DONNEES SEMI-STRUCTUREES

CHAPITRE 2:

NOYAU XML

2023-2024

II.1 Introduction à XML

II.1 Introduction à XML

The World Wide Web Consortium (W3C)

- Une communauté internationale qui travaille pour développer le web
- Dirigé par Tim Berners-Lee
- 440 membres (Adobe, Cisco, Facebook, Google, Microsoft, ...).

Le langage XML (Extensible Markup Language) Il s'agit d'un langage de balisage structuré et extensible destiné pour la description, le stockage et le transfert de données.

- XML a été conçu pour faciliter la mise en œuvre et l'interopérabilité avec SGML (très compliqué) et HTML (n'est pas extensible).

II.1 Introduction à XML

Pourquoi XML ?

- Structurer des données,
- Modéliser des données
  - Document Type Definitions (DTD)
  - Schémas XML (XML Schema)
- Publier des informations
  - Mise en forme CSS
  - Transformations XSL
- Archiver des données
  - Bases de Données XML

II.1 Introduction à XML

Pourquoi XML ? Intermédiaire entre des langages différents en programmation

II.1 Introduction à XML

La différence entre XML et HTML

- XML décrit la structure de données alors que HTML permet l'affichage de données.
- Les balises HTML sont prédéfinies (limitées) alors que les balises XML sont extensibles selon le besoin de l'utilisateur.
- XML ne remplace pas HTML mais ils se complètent car dans la plus part des application web, XML est utilisé pour le stockage et transfert de données alors que HTML est utilisé pour l'affichage.

II.1 Introduction à XML

La différence entre XML et HTML (exemple):

Code XML (Atelier1_Exemple1.xml)	Code HTML (Atelier1_Exemple1.html)
<BIBLIOTHEQUE> <LIVRE> <TITRE>titre livre 1</TITRE> <AUTEUR>auteur 1</AUTEUR> <EDITEUR>editeur 1</EDITEUR> </LIVRE> <LIVRE> <TITRE>titre livre 2</TITRE> <AUTEUR>auteur 2</AUTEUR> <EDITEUR>editeur 2</EDITEUR> </LIVRE> <LIVRE> </BIBLIOTHEQUE>	 titre livre 1 auteur 1 <u>editeur 1</u> titre livre 1 auteur 1 <u>editeur 1</u>

II.1 Introduction à XML

La différence entre XML et HTML (exemple):

Que remarquez vous?

- Le code XML permet de décrire la structure des données d'une bibliothèque.
- Le code HTML permet de spécifier l'affichage de données
- Le code XML comporte de nouvelles balises: <livre>, <auteur>, etc.
- Le code HTML ne comprend que des balises prédéfinies: <p>, <b>, etc.
- Le code XML décrit une structure arborescente.

II.2 Structure XML de base

II.2 Structure XML de base

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<livre lang="fr">
  <titre> Genie Logiciel </titre>
  <anneepub> 2023 </anneepub>
  <prix>2500</prix>
</livre>
```

<?xml version="1.0" encoding="ISO-8859-1" ?> Déclaration de la version de XML utilisée dans le document. Et les format des caractères utilisés.

<livre> … </livre> La racine du document. <livre> <titre> <prix> Les éléments.

lang= "fr" ou 'Ar' Attribut pour l'élément <livre> <!-- pour un commentaire -->

II.2 Structure XML de base

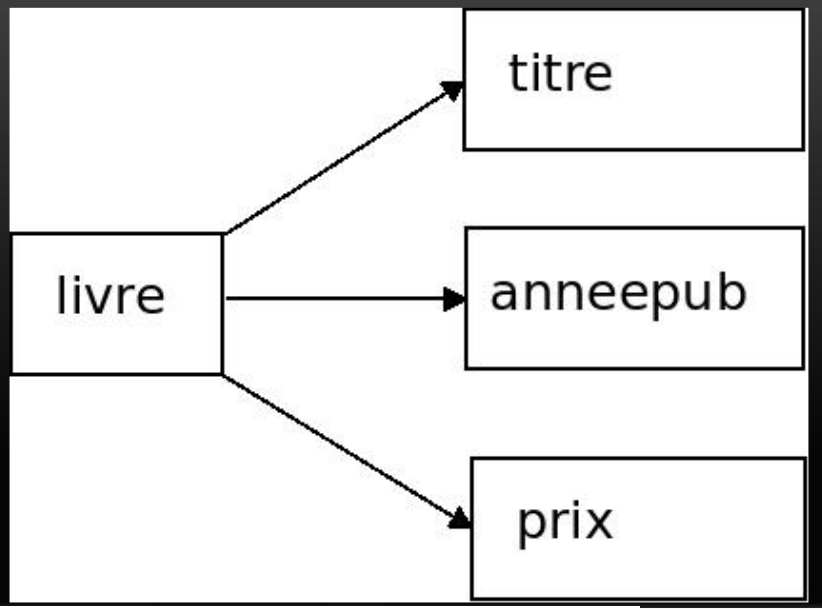
Les éléments:

<titre>Genie Logiciel</titre>

<anneepub>2023</anneepub>

<prix>2500</prix>

Trois éléments fils de l'élément <livre>



Entity References	Character
&lt;	<
&gt;	>
&amp;	&
&quot;	"
&apos;	'

II.2 Structure XML de base

Attributs réservés

xml:lang : Sa valeur indique le langage de l'élément.

<livre xml:lang="en-US">

xml:space = "default | preserve" : Sa valeur indique si un espace blanc à l'intérieur d'un élément est significatif et ne doit pas être altéré par le processeur XML.

Avec default le processeur XML est libre de faire ce qu'il veut avec les espaces. Sinon il faut utiliser preserve.

II.2 Structure XML de base

Comment choisir entre éléments et attributs en XML ?

Ajouter les informations (ISBN, DATE_EDITION) au code XML suivant (avec justification):

```
<?xml version="1.0"?>
<BIBLIOTHEQUE>
    <LIVRE lang="arabe">
            <TITRE>titre livre 1</TITRE>
            <AUTEUR>auteur 1</AUTEUR>
            <EDITEUR>editeur 1</EDITEUR>
    </LIVRE>
    <LIVRE lang="arabe">
            <TITRE>titre livre 2</TITRE>
            <AUTEUR>auteur 2</AUTEUR>
            <EDITEUR>editeur 2</EDITEUR>
    </LIVRE>
</BIBLIOTHEQUE>
```

II.2 Structure XML de base

Comment choisir entre éléments et attributs en XML ?

. Les attribut ne peuvent pas contenir des **informations multiples**.

- Les attributs ne possèdent pas **une structure arborescente**.
- Utiliser les éléments pour les données qui peuvent avoir des **informations multiples** ou une **structure arborescente**

```
<?xml version="1.0"?>
<BIBLIOTHEQUE>
    <LIVRE lang="arabe" ISBN="2-5678-6987-6">
            <DATE_EDITION>12-06-2010</DATE_EDITION>
            <TITRE>titre livre 1</TITRE>
            <AUTEUR>auteur 1</AUTEUR>
            <EDITEUR>editeur 1</EDITEUR>
    </LIVRE>
    <LIVRE lang='arabe' ISBN='6-9876-2354-9'>
            <DATE_EDITION>09-08-2010</DATE_EDITION>
            <TITRE>titre livre 2</TITRE>
            <AUTEUR>auteur 2</AUTEUR>
            <EDITEUR>editeur 2</EDITEUR>
    </LIVRE>
</BIBLIOTHEQUE>
```

II.2 Structure XML de base

Règles à respecter en XML:

- Toute balise ouvrante doit avoir une balise fermante.
- Un document XML est constitué par des éléments et des attributs.
- Chaque document comporte un seul élément racine (root element).
- L'élément racine comporte des sous-éléments (children elements)
- Un élément comprend du simple texte et/ou d'autres éléments
- Chaque élément peut avoir 0 ou plusieurs attributs
- Un nom d'élément ne peux commencer par un chiffre ou un caractère de ponctuation.
- Les valeurs d'attributs doivent être entre guillemets " " ' '.
- XML est sensible à la casse.

II.2 Structure XML de base

Exemple :

recette.txt

Fichier Édition Rechercher Options Aide

```
French Meringues

Ingredients
        4 egg whites
        2,25 cups confectioners' sugar
Directions

Step 1
Preheat the oven to 200 degrees F (95 degrees C). Butter and
flour a baking sheet.

Step 2
In a glass or metal bowl, whip egg whites until foamy using an
electric mixer. Sprinkle in sugar a little at a time, while
continuing to whip at medium speed.

Step 3
Place the meringues in the oven and place a wooden spoon
handle in the door to keep it from closing all the way. Bake
for 3 hours, or until the meringues are dry, and can easily be
removed from the pan.
```

II.2 Structure XML de base

Exemple:

```
-<recette id="1336">
   <titre>French Meringues</titre>
  -<ingredients>
     <ingredient qte="4">egg whites</ingredient>
     <ingredient qte="2.25" unite="cup">confectioners' sugar</ingredient>
   </ingredients>
  -<directions>
    -<step n="1">
       Preheat the oven to 200 degrees F (95 degrees C). Butter and flour a baking sheet.
     </step>
    -<step n="2">
       In a glass or metal bowl, whip egg whites until foamy using an electric mixer. Sprinkle in sugar a little
       at a time, while continuing to whip at medium speed.
     </step>
    -<step n="3">
       Place the meringues in the oven and place a wooden spoon handle in the door to keep it from closing
       all the way. Bake for 3 hours, or until the meringues are dry, and can easily be removed from the pan.
     </step>
   </directions>
 </recette>
```

II.2 Structure XML de base

- Visualisation de code XML:
  - Plusieurs outils pour la visualisation sous forme d'arbre,
- Exemples :
  - https://codebeautify.org/xmlviewer
  - Oxygen XML editor
  - …

II.2 Structure XML de base

Best Online XML Viewer, XML For

https://codebeautify.org/xmlviewer

Code Beautify

JSON Formatter | Hex Color Codes | HMAC Generator | My Ip | Search | Recent Links | More | Si

XML Viewer☆

Save & Sh

XML Input sample

```
1  <?xml version="1.0" encoding="ISO-8859-1" ?>
2  <livre lang="fr">
3    <titre>Genie Logiciel</titre>
4    <anneepub>2023</anneepub>
5    <auteur> Benali khaled</auteur>
6    <prix>2500</prix>
7  </livre>
8
```

Load Url

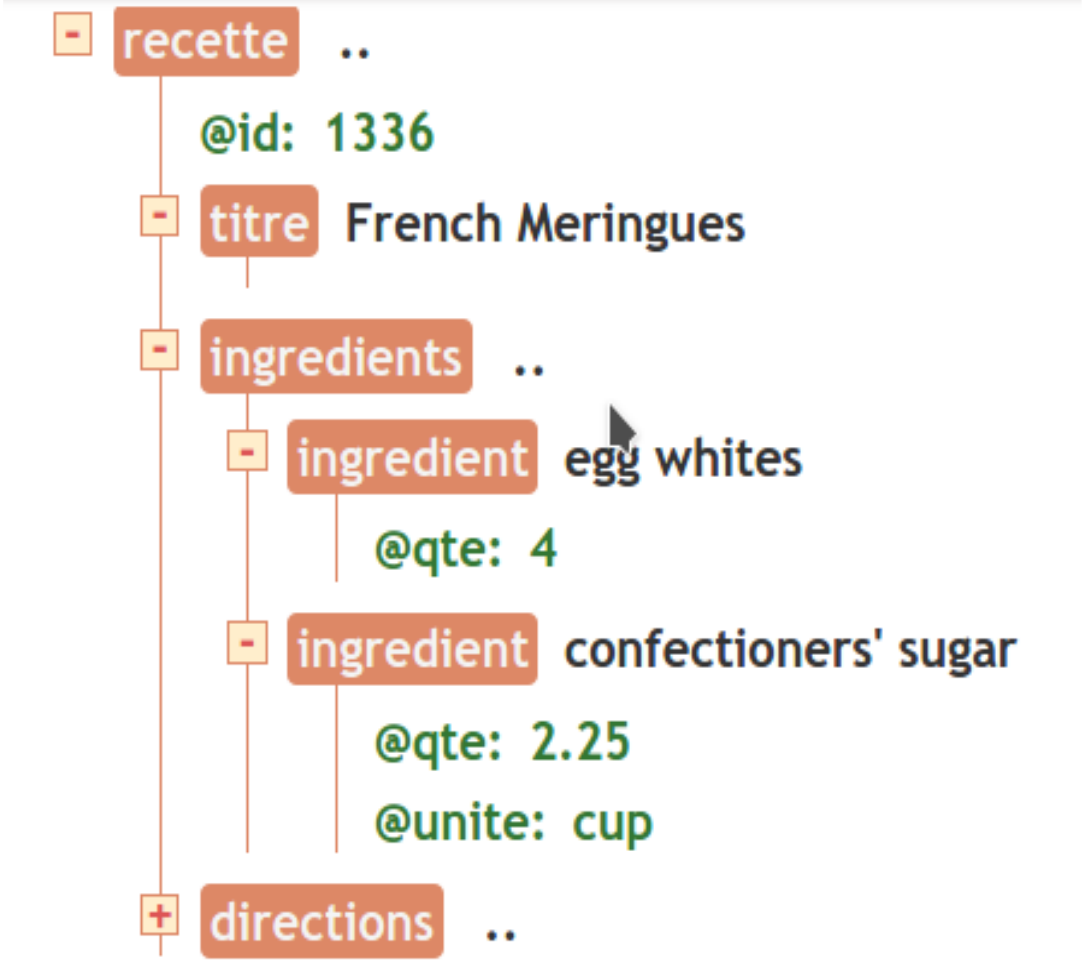
Browse

Tree View

Result : XML Tree View

- livre ..
  - @lang: fr
  - titre: Genie Logiciel
  - anneepub: 2023
  - auteur: Benali khaled
  - prix: 2500

II.2 Structure XML de base



II.3 Domaines nominaux

II.3 Domaines nominaux

Liberté en définition des balises : Risques des conflits au moment de fusion,

- Exemple :

- Livre
- Client
- Commande (?)

II.3 Domaines nominaux

```
<?xml version="1.0" ?>
<commande>
  <client>
          <code>1200</code>
           <nom>Benahmed</nom>
           <adresse>mila</adresse>
  </client>
  <livres><livre>
              <titre>GL</titre>
             <auteurs>  <auteur>
                           <nom>Benali</nom>
                          <nationnalite>algérienne</nationnalite>
                       </auteur>
            </auteurs>
            <prix>2500</prix>
   </livre>  </livres>
</commande>
```

II.3 Domaines nominaux

Namespaces

- Définition de collections de nom,
- Applicables sur les balises et les attributs
- Chaque collection est définie par un URI,
- L'URI est associé à un préfixe
  - xmlns:préfixe="URI"

II.3 Domaines nominaux

```
<?xml version="1.0" ?>
<commande>
  xmlns="http://www.univ-mila.dz/commande/"
  xmlns:l="http://www.univ-mila.dz/livre/"
  xmlns:c="http://www.univ-mila.dz/client/">
  <c:client>
     <c:code>1200</c:code>
          <c:nom>Benahmed</c:nom>
          <c:adresse>mila</c:adresse> </
  c:client>
  <livres><l:livre>
    <l:titre>GL</l:titre>
    <l:auteurs> <l:auteur>
                  <l:nom>Benali</l:nom>
                  <l:nationnalite>algérienne</l:nationnalite>
                </l:auteur>
    </l:auteurs>
    <l:prix>2500</l:prix>
  </l:livre> </livres>
</commande>
```

II.4 DTD

II.4 DTD

II.4.1 Définition d'une DTD (Document Type Definition, Déclaration de type de doc)

Les DTD sont des documents permettant de valider la conformité d'un document XML. La validation est un moyen pour vérifier que votre doc est conforme à une grammaire (certains contraintes à respecter).

DTD précise la structure d'un fichier xml (est ce qu'on utilise un attribut ou un élément, le type d'élément utilisé, l'ordre des éléments, la présence de texte brut, les attributs autorisés ou obligatoires pour chaque élément…)

II.4 DTD

II.4.1 Définition d'une DTD (Document Type Definition, Déclaration de type de doc)

Fichier xml valide = fichier bien formé + respecte les contraintes de DTD

II.4.2 Utilité de DTD

Si je vous donne un paragraphe à 10 personnes, il est possible d'avoir 10 fichiers xml différents mais si on utilise un fichier DTD qui précise la structure de fichier xml on n'aura pas ce problème (on limite le nbr des fichiers xml mais pas forcément une seule solution à l'inverse au schéma xml).

II.4 DTD

II.4.2 Utilité d'une DTD

- Bien que facultative, il est souvent très intéressant de posséder une DTD, simplement pour **vérifier la validité du document XML**. Il est ainsi recommandé d'en utiliser une dans le cas du développement **parallèle** de plusieurs fichiers XML destinés à subir un traitement particulier (développement par plusieurs personnes par exemple).
- Avec une DTD, chacun de vos fichiers XML peut contenir une description de son **propre format**.
- Avec une DTD, **des groupes de personnes indépendants** peuvent convenir d'utiliser une DTD standard pour échanger des données.

II.4 DTD

II.4.3 Types de DTD

Une DTD peut être **interne** (à l'intérieur d'un fichier xml) ou **externe** (un fichier à part) où on peut le partager par plusieurs fichiers xml.

2 types de DTD externes : **public et system**

public: le mot clé PUBLIC indique une ressource disponible pour tous sur **un serveur distant**.

system: le mot clé SYSTEM indique que le fichier spécifié se trouve sur **l'ordinateur local** et il est disponible uniquement à titre privé.

II.4 DTD

Exemple d'une DTD externe

```
<!DOCTYPE biblio SYSTEM "bibliographie.dtd">
```

```
<!ELEMENT biblio (livre) * >
<!ELEMENT livre (titre, auteur, nb_pages) * >
<!ATTLIST livre type (roman | nouvelles |  poemes  |  theatre ) #IMPLIED
land CDATA "fr">
<!ELEMENT titre (#PCDATA)>
<!ELEMENT auteur (#PCDATA)>
<!ELEMENT nb_pages (#PCDATA)>
```

```
<!DOCTYPE biblio PUBLIC "http://www.w3c.org/TR/xhtml1/DTD/bibliographie.dtd">
```

- Exemple d'une DTD interne

```
<? Xml version = "1.0" encoding ="utf-8"?>
<!DOCTYPE biblio [
                        <!ELEMENT biblio (livre) * >
                        <!ELEMENT livre (titre, auteur, nb_pages) * >
                        <!ATTLIST livre type (roman | nouvelles | poemes | theatre ) #IMPLIED
                         land CDATA "fr">
                        <!ELEMENT titre (#PCDATA)>
                        <!ELEMENT auteur (#PCDATA)>
                        <!ELEMENT nb_pages (#PCDATA)>
                        ] >
<biblio>
    <livre>
        <titre> UML </titre> <auteur> Benali khaled </auteur>
         <nb_pages> 150 </nb_pages>
    </livre>
</biblio>
```

II.4 DTD

Dans une DTD on peut déclarer: élément, attribut et entité

II.4.4- Déclaration des éléments

<!ELEMENT nom-élément Type-élément>

Type-élément: Texte, Vide (0 ou plusieurs attributs), Contient d'autres éléments

1- Elément Texte: <!ELEMENT nom-élément (#PCDATA)>

Exp: <!ELEMENT titre (#PCDATA)>

2- Elément Vide: <!ELEMENT nom-élément EMPTY>

Exp: <!ELEMENT Formation EMPTY)> en DTD

<Formation type="complet"/> en XML

II.4 DTD

II.4.4- Déclaration des éléments

3- Elément qui contient d'autres éléments:

a- **Séquence d'éléments:** une liste ordonnée d'éléments enfants de l'élément défini. On utilise une virgule (et séquentiel) pour séparer et ordonner les éléments.

<!ELEMENT elt0 (elt1, elt2, elt3)> elt1, elt2 et elt3 des éléments enfants de elt0 et elles sont ordonnées.

<!ELEMENT getiteasy (Formation)>

<!ELEMENT Formation (titre, formateur, ajout) >

II.4 DTD

- Exemple d'une séquence d'éléments

```
<? xml version = "1.0" encoding ="utf-8"?>
<!DOCTYPE getiteasy [
<!ELEMENT getiteasy (formation)>
<!ELEMENT formation (titre, formateur, ajout)  >
<!ELEMENT titre (#PCDATA)>
<!ELEMENT formateur (#PCDATA)>
<!ELEMENT ajout EMPTY>
]>
<getiteasy>
    <formation>
        <titre> Formation XML </titre>
         <formateur> Benali khaled </formateur>
        < ajout  />
   </formation>
</getiteasy>
```

II.4 DTD

II.4.4- Déclaration des éléments

b – Choix d'éléments: le choix (ou) dans une liste de plusieurs éléments possibles. De même que pour la séquence, les éléments-enfants doivent être déclarés dans la DTD. Cette liste est composée d'éléments séparés par le caractère | (alt + 6).

| = ou (or)

<!ELEMENT getiteasy (formation)>

<!ELEMENT Formation (titre | formateur | ajout) > <!-- je peux faire l'élément titre ou formateur ou ajout comme élément enfant dans formation -->

II.4 DTD

- Exemple de choix d'éléments

```
<? xml version = "1.0" encoding ="utf-8"?>
<!DOCTYPE getiteasy [
<!ELEMENT getiteasy (formation)>
<!ELEMENT formation (titre | formateur | ajout)  >
<!ELEMENT titre (#PCDATA)>
<!ELEMENT formateur (#PCDATA)>
<!ELEMENT ajout EMPTY>
]>
<getiteasy>
    <formation>
        <titre> Formation XML </titre>
    </formation>
</getiteasy>
```

II.4 DTD

II.4.4- Déclaration des éléments

4 - Indicateurs d'occurrences: lors de la déclaration de séquence ou de choix d'éléments, à chaque élément enfant peut être attribué une indication d'occurrence (?,+ ou *).

Ces opérateurs sont liés aux concept de cardinalité lorsqu'il y a pas d'opérateur, par défaut le min = 1 le max = 1.

- * : 0 , n le nombre d'occurrence d'un élément dans le fichier (min 0 max n)
- + : 1 , n le nbr d'occurrence d'un élément dans le fichier (min 1 max n)
- ? : 0 , 1 le nbr d'occurrence d'un élément dans le fichier (min 0 max 1)

<!ELEMENT elt0 (element1, element2?, element3+, element4*) >

element2 peut apparaitre 0 ou une seule fois dans elt0

II.4 DTD

- Exemple d'indicateurs d'occurrences

```
<? xml version = "1.0" encoding ="utf-8"?>
<!DOCTYPE getiteasy [
<!ELEMENT getiteasy (formation)>
<!ELEMENT formation (titre*, formateur+, ajout?)  >
<!ELEMENT titre (#PCDATA)>
<!ELEMENT formateur (#PCDATA)>
<!ELEMENT ajout EMPTY>
]>
<getiteasy>
    <formation>
        <titre> Formation XML </titre>
        <titre> Formation XML </titre>  <!--  0,n-->
          <formateur> </formateur>  <!--  1,n-->
          <ajout/>   <!--  0,1-->
</formation>
</getiteasy>
```

II.4 DTD

II.4.4- Déclaration des éléments

5 - Elément quelconque (Mixte): Il peut contenir tout autre élément défini dans la DTD (vide ou texte). Les éléments-enfants peuvent apparaitre dans n'importe quel ordre et en quantité non définie.

- Il est préférable de ne pas utiliser trop souvent ce type de déclaration, car on perd les avantages de DTD (fixer des contraintes précises sur la structure de XML).

On utilise le mot clef ANY.

<!ELEMENT nom_element ANY>

II.4 DTD

- Exemple d'un élément quelconque ANY

```
<? xml version = "1.0" encoding ="utf-8"?>
<!DOCTYPE getiteasy [
<!ELEMENT getiteasy (doc)>
<!ELEMENT doc ANY  >
<!ELEMENT i (#PCDATA)>
]>

<!– tous les éléments sont autorisés à être dans l'élément doc soit balise soit texte-->

<getiteasy>
    <doc>  ceci est une  <i> documentation <i/>   </doc>
</getiteasy>
```

II.4 DTD

II.4.4- Déclaration des éléments

6 - Elément à contenu mixte: peut contenir aussi bien du texte que des éléments-enfants. Il se présente comme une liste de choix |, avec des indicateurs d'occurrence bien choisis. Le texte contenu peut se trouver à n'importe quel endroit dans l'élément, et peut être une section CDATA.

Exemple de déclaration

<!ELEMENT auteurs (PCDATA | auteur)*>

II.4 DTD

- Exemple d'un élément de contenu mixte

```
<? xml version = "1.0" encoding ="utf-8"?>
<!DOCTYPE getiteasy [
<!ELEMENT getiteasy (auteurs)*>
<!ELEMENT auteurs (#PCDATA | auteur)*  >
<!ELEMENT auteur (#PCDATA)>
]>

<getiteasy>
<!–1 er utilisation-->
     <auteurs> Getiteasy <auteurs/>
<!–2em utilisation-->
 <auteurs>
  <auteur> benali khaled  <auteur/>
  <auteur> Kortich AMin <auteur/>
<auteurs/>
</getiteasy>
```

II.4 DTD

II.4.5- Déclaration des attributs

On peut faire des contraintes sur les attributs. On a 2 types d 'attributs

1- Attribut dans un élément de texte <domaine type="SQL"> BDD </domaine>

2- Attribut dans un élément vide

Chaque attribut à un nom, un type, un type d'obligation et une valeur par défaut.

Le mot clef utilisé est ATTLIST.

II.4 DTD

II.4.5- Déclaration des attributs

Syntaxe:

<!ATTLIST nom_elt nom_att TYPE OBLIGATION "VALEUR_PAR_DEFAUT">

Chaque attribut peut être requis, optionnel ou fixe (obligation) et avoir une valeur par défaut.

Exp: <!ATTLIST formation lang CDATA #FIXED "ar">

#IMPLIED: attribut optionnel #REQUIRED: attribut obligatoire

#FIXED: attribut toujours présent avec une valeur (constante)

"ar" valeur par défaut de la langue même si je n'ai pas initialisé l'attribut.

II.4 DTD

II.4.5- Déclaration des attributs

TYPE:

CDATA: texte (chaîne de caractères)

ID: identifiant unique (combinaison chiffre + lettres)

NMTOKEN: un mot (donc pas de blanc)

NMTOKENS: une liste de mots (séparation par un blanc)

Une énumération de valeurs (un type avec plusieurs valeurs): chaque valeur est séparée par le caractère | auteur (Benali | Kortich)

IDREF: reference un identifiant d'un autre element

II.4 DTD


```
<? xml version = "1.0" encoding ="utf-8"?>
<!DOCTYPE getiteasy [
<!ELEMENT getiteasy (formation)+>
<!ELEMENT formation (#PCDATA)>
<!ATTLIST formation lang CDATA #IMPLIED>
]>
<getiteasy>
    <formation> JAVA </formation>
    <formation lang=  "ar"> LANGAGE C </formation>
</getiteasy>
```

```
<? xml version = "1.0" encoding ="utf-8"?>
<!DOCTYPE getiteasy [
<!ELEMENT getiteasy (formation)+>
<!ELEMENT formation (#PCDATA)>
<!ATTLIST formation lang CDATA #FIXED "ar">
]>
<!—valeur langue ne peut être changé -->
<getiteasy>
    <formation"> JAVA </formation>
     <formation lang=  " ar"> LANGAGE C </formation>
</getiteasy>
```

II.4 DTD


```
<?xml version = "1.0" encoding ="utf-8"?>
<!DOCTYPE getiteasy [
<!ELEMENT getiteasy (formation)+ >
<!ELEMENT formation (#PCDATA)>
<!ATTLIST formation etat (ar) #IMPLIED>
<!ATTLIST formation auteur (Benali | kortich) #IMPLIED>
<!ATTLIST formation site CDATA #FIXED  "www.univ-
mila.dz"> ]>
<getiteasy>
    <formation etat="ar">  programmer JAVA </formation>
    <formation auteur="Benali">  programmer C </formation>
    <formation auteur= "kortich">  programmer C </formation>
    <formation site=  " www.univ-mila.dz"> LANGAGE C++ </formation>
    <formation> Langage XML </formation>
</getiteasy>
```

II.4 DTD


```
<!—exemple Type ID   code unique ->
<? xml version = "1.0" encoding ="utf-8"?>
<!DOCTYPE getiteasy [
<!ELEMENT getiteasy (formation) +>
<!ATTLIST formation code ID #IMPLIED>
]>
<getiteasy>
    <formation code="a1">  programmer JAVA </formation>
    <formation code="a2">  programmer C </formation>
    <formation code="a3">  UML </formation>
</getiteasy>
```

II.4.5- Déclaration des attributs

On peut utiliser l'instruction suivante pour gagner du temps:

```
<!ATTLIST Element
        attr1 CDATA #IMPLIED
        attr2 CDATA #REQUIRED
        ….
>
<!ATTLIST Formation
        lang (ar) #IMPLIED
        site CDATA #FIXED  "univ-mila.dz" ….


>
```

II.4 DTD

II.4.6- Déclaration des entités

Permet de disposer de l'équivalent de raccourcis clavier qui nous permet d'écrire des caractères non autorisés (par exemple les lettres arabes) avec le format d'encodage utilisé.

On utilise l'instruction ENTITY.

L'entité associe un nom à une valeur. Ce nom est employé dans le doc xml comme une forme de raccourci vers la valeur. Syntaxe: &nom;

Syntaxe: <!ENTITY nom "VALEUR"> en DTD &nom; appel en xml

Exemple: <!ENTITY aacc " "> en DTD &aacc; en xml

  valeur de Unicode, ce code remplace a accent (à)

II.4 DTD

II.4.6- Déclaration des entités

Les entités paramétriques sont des raccourcis (symboles) pour remplacer des valeurs ou des types prédéfinis en DTD.

<!ENTITY % nom "valeurs">

Exemples <!ENTITY % HEADING "H1 | H2 | H3 | H4 | H5 | H6">

<!ENTITY % Number "CDATA">

en DTD

<!ATTLIST Etudiant Numero %Number; #IMPLIED>

II.4 DTD

II.4.6- Déclaration des entités (internes ou externes)

Les entités internes sont des symboles définis dans une DTD interne et utilisés dans un doc xml comme raccourcis d'écriture.

```
<!DOCTYPE getiteasy [
<!ENTITY ADN "Acide désoxyribonucléique">
]>
<info>
   L' &ADN; est une molécule complexe
</info>
```

Les entités externes sont des symboles définis dans un autre fichier (xml ou dtd).

```
<!ENTITY Inclusion SYSTEM "regle.xml">
```

II.4 DTD

Résumé- DTD

- Déclaration d ' éléments
  - <!ELEMENT nom_element catégorie>
  - <!ELEMENT nom_element (sous_elements)>
- Opérateurs :
  - + : 1 ou plusieurs,
  - * : 0 ou plusieurs,
  - ? : 0 ou 1,
  - , : liste d'éléments,
  - | : OU,
  - #PCDATA
  - EMPTY (vide)
  - ANY

II.4 DTD

Résumé-DTD

- Déclaration d'Attributs

  <!ATTLIST nom_element nom_attribut type_attribut valeur_attribut>

- Types Attributs
  - CDATA : chaîne de carcatères,
  - (val1|val2|val3) : une valeur de la liste,
  - ID : valeur unique,
  - ...
- Valeurs Attributs & obligation
  - "valeur" : valeur par défaut,
  - #REQUIRED : obligatoire,
  - #IMPLIED : optionnel,
  - #FIXED « valeur » : valeur fixe.

II.5 XML Schéma (XSD)

II.4 XML Schéma (XSD)

XML - Schéma est une alternative XML aux DTD. En réalité XML Schéma apparaît comme le successeur des DTD car il est par nature extensible et s'appuie sur XML.

Ainsi XML Schéma décrit (en XML) la structure d'un document XML c'est-à-dire: les éléments qui composent un document, les attributs, la hiérarchie entre les éléments, l'ordre des sous éléments, le nombre de sous éléments, les types des éléments et attributs, les valeurs par défaut, le format ou la restriction des valeurs d'un élément ou d'un attribut.

On parle ainsi de XML Schéma Définition (XSD).

II.4 XML Schéma (XSD)

Manques de DTD:

1- DTD hérite sa syntaxe de langage SGML (un peu difficile)

2- DTD ne supporte pas les espaces de nom.

3- DTD ne précise pas les types de données (DTD utilise des types en général)

Avantages de XML schéma:

1- Typage des données (entier, réel, booléen,…)

2- Supporte les espaces de nom

3- Les indicateurs d'occurrences peuvent être tout nombre non négatif

4- Facile à comprendre

5-…

II.4 XML Schéma (XSD)

Structure de base :

Comme tout document XML, un Schema XML commence par un prologue, et a un élément racine.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema
      xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
      <!-- déclarations d'éléments, d'attributs et de types ici -->
</xsd:schema>
```

- L'élément racine est l'élément xsd:schema.
- l'attribut xmlns:xsd qui fait référence à l'espace de noms utilisé pour l'écriture du fichier.
- Il faut simplement retenir que tout élément d'un schéma doit commencer par le préfixe xsd (on peut changer ce préfixe).

II.4 XML Schéma (XSD)

Déclaration d'éléments

Un élément, dans un schéma, se déclare avec la balise <xsd:element>. Par exemple:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
    <xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
      <xsd:element name="contacts" type="typeContacts" />
      <xsd:element name="remarque" type="xsd:string" />
    <!-- déclarations de types ici -->
    </xsd:schema>
```

- Le schéma précédent déclare deux éléments : un élément contacts et un élément remarque.
- Chaque élément est « typé » -c'est-à-dire qu'il doit respecter un certain format de données.

II.4 XML Schéma (XSD)

Déclaration d'éléments

- L'élément contacts est ainsi du type typeContacts, qui est un type complexe défini par l'utilisateur.
- L'élément remarque quant à lui est du type xsd:string qui est un type simple prédéfini de XML Schema.
- Chaque élément déclaré est associé à un type de données via l'attribut type.
- Les éléments pouvant contenir des élément-enfants ou posséder des attributs sont dits de type-complexe, tandis que les éléments n'en contenant pas sont dits de type-simple.

II.4 XML Schéma (XSD)

Déclaration d'attributs

- À la différence des éléments, un attribut ne peut être que de type simple.
- Cela signifie que les attributs, comme avec les DTD, ne peuvent contenir d'autres éléments ou attributs.
- De plus, les déclarations d'attributs doivent être placées après les définitions des types complexes, autrement dit, après les éléments xsd:sequence, xsd:choice et xsd:all.
- Pour mémoire, rappelons que dans une DTD, l'ordre des déclarations n'a pas d'importance au contraire de XSD.

II.4 XML Schéma (XSD)

Déclaration d'attributs

```
<?xml version="1.0" encoding="ISO-8859-1"?>
    <xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
    <xsd:element name="contacts" type="typeContacts" />

    <xsd:element name="remarque" type="xsd:string">
        <!-- déclarations de types ici -->
        <xsd:complexType>
        <!-- déclarations du modèle de contenu ici -->
          <xsd:attribute name="maj" type="xsd:date" />
        </xsd:complexType>
    </xsd:element>
    </xsd:schema>
```

II.4 XML Schéma (XSD)

Contraintes d'occurrences

- Tout comme dans une DTD, un attribut peut avoir un indicateur d'occurrences.
- L'élément attribute d'un Schema XML peut avoir trois attributs optionnels : use, default et fixed.
- Des combinaisons de ces trois attributs permettent de paramétrer ce qui est acceptable ou non dans le fichier XML final (attribut obligatoire, optionnel, possédant une valeur par défaut...)

II.4 XML Schéma (XSD)

Contraintes d'occurrences

Par exemple, la ligne suivante permet de rendre l'attribut maj **optionnel**, avec une **valeur par défaut** au 11 octobre 2023 s'il n'apparaît pas.

Note: le format de date est standardisé : cette date s'écrit donc à l'anglo-saxonne année-mois-jour.

```
<xsd:attribute name="maj" type="xsd:date" use="optional" default="2023-10-11" />
```

- Quand l'attribut **fixed** est renseigné, la seule valeur que peut prendre l'attribut déclaré est celle de l'attribut fixed.

II.4 XML Schéma (XSD)

Obligation et valeurs par défaut (comparaison entre DTD et XSD)

DTD	Attribut **use**	Attribut **default**	Commentaire
#REQUIRED	required	-	
"blabla" #REQUIRED	required	blabla	
#IMPLIED	optional	-	
"blabla" #IMPLIED	optional	blabla	
-	prohibited	-	Cet attribut ne doit pas apparaître

II.4 XML Schéma (XSD)

Déclaration d'élément ne contenant que du texte avec un (ou plusieurs) attribut(s)

Un tel élément est de type complexe, car il contient au moins un attribut. Afin de spécifier qu'il peut contenir également du texte, on utilise l'attribut mixed de l'élément xsd;complexType. Par défaut, mixed="false"; il faut dans ce cas forcer mixed="true". Par exemple:

```
<xsd:element name="elt">
    <xsd:complexType mixed="true">
      <xsd:attribute name="attr" type="xsd:string" use="optional" />
    </xsd:complexType>
</xsd:element>
```

II.4 XML Schéma (XSD)

Déclaration et référencement

Pour des raisons de clarté, il est beaucoup plus avantageux d'ordonner ces déclarations.

Il est recommandé de commencer par déclarer les éléments et les attributs de type simple, puis ceux de type complexe. On peut en effet faire référence, dans une déclaration de type complexe, à un élément de type simple préalablement défini.

Déclaration et référencement

```
<xsd:element name="livre">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="auteur" type="xsd:string" />
        <xsd:element name="pages" type="xsd:positiveInteger" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

Plus difficile à maintenir que:

```
<xsd:element name="pages" type="xsd:positiveInteger" />
  <xsd:element name="auteur" type="xsd:string" />
  <xsd:element name="livre">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="auteur" />
        <xsd:element ref="pages" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

II.4 XML Schéma (XSD)

Les types de données

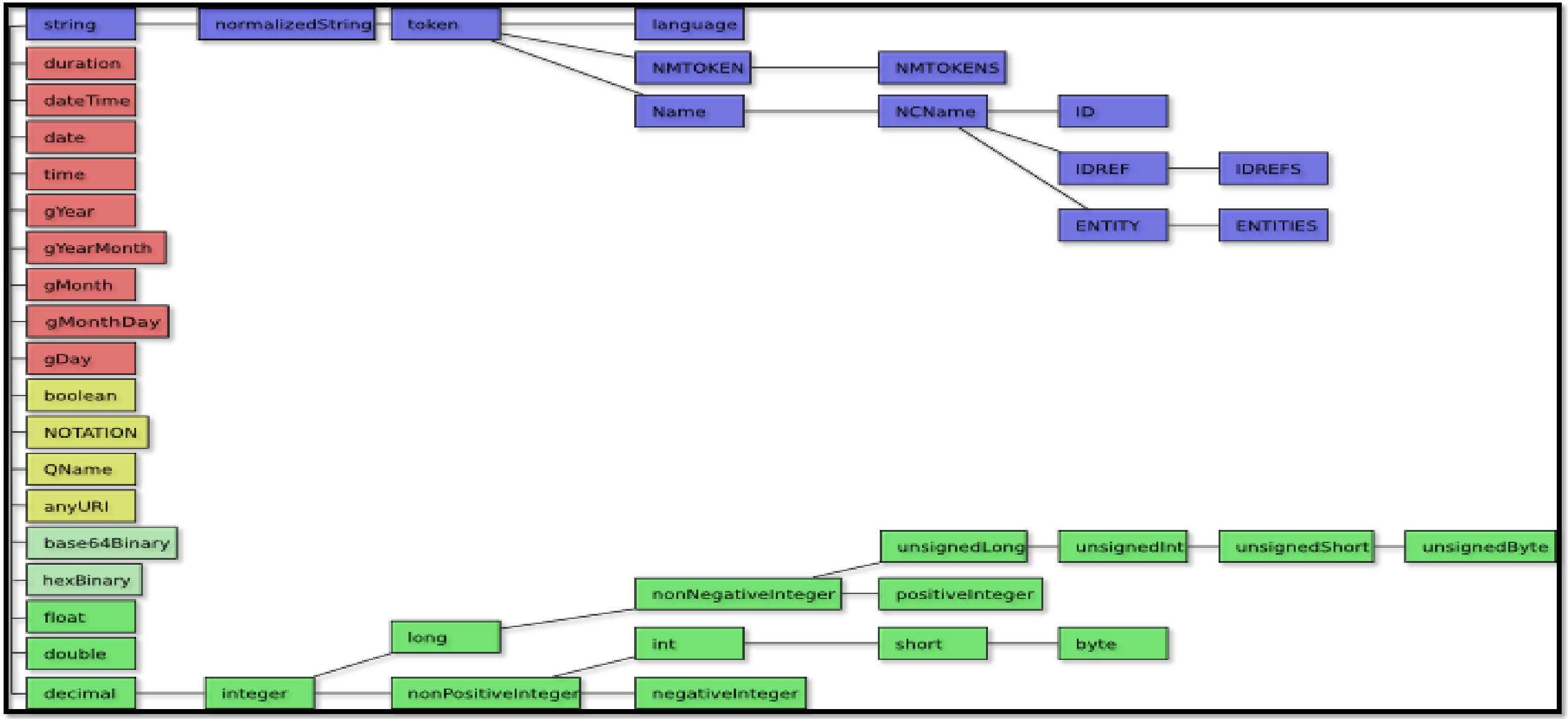
Il distingue notamment types simples et types complexes.

Les types simples (Bibliothèque de types intégrés)

ne peuvent comporter ni attributs, ni éléments enfants. Il en existe de nombreux, prédéfinis. Les types de données les plus simples (les chaînes de caractères) que permettaient les DTDs sont conservés, mais d'autres ont fait leur apparition. On pourra envisager, par exemple, dans un schéma décrivant un bon de commande, la déclaration d'un attribut quantite qui force la valeur de l'attribut à un être un entier positif. Un bon de commande XML suivant ce schéma, ayant une commande spécifiant quantite="-3" sera alors automatiquement refusé par le système.

```
<xsd:attribute name="quantite" type="xsd:positiveInteger" use="default" value="1" />
```

II.4 XML Schéma (XSD)



II.4 XML Schéma (XSD)

Les types de données

Les types simples : Liste

sont des suites de types simples (ou atomiques). Il est également possible de créer une liste personnalisée, par « dérivation » de types existants. Par exemple:

```
<xsd:simpleType name="numéroDeTéléphone">
    <xsd:list itemType="xsd:unsignedByte" />
</xsd:simpleType>
```

Un élément conforme à cette déclaration serait

<téléphone>01 44 27 60 11</téléphone>.

II.4 XML Schéma (XSD)

Les types de données

Les types simples : Union

Les listes et les types simples intégrés ne permettent pas de choisir le type de contenu d'un élément. On peut désirer, par exemple, qu'un type autorise soit un nombre, soit une chaîne de caractères particuliers. Il est possible de le faire à l'aide d'une déclaration d'union. Par exemple, sous réserve que le type simple numéroDeTéléphone ait été préalablement défini (voir précédemment), on peut déclarer:

```
<xsd:simpleType name="numéroDeTéléphoneMnémoTechnique">
    <xsd:union memberTypes="xsd:string numéroDeTéléphone" />
</xsd:simpleType>
```

Les éléments suivants sont alors des "instances" valides de cette déclaration :

<téléphone>18</téléphone>

<téléphone>Pompiers</téléphone>

II.4 XML Schéma (XSD)

Les types de données:

Les types complexes

- Un élément de type simple ne peut contenir pas de sous-élément. Il est nécessaire pour cela de le déclarer de type complexe.
- On peut alors déclarer des séquences d'éléments, des types de choix ou des contraintes d'occurrences.

Séquences d'éléments

On utilise pour ce faire l'élément xsd:sequence, qui reproduit l'opérateur , du langage DTD.

II.4 XML Schéma (XSD)

Les types de données:

Les types complexes Séquences d'éléments

```
<xsd:complexType>
    <xsd:sequence>
        <xsd:element name="nom" type="xsd:string" />
        <xsd:element name="prénom" type="xsd:string" />
        <xsd:element name="dateDeNaissance" type="xsd:date" />
        <xsd:element name="adresse" type="xsd:string" />
        <xsd:element name="adresseElectronique" type="xsd:string" />
        <xsd:element name="téléphone" type="numéroDeTéléphone" />
    </xsd:sequence>
</xsd:complexType>
```

xsd:sequence est équivalent à une déclaration d'élément, dans une DTD, où apparaîtrait (nom, prénom, dateDeNaissance, adresse, adresseElectronique, téléphone).

II.4 XML Schéma (XSD)

Les types de données: Les types complexes : Choix d'éléments

On utilise un élément xsd:choice : Ce connecteur a donc les mêmes effets que l'opérateur | dans une DTD.

```
<xsd:complexType name="typePersonne">
    <sequence>
      <xsd:element name="nom" type="xsd:string" />
      <xsd:element name="prénom" type="xsd:string" />
      <xsd:element name="dateDeNaissance" type="xsd:date" />
      <xsd:choice>
        <xsd:element name="adresse" type="xsd:string" />
        <xsd:element name="adresseElectronique" type="xsd:string" />
      </xsd:choice>
    </sequence>
    <xsd:element name="téléphone" type="numéroDeTéléphone" />
  </xsd:complexType>
```

II.4 XML Schéma (XSD)

Les types de données: Les types complexes : l'élément ALL

Cet élément est une nouveauté par rapport aux DTD. Il indique que les éléments enfants doivent apparaître une fois (ou pas du tout), mais dans n'importe quel ordre. Cet élément xsd:all doit être un enfant direct de l'élément xsd:complexType. Il indique que chacun de ces éléments peut apparaître une fois ou pas du tout (équivalent de l'opérateur ? dans une DTD), et que l'ordre des éléments n'a pas d'importance (cela n'a pas d'équivalent dans une DTD)

```
<xsd:complexType>
    <xsd:all>
      <xsd:element name="nom" type="xsd:string" />
      <xsd:element name="prénom" type="xsd:string" />
      <xsd:element name="dateDeNaissance" type="xsd:date" />
      <xsd:element name="adresse" type="xsd:string" />
      <xsd:element name="adresseElectronique" type="xsd:string" />
      <xsd:element name="téléphone" type="numéroDeTéléphone" />
    </xsd:all>
  </xsd:complexType>
```

II.4 XML Schéma (XSD)

Les types de données: Les types complexes : déclaration d'attributs

Si un tel élément contient des attributs, alors les déclarations de ces derniers doivent se faire juste avant la balise fermante </xsd:comlexType>

```
<xsd:complexType>
      <xsd:sequence>
        <xsd:element name="dateDeNaissance" type="xsd:date" />
        <xsd:element name="adresse" type="xsd:string" />
        <xsd:element name="adresseElectronique" type="xsd:string" />
        <xsd:element name="téléphone" type="numéroDeTéléphone" />
      </xsd:sequence>
      <xsd:attribute name="nom"/>
      <xsd:attribute name="prénom"/>
</xsd:complexType>
```

II.4 XML Schéma (XSD)

Indicateurs d'occurrences

Dans une DTD, un indicateur d'occurence ne peut prendre que les valeurs 0, 1 ou l'infini. On peut forcer un élément sselt à être présent 378 fois, mais il faut pour cela écrire (sselt, sselt..., sselt, sselt) 378 fois. XML Schema permet de déclarer directement une telle occurence, car tout nombre entier non négatif peut être utilisé. Pour déclarer qu'un élément peut être présent un nombre illimité de fois, on utilise la valeur unbounded (n). Les attributs utiles sont minOccurs et maxOccurs, qui indiquent respectivement les nombres minimal et maximal de fois où un élément peut apparaître. Le tableau suivant récapitule les possibilités :

Dans une DTD	Valeur de minOccurs	Valeur de maxOccurs
*	0	unbounded
+	1 (pas nécessaire, valeur par défaut)	unbounded
?	0	1 (pas nécessaire, valeur par défaut)
rien	1 (pas nécessaire, valeur par défaut)	1 (pas nécessaire, valeur par défaut)
impossible	nombre entier n quelconque	nombre entier m quelconque supérieur ou égal à n

II.4 XML Schéma (XSD)

Comment lier un fichier XML à un Schéma (un fichier xsd) ?

- On utilise le préfixe xmlns.
- Un schéma est en effet un document XML, et on trouve dans son élément racine l'attribut xmlns:xsd="http://www.w3.org/2001/XMLSchema". Cela signifie que dans le document, tous les éléments commençant par xsd sont référencés à cette URL. Donc si on a déposé un schéma à l'adresse http://www.monsite.org/collection_schemas/biblio, on peut l'appeler par <xsd:biblio xmlns="http://www.monsite.org/collection_schemas/biblio"> (DTD PUBLIC).

II.4 XML Schéma (XSD)

Comment lier un fichier XML à un Schéma (un fichier xsd) ?

- Dans le cas d'une référence locale (Dans DTD de type SYSTEM), on fait référence au schéma dans le document XML en utilisant l'attribut noNamespaceSchemaLocation, par:

```
<biblio xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="chemin de fichier xsd">.
```

Par exemple,

```
<biblio xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="biblio10.xsd">.
```

II.4 XML Schéma (XSD)

Restriction de type

- La dérivation par restriction permet de créer de nouveaux types simples à partir des types simples prédéfinis par le format XML Schema. On utilise des contraintes supplémentaires appliquées à un type simple particulier.
- On place une contrainte sur l'ensemble des valeurs que peut prendre un type de base. Par exemple, on peut souhaiter créer un type simple, appelé MonEntier, limité aux valeurs comprises entre 0 et 99 inclus. On dérive ce type à partir du type simple prédéfini nonNegativeInteger, en utilisant la contrainte maxExclusive.

II.4 XML Schéma (XSD)

Restriction de type (Type entier entre 1 et 100)

```
<xsd:simpleType name="monEntier">
        <xsd:restriction base="nonNegativeInteger">
          <xsd:maxExclusive value="100" />
        </xsd:restriction>
      </xsd:simpleType>
```

II.4 XML Schéma (XSD)

Restriction de type (Enumération)

```
<xsd:attribute name="jour" type="typeJourSemaine" use="required" /
        <xsd:simpleType name="typeJourSemaine">
          <xsd:restriction base="xsd:string">
            <xsd:enumeration value="lundi" />
            <xsd:enumeration value="mardi" />
            <xsd:enumeration value="mercredi" />
            <xsd:enumeration value="jeudi" />
            <xsd:enumeration value="vendredi" />
            <xsd:enumeration value="samedi" />
            <xsd:enumeration value="dimanche" />
          </xsd:restriction>
        </xsd:simpleType>
```

II.4 XML Schéma (XSD)

Limiter la longueur d'une chaîne

```
<xsd:simpleType name="typeMotLangueFrancaise">
        <xsd:restriction base="xsd:string">
          <xsd:length value="21" />
        </xsd:restriction>
      </xsd:simpleType>
```

II.4 XML Schéma (XSD)

Expressions régulières : e-mail

```
<xsd:simpleType name="typeAdresseElectronique">
        <xsd:restriction base="xsd:string">
          <xsd:pattern value="(.)+@(.)+" />
        </xsd:restriction>
      </xsd:simpleType>
```

Dans cet exemple, **(.)+** signifie que l'on peut mettre n'importe quel caractère au moins une fois, et qu'entre les deux chaînes doit impérativement apparaître le caractère **@**.

Expressions régulières : ISBN

```
<xsd:simpleType name="typeISBN">
          <xsd:restriction base="xsd:string">
            <xsd:pattern value="[0-9]{10}" />
          </xsd:restriction>
        </xsd:simpleType>
```