</body></html>
```

Les schémas (*XML Schema*) (1)

- Les **schémas XML** permettent, comme les DTD, de définir des **modèles de documents**. Il est ensuite possible de vérifier qu'un document donné est valide pour un schéma. Les schémas ont été introduits pour combler certaines lacunes des DTD.
- **XML Schema** est une norme W3C, aussi connue sous le nom **XSD** (*XML Schema Definition*). Il s'agit d'un dialecte XML.
- Le schéma est défini dans un fichier externe au document. Sa structure globale est la suivante :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <!-- Déclarations d'éléments, d'attributs et
         définitions de types -->
    ...
</xsd:schema>
```

Les schémas (*XML Schema*) (2)

- Un schéma est un ensemble de **composants de schéma**, dont :
 - la déclaration d'**éléments**,
 - la déclaration d'**attributs**,
 - la définition de **types simples** : prédéfinis ou créés par l'utilisateur (en général par dérivation des types prédéfinis),
 - la définition de **types complexes** : créés pour définir des éléments constitués d'attributs et d'autres éléments.
- Les **types simples prédéfinis**. Ils sont très nombreux :
 - types numériques : `xsd:boolean`, `xsd:int`, `xsd:float`, `xsd:double`,...
 - types chaînes et noms : `xsd:string`, `xsd:token`, `xsd:Name`, `xsd:language`, `xsd:ID`, `xsd:IDREF`,...
 - types date et heure : `xsd:time` (*hh:mm:ss[.sss]*), `xsd:date` (*YYYY-MM-DD*), `xsd:dateTime`,...

Les schémas (*XML Schema*) (3)

- Dans le fichier XML, le schéma à utiliser est indiqué par un attribut (*schemaLocation* ou *noNamespaceSchemaLocation*) dans l'élément racine. Ces deux attributs se trouvent dans l'espace de noms des instances de schémas identifié par l'URI *http://www.w3.org/2001/XMLSchema-instance*.

L'attribut *noNamespaceSchemaLocation* est utilisé lorsque le document n'utilise pas d'espace de noms, l'attribut *schemaLocation* est utilisé dans le cas contraire. Ex :

```
<inventaire  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:noNamespaceSchemaLocation="livres.xsd">
```

Les schémas (*XML Schema*) (4)

- Exemple complet de fichier XSD (équivalent DTD p.16):

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" > ①
  <xsd:element name="inventaire"> ②
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="livre" minOccurs="0" maxOccurs="unbounded"/> ③
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="livre"> ④
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="titre" type="xsd:string"/>
        <xsd:element name="auteur" type="xsd:string" ⑤
          maxOccurs="unbounded"/>
        <xsd:element name="pages" type="xsd:positiveInteger"/>
      </xsd:sequence>
      <xsd:attribute name="isbn" type="xsd:ID" use="required"/> ⑥
      <xsd:attribute name="editeur" type="xsd:string" use="required"/>
      <xsd:attribute name="lang" type="xsd:language" use="optional"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```


Les schémas (*XML Schema*) (5)

- Légende de la page précédente :
 - 1 Élément racine *xsd:schema* avec la déclaration de l'espace de noms des schémas XML associé au préfixe *xsd*.
 - 2 Déclaration de l'élément *inventaire* de type complexe.
 - 3 Référence à l'élément *livre* dans la définition d'*inventaire*. Il peut y avoir 0 (*minOccurs*) à N (*maxOccurs*) livres. Par défaut *minOccurs* et *maxOccurs* valent 1.
 - 4 Début de la définition de l'élément *livre*, de type complexe. Il est constitué :
 - des éléments *titre*, *auteur* et *pages* (dans cet ordre car *xsd:sequence*). L'élément *auteur* peut être répété (*maxOccurs*=illimité).
 - des attributs *isbn* (obligatoire, de type prédéfini *xsd:ID*), *editeur* (obligatoire) et *lang* (facultatif, de type prédéfini *xsd:language*).

Affichage de documents XML

Il existe plusieurs moyens d'afficher des documents XML dans un navigateur web :

- à l'aide de feuilles de styles CSS,
- à l'aide de feuilles de styles XSL,
- à l'aide de scripts JavaScript/DOM,
- à l'aide de programmes XQuery.

XML et CSS (1)

- Le langage **CSS** (*Cascading Style Sheet*) sert à décrire la présentation des documents HTML et **XML**.
- Les deux niveaux de norme largement acceptés par les navigateurs sont **CSS1** (1996) et **CSS2.1** (2007). CSS3 (du moins certains modules) est une norme depuis 2011, mais est encore peu présente dans les navigateurs. CSS4 est en développement depuis 2009.
- Fonctionnalités CSS selon les navigateurs (2012)

Navigateur	IE	Mozilla	Safari/Chrome	Konqueror	Opera
CSS1	Oui	Oui	Oui	Oui	Oui
CSS2.1	Oui	Oui	Oui	Oui	Oui
CSS3	Partiel	Partiel	Partiel	Partiel	Majorité

XML et CSS (2)

- Ex. de fichier CSS (*livres.css*):

livre

```
{display:block;  
  margin-top:12pt;  
  font-size:10pt;}
```

titre

```
{font-weight:bold}
```

auteur

```
{font-style:italic}
```

XML et CSS (3)

- Si on rajoute au fichier XML d'exemple (p.18) la ligne suivante dans le prologue:

```
<?xml-stylesheet type="text/css" href="livres.css"?>
```

on obtient en affichage:

Systèmes de bases de données *Thomas Conolly Carolyn Begg* 1429

Building Oracle XML Applications *Steve Muench* 792

Conversion en HTML avec XSLT (1)

- *XSL (eXtensible Stylesheet Language)* est le langage de description de feuilles de style du W3C associé à XML. XSLT est un sous-ensemble de XSL (XSLT 1.0, norme de 1999).
- XSLT est le langage de description des règles de transformation d'un arbre XML source en un arbre XML résultat. XSLT est lui-même un dialecte XML.
- Une feuille de style XSLT est bien plus puissante et plus souple qu'une feuille de style CSS. Notamment, XSLT permet de redéfinir l'ordre des éléments, afficher les valeurs d'attributs, réaliser des affichages conditionnels,...
- XSLT s'appuie sur XPath, qui est un langage (non XML) pour localiser une partie d'un document XML. XSLT 1.0 utilise XPath 1.0




Conversion en HTML avec XSLT (2)

- Le langage XSLT permet de programmer selon deux modes (qu'on peut mélanger) :
 - **Déclaratif** : on donne des **règles** (*templates*), et le processeur XSLT fait le travail.
 - **Impératif** : on utilise les structures habituelles d'un langage de programmation (tests, boucles, variables).
- De base, c'est cependant un **langage à base de règles**. Il est indispensable de comprendre ce fonctionnement, car XSLT définit des règles implicites (règles par défaut) :
 - **Règle 1** : si aucune règle ne s'applique à un noeud (y compris la racine), XSLT recherche des règles pour ses nœuds fils (récursivement jusqu'aux feuilles).
 - **Règle 2** : lorsqu'un élément contient du texte (#PCDATA), ce texte est, à défaut de règle, recopié dans le document final.

Conversion en HTML avec XSLT (3)

- Ex de fichier XSLT (orienté «impératif»): *livres.xsl*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html><head><title>Inventaire des livres</title></head>
<body><h2>Inventaire des livres</h2>
<xsl:for-each select="inventaire/livre">
  <p><b><xsl:value-of select="titre" /></b>
    (<xsl:value-of select="pages" /> pages) par <i>
<xsl:for-each select="auteur">
  <xsl:value-of select="." />
  <xsl:if test="position() != last()">,</xsl:if>
</xsl:for-each></i>Ed.<xsl:value-of select="@editeur"/>
</p></xsl:for-each>
</body></html>
</xsl:template>
</xsl:stylesheet>
```

	Partie purement XSL
	Partie fixe HTML
	Expressions XPath

Conversion en HTML avec XSLT (4)

- Le même exemple (orienté «déclaratif»): *livres.xsl*

Les 3 premières et la dernière ligne sont identiques (prologue et marquage de l'élément racine `<xsl:stylesheet>`)

```
<xsl:template match="/"> Règle 1
  <html><head><title>Inventaire des livres</title></head>
  <body><h2>Inventaire des livres</h2>
  <xsl:apply-templates/>
  </body></html>
</xsl:template>
<xsl:template match="livre"> Règle 2
  <p><b><xsl:value-of select="titre"/></b>
  (<xsl:value-of select="pages"/> pages) par <i>
  <xsl:apply-templates select="auteur"/></i>
  Ed. <xsl:value-of select="@editeur"/></p>
</xsl:template>
<xsl:template match="auteur"> Règle 3
  <xsl:value-of select="."/>
  <xsl:if test="position() != last()">, </xsl:if>
</xsl:template>
```

■ Partie purement XSL
■ Partie fixe HTML
■ Expressions XPath

Conversion en HTML avec XSLT (5)

- Si on rajoute au fichier XML d'exemple (p.18) la ligne suivante dans le prologue:

```
<?xml-stylesheet type="text/xsl" href="livres.xsl"?>
```

on obtient en affichage:

Inventaire des livres

Systèmes de bases de données (1429 pages) par *Thomas Conolly, Carolyn Begg* Ed.Eyrolles

Building Oracle XML Applications (792 pages) par *Steve Muench* Ed.O'Reilly

Le langage *XQuery* (1)

- *XQuery* (ou *XML Query*) est une norme W3C depuis 2007 (*XQuery 1.0*). Attention! *XQuery* n'est pas un dialecte XML.
- On dit souvent que *XQuery* est le « SQL de XML ». Il permet :
 - d'extraire des informations (*requêtes*) d'un document XML, d'une collection de documents, ou d'une BD XML,
 - d'effectuer des calculs à partir des informations extraites et de reconstruire de nouveaux documents ou fragments XML (*transformations*). En ce sens, c'est un concurrent de XSLT.
- *XQuery*, comme XSLT, est très lié à *XPath* : *XQuery 1.0* utilise *XPath 2.0*. Les types de données sont issus de *XML Schema*.
- Il n'y a pas de norme pour l'*extension* à donner à un fichier contenant un programme *XQuery* : on conseille *.xq*, *.xqy* ou *.xquery*, mais on peut trouver aussi *.xql* ou *.xqm*.
- La version 3.0 de *XQuery* est en cours d'élaboration par le W3C.

Le langage *XQuery* (2)

- L'idée initiale de *XQuery* est la **similitude entre une BD relationnelle et une collection de données XML** (qui peut être stockée dans des fichiers ou dans une BD). Le langage *XQuery* veut être pour XML l'équivalent de *SQL* pour le relationnel.
- Le *W3C* avait envisagé la possibilité d'utiliser *SQL* directement, mais les différences entre la structure des données XML et celle des données relationnelles sont trop importantes :
 - 2 dimensions pour le modèle relationnel (lignes/colonnes), profondeur non prévisible pour les données XML (imbrication),
 - l'homogénéité des données relationnelles permet de les décrire avec des métadonnées externes (catalogue), l'hétérogénéité des données XML oblige à disperser des métadonnées (balises) dans tout le document,
 - il est nécessaire de respecter la notion d'ordre des données en XML, et pas dans le modèle relationnel.




Le langage *XQuery* (3)

- Une partie des structures de base du langage *XQuery* s'inspirent de SQL, particulièrement les expressions dites "*FLWOR*" (prononcer "*flower*") pour *For. Let. Where. Order by. Return*, proches du SQL *Select from. Where. Order by*. *XQuery* permet aussi les *jointures* (internes et externes) et le *group by* (en *XQuery* 3.0).
- Il existe d'autres structures telles que *if* et *typeswitch* qui peuvent se composer avec le *FLWOR*.
- Les structures et fonctions XPath sont les mêmes qu'en XSLT.
- Un peu comme en XSLT, un script *XQuery* peut se composer de *parties fixes* (les « *constructeurs* »), contenant des balises, envoyées directement en sortie, et de *parties interprétées*, entre *{ et }*, contenant des instructions *XQuery*.

Le langage *XQuery* (4)

- Ex. de programme XQuery: *livres.xqy*

```
<html><head><title>Inventaire des livres</title></head>
<body><h2>Inventaire des livres</h2>
{
for $livre in doc("livres.xml")/inventaire/livre
  return
    <p>
      <b>{data($livre/titre)}</b>
      ({data($livre/pages)} pages) par <i>
        {for $auteur in $livre/auteur[position() != last()]}
          return (data($auteur), ', ')}
        {data($livre/auteur[position() = last()])}</i>
      Ed. {data($livre/@editeur)}
    </p>
}
</body></html>
```

	Partie purement XQuery
	Partie fixe HTML
	Expressions XPath

Le langage *XQuery* (5)

- Aucun navigateur (en 2012) n'intègre nativement un processeur *XQuery*, mais il existe des extensions (ex: *xqib*, *XqUSEme*).
- On peut utiliser un **interpréteur *XQuery*** externe qui prend en entrée un (des) fichier(s) XML et génère un fichier XML (ou HTML) en sortie. Quelques outils connus : *saxon*, *zorba*, *xqilla*.

Par ex., avec *saxon* et notre fichier *.xqy* de la p. précédente :

```
C:\Program Files\Saxonica\SaxonHE9.4N\bin\Query.exe  
    livres.xqy -o:livres.html !encoding=iso-8859-1
```

on obtient à l'affichage du fichier résultat *livres.html* :

Inventaire des livres

Systèmes de bases de données (1429 pages) par *Thomas Conolly, Carolyn Begg* Ed.Eyrolles

Building Oracle XML Applications (792 pages) par *Steve Muench* Ed.O'Reilly

Le langage *XQuery* (6)

- *XQuery* (de base) est simplement un langage d'interrogation (qui correspond au « *select* » du SQL). Ses concepteurs (W3C) lui ont ajouté deux extensions normalisées (depuis mars 2011) :
 - *XQuery Update Facility* (*XQUF*), qui permet des mises à jour directes sur un ou plusieurs documents XML (équivalent des « *insert* », « *delete* », « *update* » du SQL).
 - *XQuery Full Text* (*XQFT*), qui permet des recherches textuelles élaborées (contenus d'éléments ou d'attributs) sur un ou plusieurs documents XML (existe en SQL, mais non-standard : syntaxe différente selon les SGBD).
- Peu d'outils ont, en 2012, implémenté ces deux extensions :
 - *saxon EE* (*Enterprise Edition*) permet *XQuery Update*,
 - *zorba* permet *XQuery Update* et *Full Text*,
 - des bases XML « natives » (cf. p.44) : *BaseX*, *eXist*,...

XML et bases de données (1)

- Dans tout le cours qui précède un document XML a toujours été évoqué en tant que fichier (isolé ou appartenant à un ensemble de fichiers). Mais un document XML peut aussi être stocké dans une base de données, qui peut être :
 - **base compatible XML** (« *XML enabled* ») : il s'agit en général d'une base de données relationnelle ayant des possibilités de stockage de données sous forme XML,
 - **base native XML** (« *XML native* »), conçue spécifiquement pour stocker des documents XML.
- En théorie, on distingue deux grands types de contenus XML :
 - contenu **orienté « données »** (« *data centric* »), caractérisé par une structure XML assez régulière : se stocke bien dans une BD relationnelle compatible XML,
 - contenu **orienté « document »** (« *document centric* »), à structure plus irrégulière : nécessite plutôt une BD native XML.

XML et bases de données (2)

- Tous les « grands » **SGBDR**, à des degrés divers, **supportent XML**, mais le niveau de support varie beaucoup. On distingue 3 niveaux :
 - stockage du XML en bloc, dans une **colonne de type « CLOB »**,
 - **« éclatement » (« shredding »)** d'un schéma XML dans des tables et colonnes (schéma relationnel) : stockage « structuré »,
 - stockage du XML dans une(des) colonne(s) de **type normalisé XML**, permettant l'utilisation en SQL de requêtes et fonctions spécifiques. Ce type et ces fonctions sont définis par la norme SQL:2003-14 (puis 2006, 2008 et 2011) : cette partie de la norme est appelée SQL/XML.
- Conformité de quelques **SGBDR** à la **norme ISO SQL/XML:2006** et support XQuery (adapté de <http://en.wikipedia.org/wiki/SQL/XML>, 2012) :

SGBD	Oracle	MS SQL Server	mySQL	PostgreSQL
Version	11g R1	2008	5.1.30	9.1
XML Datatype	Partiel	Elevé	Non	Partiel
Prédicats SQL/XML	Elevé	Partiel	Non	Partiel
Fonctions SQL/XML	Elevé	Partiel	Bas	Elevé
Support XQuery	Oui	Partiel	Non	Non

Stockage de XML dans une BD relationnelle (ex : *Oracle*)

XML data: employee_2002.xml

```
...  
<employee>  
  <first_name>Scott</first_name>  
  <last_name>Tiger</last_name>  
  <email>scott.itger@oracle.com</email>  
  ...  
  <hire_date>040402</hire_date>  
  ...  
  <department_id>1234</department_id>  
</employee>  
...
```

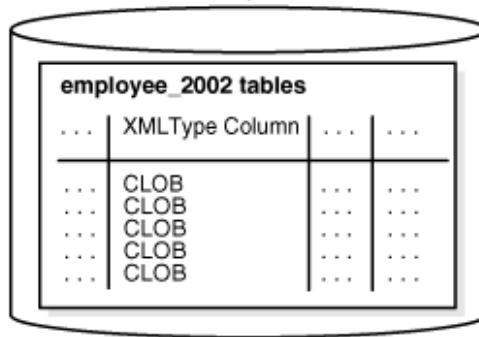
XML schema definition: employee.xsd

```
...  
<sequence>  
  <element name="first name" type="string"/>  
  <element name="last name" type="string"/>  
  <element name="email" type="string"/>  
  ...  
  <element name="hire_date" type="date"/>  
  ...  
  <element name="department_id" type="integer"/>  
</sequence>  
...
```

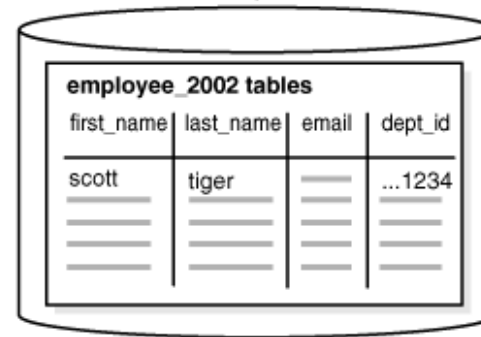
Create
XMLType
Table

Store as CLOB

Structured
Storage



Here the whole XML document or parts of it are stored in CLOBs in tables.



Here the XML elements are mapped to columns in tables.

XML et bases de données (3)

- Les **bases de données natives XML** sont des SGBD conçus spécifiquement pour stocker des documents XML :
 - le **document** (XML) est l'**unité fondamentale de stockage** (logique), au même niveau qu'une ligne d'une table dans une base de données relationnelle ,
 - la « **collection** » est un groupement logique de documents, au même niveau qu'une table dans une base de données relationnelle,
 - ils supportent au moins un **langage de requête** adapté (souvent *XQuery*), comprenant des instructions d'interrogation, mais aussi de mise à jour,
 - il n'y a aucune contrainte quand au mode de stockage physique sous-jacent : ça peut être du relationnel, du hiérarchique, ... ou tout système propriétaire.
- Caractéristiques de quelques **bases natives XML**, notamment support de XQuery et XSLT (source: http://en.wikipedia.org/wiki/XML_database, 2012) :

XML Database	License	XQuery 3.0	XQuery Update	XQuery Full Text	XSLT 2.0
BaseX	BSD License	Partial	Yes	Yes	Yes
eXist	LGPL License	Partial	Partial	Proprietary	Yes
MarkLogic Server	Commercial	Partial	Proprietary	Proprietary	Yes
Sedna	Apache License	No	Yes	Yes	No