Module MLBDA
Master Informatique
Spécialité DAC

Cours 4 – Données du Web
Modèle semi-structuré et XML

Plan

- Données du Web
  - Caractéristiques
  - Données semi-structurées
- Modèles de données semi-structurées
  - OEM (Object Exchange Model)
  - XML (Extensible Markup Language)

Modèles de données du Web

Objectifs :

Prendre en compte :

- les données du Web, très hétérogènes
  - Documents HTML, SGML, Latex,
  - Données multimédia : son, graphique, images, vidéos, photos, dessins, etc.
- ses applications (e-commerce , B2B, bibliothèques, …)
  - besoin d'échanger, d'interroger et d'intégrer des données hétérogènes

Caractéristiques (1)

- Structure irrégulière :
  nécessité de modéliser et d'interroger ces structures

- Structure implicite :
  ex: un document électronique a un texte et une grammaire. On peut parser pour détecter des informations et des relations entre ces informations. Mais cela nécessite des outils spéciaux.

- Structure partielle :
  il manque des informations, certaines informations sont stockées hors de la base, et ne sont pas structurées.

- Structure indicative (et non pas contrainte, comme dans les BD) :
  pas de typage, pas de schéma (trop contraignants)

Caractéristiques (2)

- Un schéma a posteriori, et non a priori :
  en semi-structuré, le schéma est souvent postérieur aux données (c'est l'inverse en BD). Mais parfois, on peut suggérer ou guider (ex: pages personnelles).
- Un schéma très vaste (on ne sait pas tout),
  => il faut pouvoir l'interroger
- Un schéma ignoré (on parcourt tout à la recherche d'une chaîne)
  => pas possible en SQL, il faut trouver d'autres langages
- Un schéma qui évolue rapidement
  => à prendre en compte dans le langage de requêtes
- La distinction entre schéma et données est peu claire.
- Le typage est flou.

Quel modèle ?

Utiliser les modèles de données des SGBD pose des problèmes :

- La structure est trop rigide
- les données peuvent ne pas être conformes au schéma
  (==> valeurs nulles, difficultés de traitement et ambiguïtés)
- l'évolution fréquente de la structure de données conduit à des évolutions de schéma pas toujours maîtrisées.
- les données du Web sont indexées par des moteurs de recherche dont les services sont limités, l'interrogation souvent imprécise.

Nécessité d'un modèle général et souple, «sans schéma», avec un langage de requêtes associé : modèles semi-structurés

Modèle de données semi-structurées

Principe : partir des documents existants (HTML, XML, etc.) et trouver une structure commune, souple.

Les modèles semi-structurés utilisent des graphes annotés pour représenter les données.
Les différents modèles diffèrent par

- l'endroit où sont situées les annotations (arêtes et/ou nœuds)
- l'existence ou non d'un ordre sur les fils d'un nœud
- la façon de représenter le partage d'information

Ex : OEM : annotations sur arcs et feuilles, pas d'ordre
XML : annotations sur les nœuds et feuilles, existence d'un ordre
UnQL : annotations sur les arcs, pas d'ordre

OEM (Object Exchange Model)

Graphe dont les nœuds sont des objets, et dont les arcs sont étiquetés.

Toutes les entités sont représentées par des objets.

Chaque objet a un identificateur unique (oid), une étiquette le décrivant, un type et une valeur.

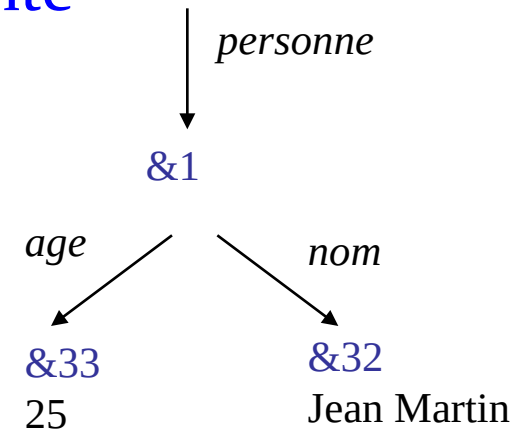
Certains objets sont atomiques (feuilles) et contiennent une valeur d'un type atomique quelconque (integer, real, string, html, audio, java, etc.)

Tous les autres objets sont complexes, leur valeur est une collection de références d'objets.

Les étiquettes sont des chaînes de caractères.

Représentation graphique

- Chaque objet ou valeur est un nœud
- Chaque *attribut (role)* est un arc orienté
- Les arcs sont étiquetés



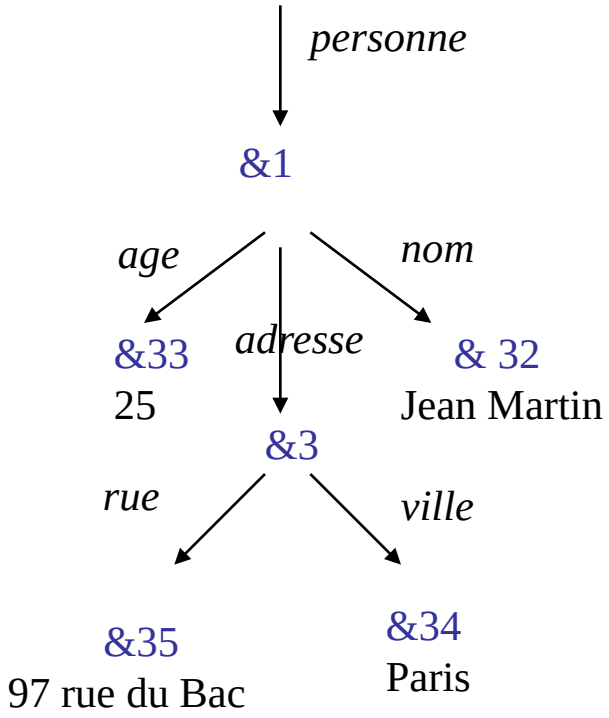

```
<personne &1
    nom &32 "Jean Martin "
    age &33 25
    />
```

Sous-objet

- Utilisation des identificateurs

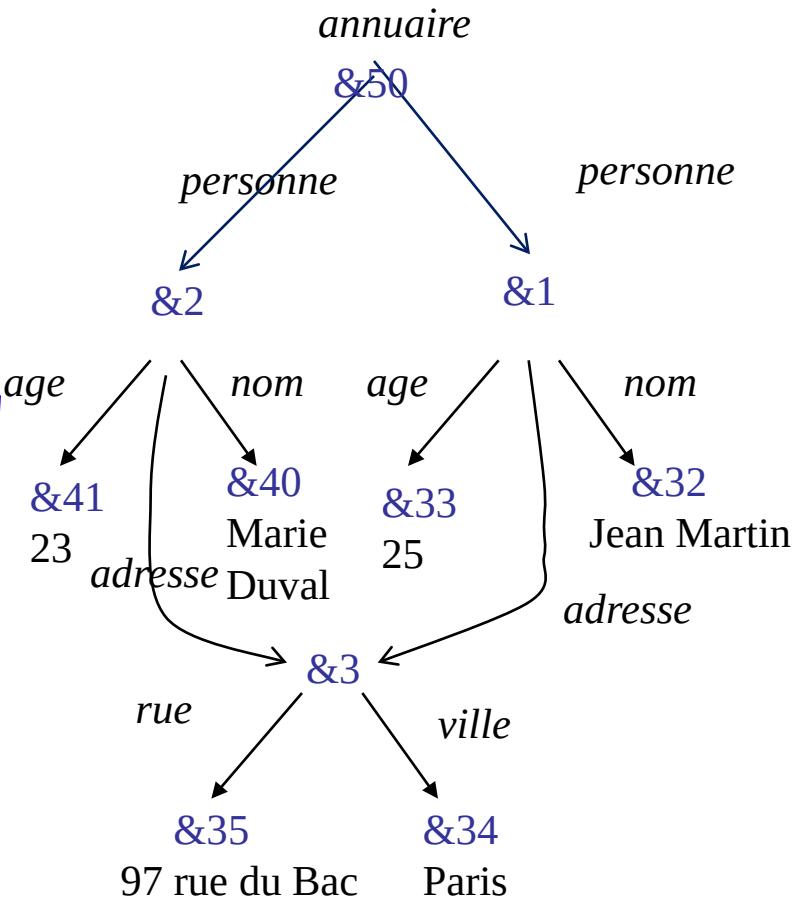
```
<personne &1
   nom &32 "Jean Martin "
   age &33 25
   adresse &3
       rue &35 "97 rue du Bac"
       ville &34 "Paris"

   />
```



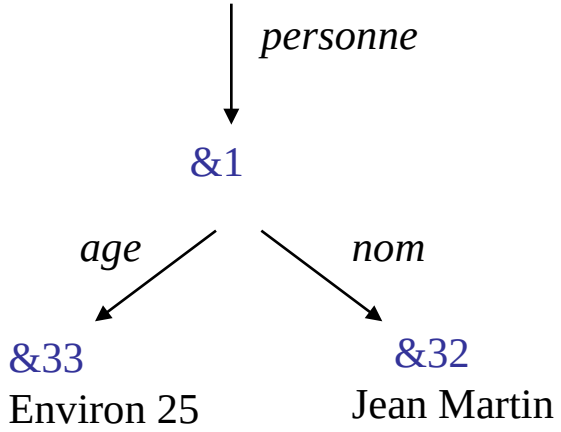
Sous-objets partagés

```
<annuaire &50
personne &1
   nom &32 "Jean Martin "
   age &33 25
   adresse &3
       rue &35 "97 rue du Bac"
       ville &34 "Paris"
personne &2
   nom &40 “Marie Duval“
   age &41 23
   adresse &3
/>
```

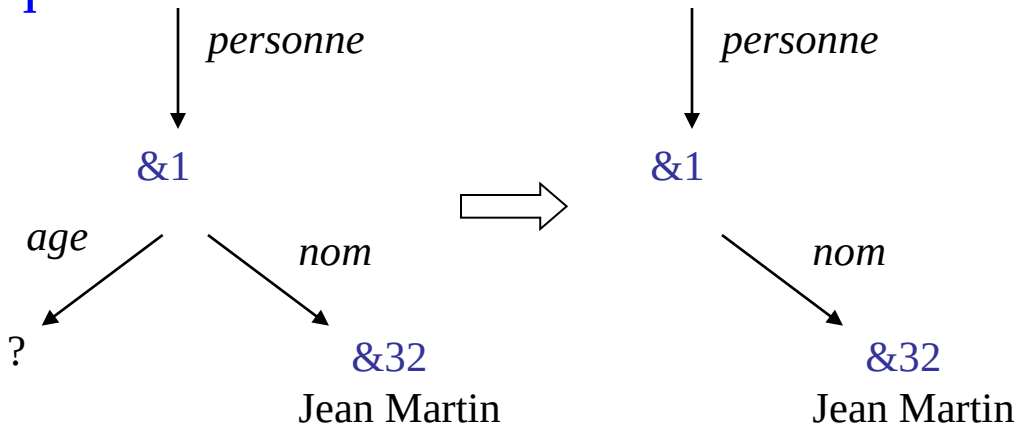


Souplesse de description

- L'âge peut ne pas être un entier

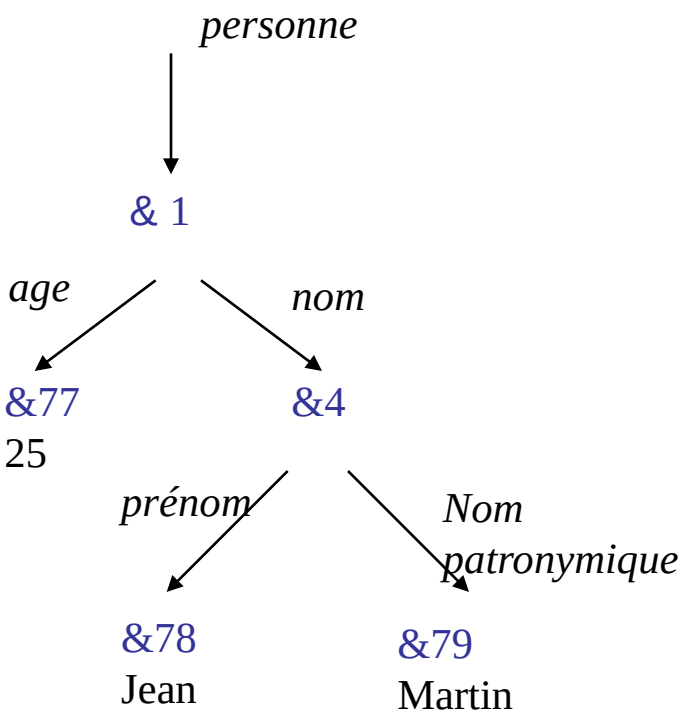


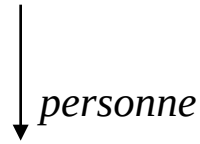
- L'âge peut ne pas être indiqué

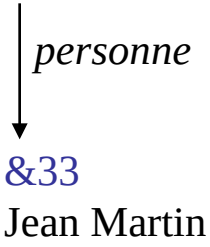


Souplesse de description

- Pas de cohérence dans les structures





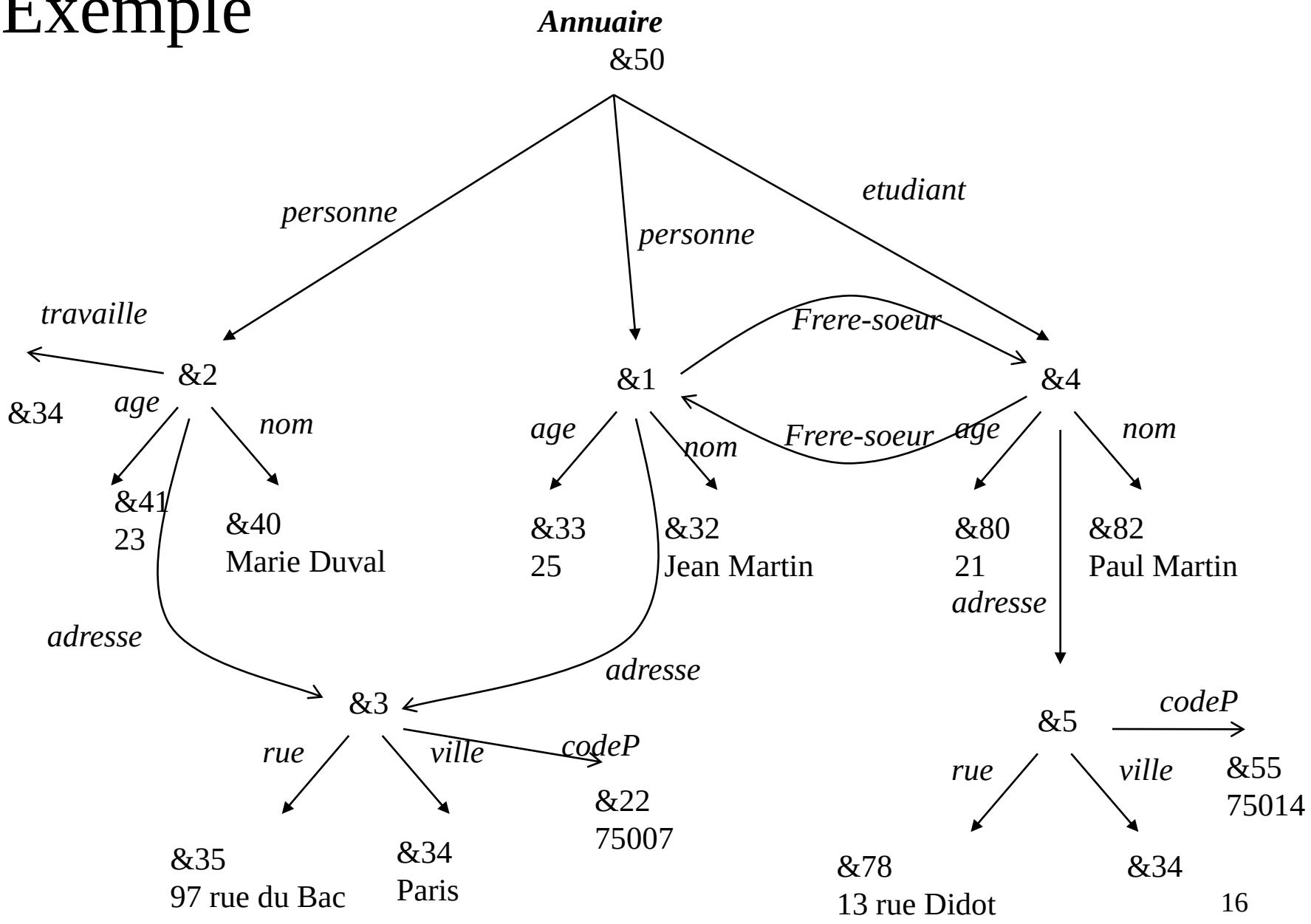


Langage de requêtes

- Les langages classiques ne sont plus appropriés:
  - la structure exacte des données n'est pas connue
  - la structure est irrégulière
  - des concepts similaires sont représentés différemment
  - ensembles hétérogènes
- Le langage doit :
  - S'adapter à la structure souple, irrégulière, mouvante des données
  - Pouvoir interroger sans connaitre la structure exacte des données
  - Rester proche du style SQL/OQL (langage déclaratif simple)

⇨ interrogation navigationnelle, (parcours de graphe) qui s'appuie sur des expressions de chemins.

Exemple


Annuaire
&50
personne
personne
etudiant
travaille
Frere-soeur
&2
&1
&4
&34
age
nom
age
nom
Frere-soeur
age
nom
&41
23
&40
Marie Duval
&33
25
&32
Jean Martin
&80
21
&82
Paul Martin
adresse
adresse
adresse
&3
&5
codeP
rue
ville
codeP
rue
ville
&55
75014
&22
75007
&35
97 rue du Bac
&34
Paris
&78
13 rue Didot
&34

Représentation textuelle

```
Annuaire &50
    Personne &1
     nom &32 “Jean Martin“
     age &33 25
     frere-soeur &4
     adresse &3
      rue &35 “97 rue du Bac“
      ville &34 “Paris“
       codeP &22 75007
    Personne &2
     nom &40 “Marie Duval“
     age &41 23
     adresse &3
     travaille &34
    Etudiant &4
     nom &82 “Paul Martin“
     age &80 21
     frere-soeur &1
     adresse &5
      rue &78 “13 rue Dutot“
      ville &34
       codeP &55 75014
```

Expression de chemin

Expression de chemin (suite d'étiquettes) :

Une expression de chemin est une séquence Z.l1.l2…ln, où les li sont des étiquettes et Z est un nom d'objet (ou une variable dénotant un objet).

Exemple : **`annuaire.nom.adresse`**

Chemin de données :

Un chemin de données est une séquence O_0, l_1, O_1, …, l_n, O_n, où les Oi sont des objets, et pour chaque i, il existe un arc étiqueté li entre Oi-1 et Oi.

Partant d'un objet $Z=O_0$, il peut exister plusieurs chemins de données correspondant à l'expression Z. $l_1.l_2…l_n$.

Expression de chemin

- Naviguer vers un nœud
- Comparaison d'une expression de chemin avec les chemins partant de la racine
- Les nœuds qui correspondent sont retenus
- Valeurs nulles s'il n'y a pas de correspondance

Expressions de chemin

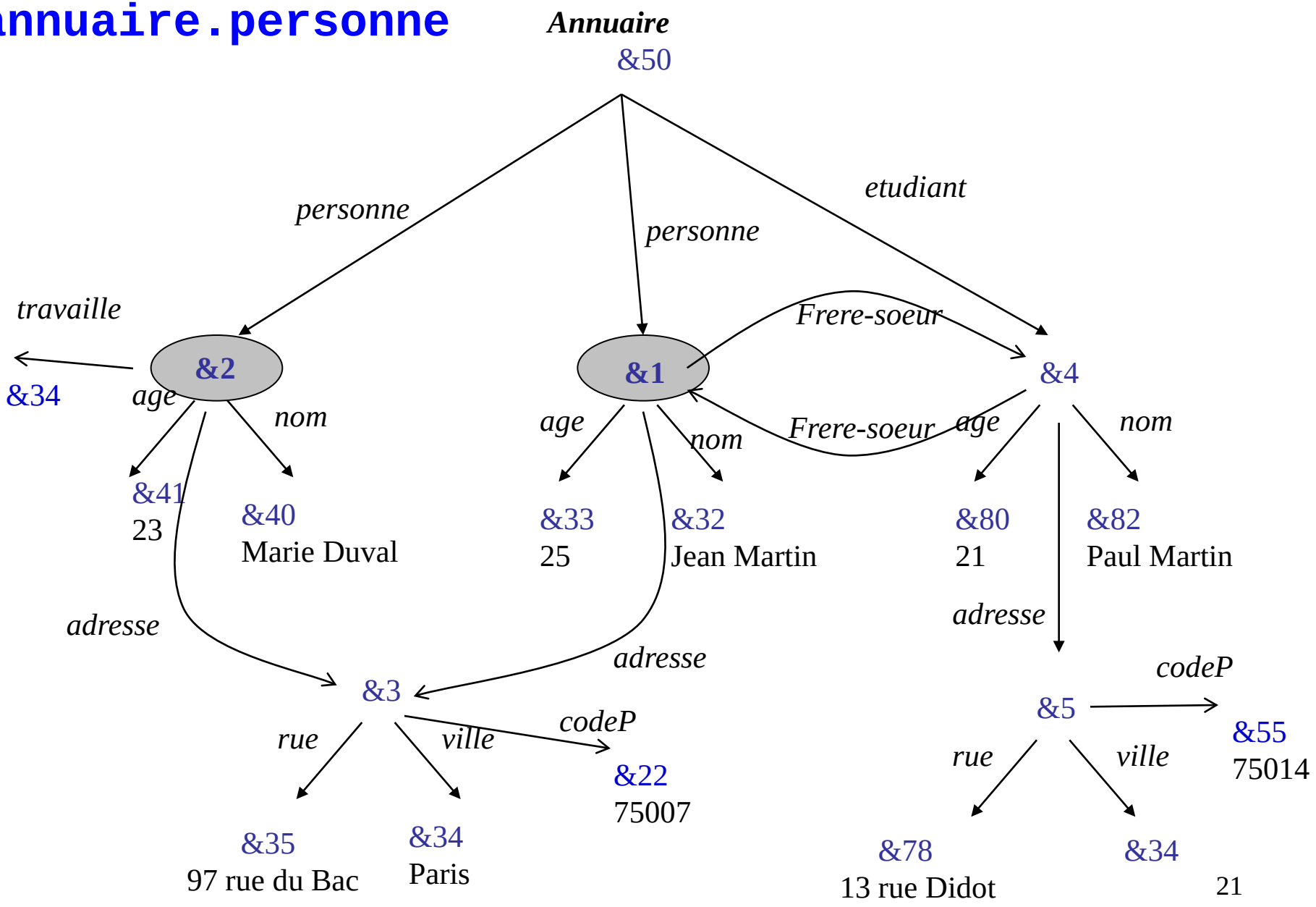
- **Correspondance exacte : `personne`**
- **Correspondance partielle % : `per%`**
- **Alternative | : `personne | etudiant`**
- **Composition X.Y : `personne.nom`**
- **Arc optionnel (.X)? : `personne (.adresse)?.rue`**
- **Nombre arbitraire d'arcs (.X)* :**

  `personne(.frere-soeur)*.nom`
- **« joker » signifie qu'il y a un nombre indéfini d'étiquettes non spécifiées # :**

  `personne.#.ville`

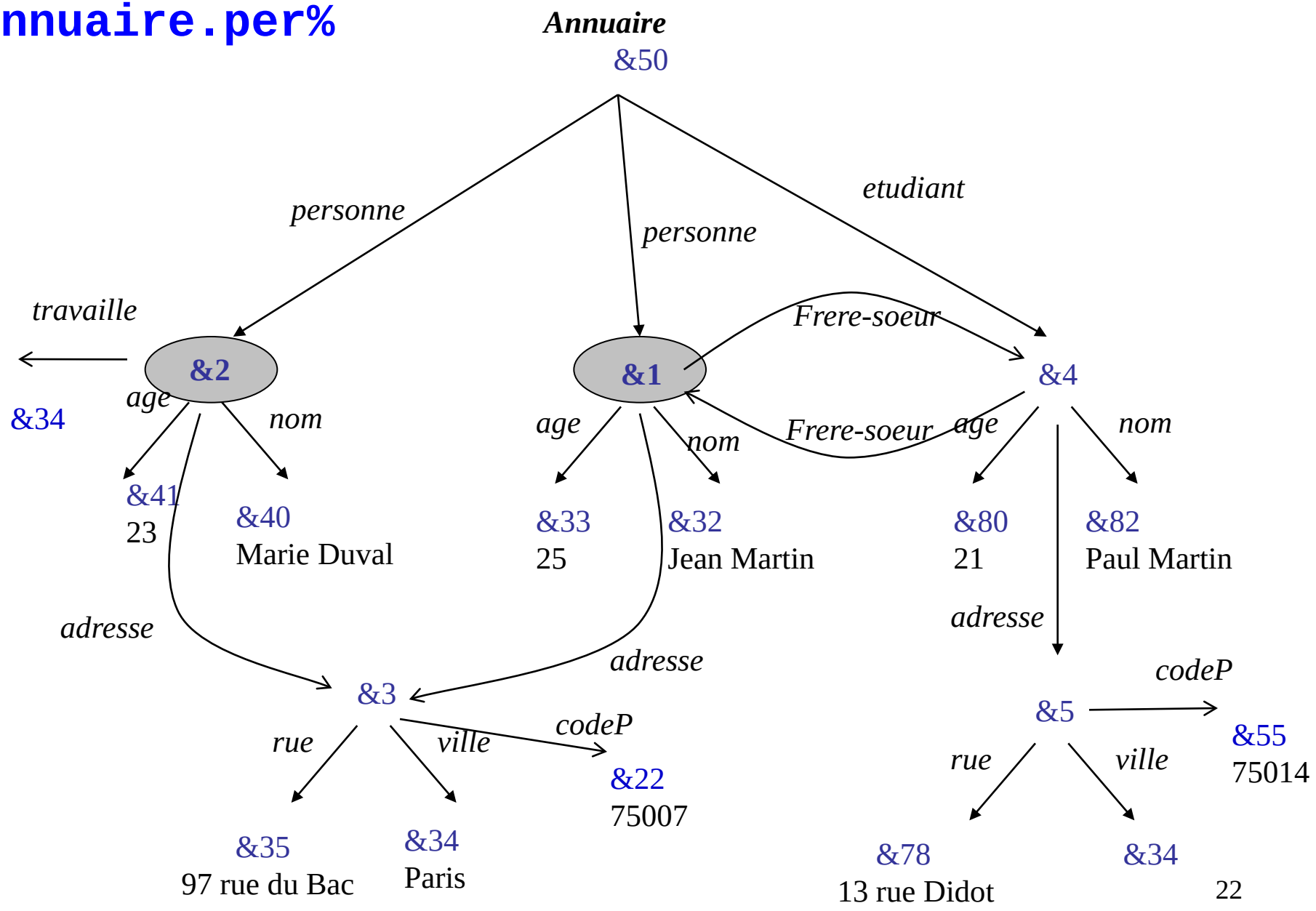
Correspondance exacte :

annuaire.personne

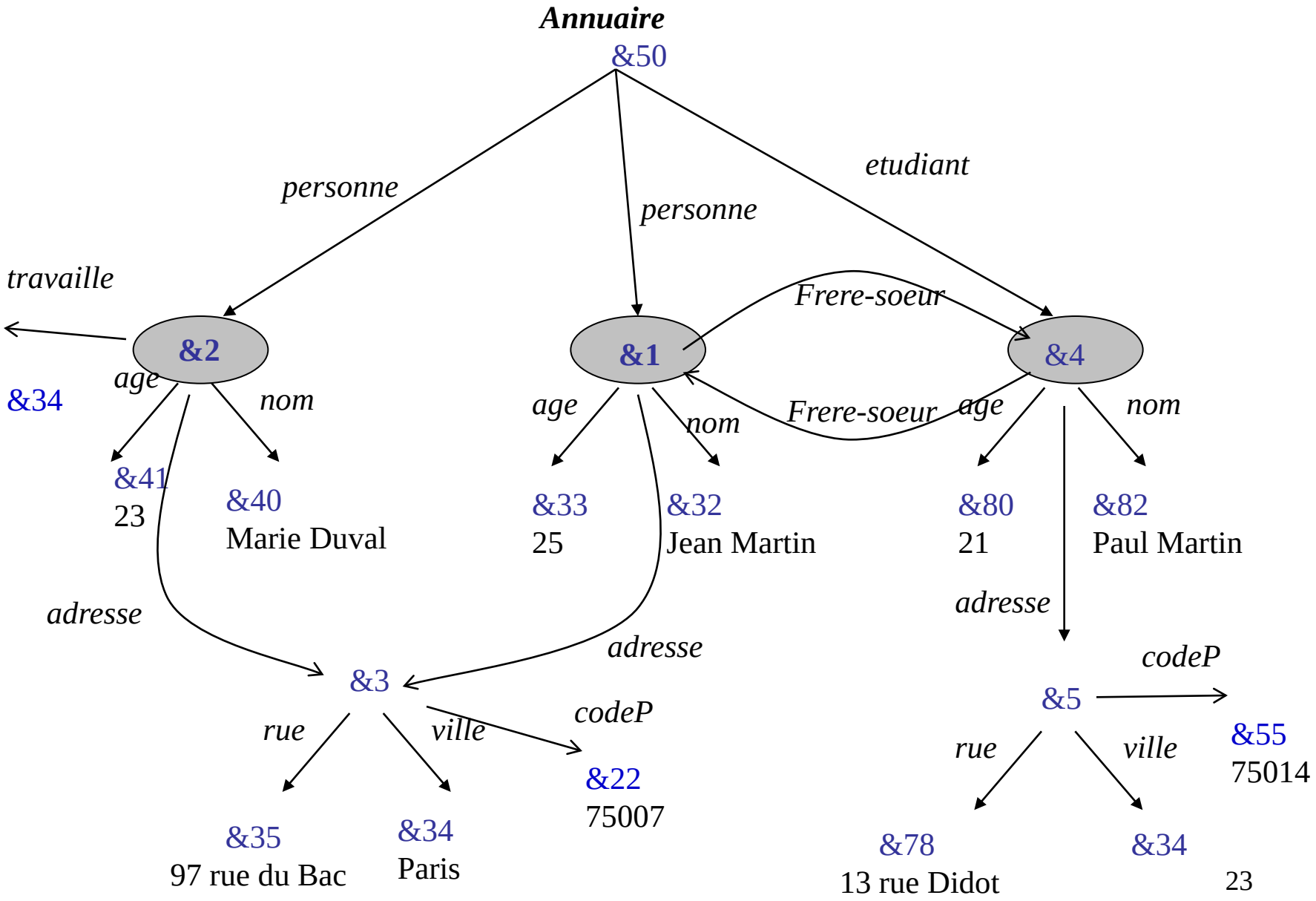


Correspondance partielle :

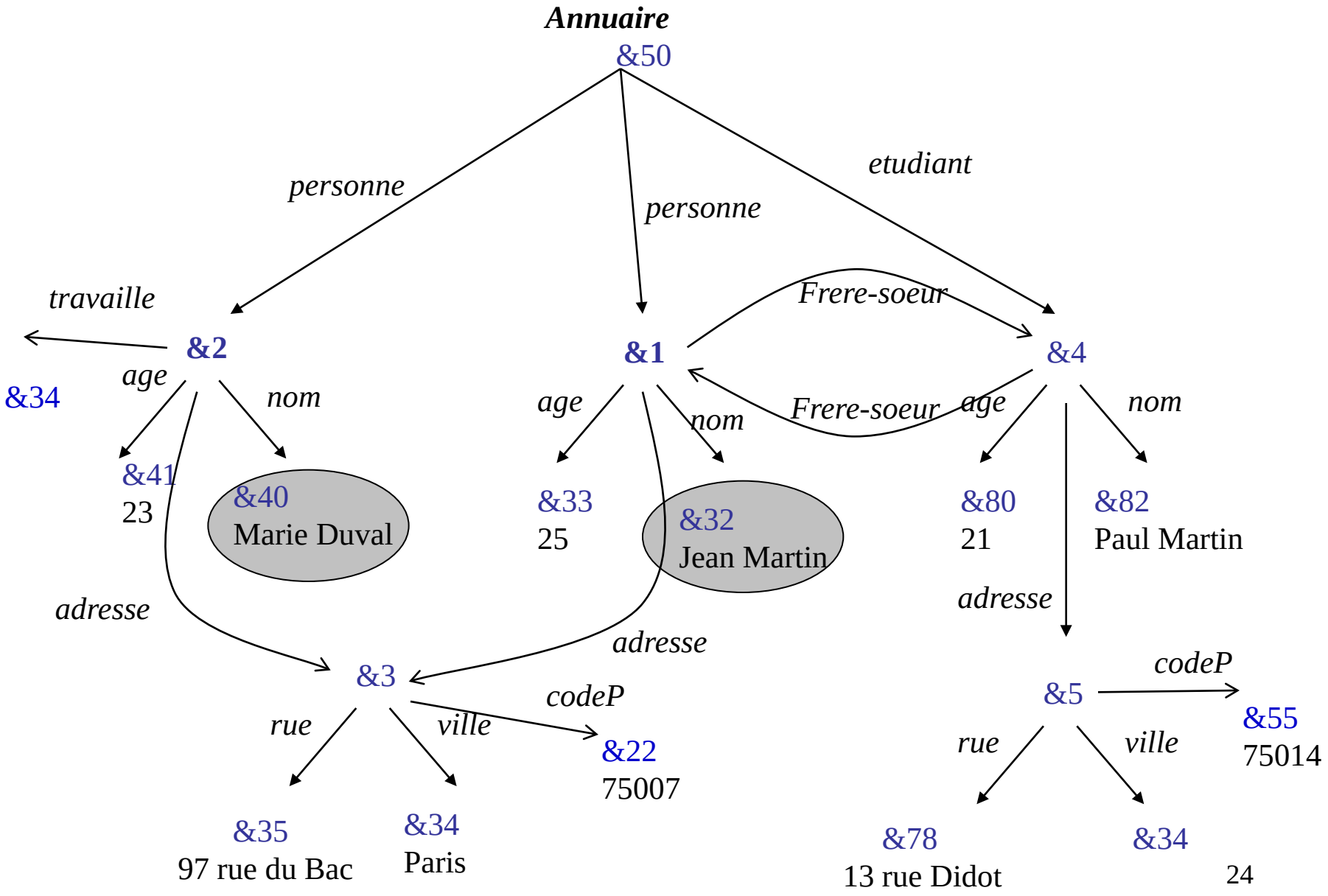
annuaire.per%



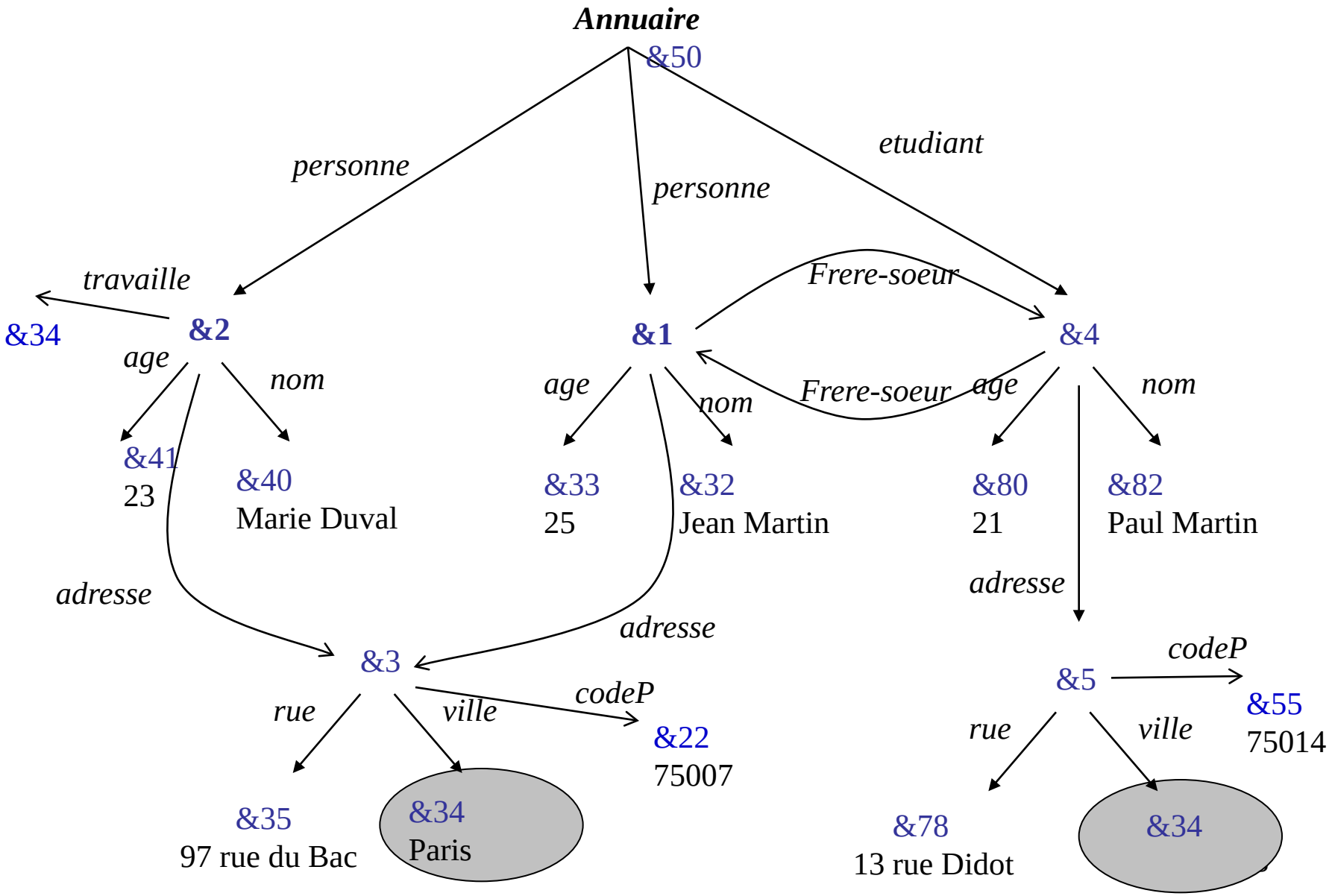
Alternative : **annuaire.(personne | etudiant)**



Composition X.Y : annuaire.personne.nom



« joker » # : **personne.#.ville**



XML
Extensible Markup Language

Standard du W3C (1998)

Le langage XML : une syntaxe, un format d'échange

Description de documents XML :

- DTD (Structure type de document)
- XSchéma

Interrogation :

- XPath
- XQuery

XML

Objectifs

- Echange, partage, diffusion et publication de documents
- Recherche d'information : moteurs de recherche généralisée, portails spécialisés
- Commerce électronique : billetterie, catalogues électroniques,…

Format/langage standard pour les données semi-structurées du Web

- Consortium W3C (Oracle, IBM, MS, MIT, INRIA….) 1996
- Version 1.0 en 1998, nombreux développements depuis.
- Successeur de HTML, héritier de SGML

XML : Principes

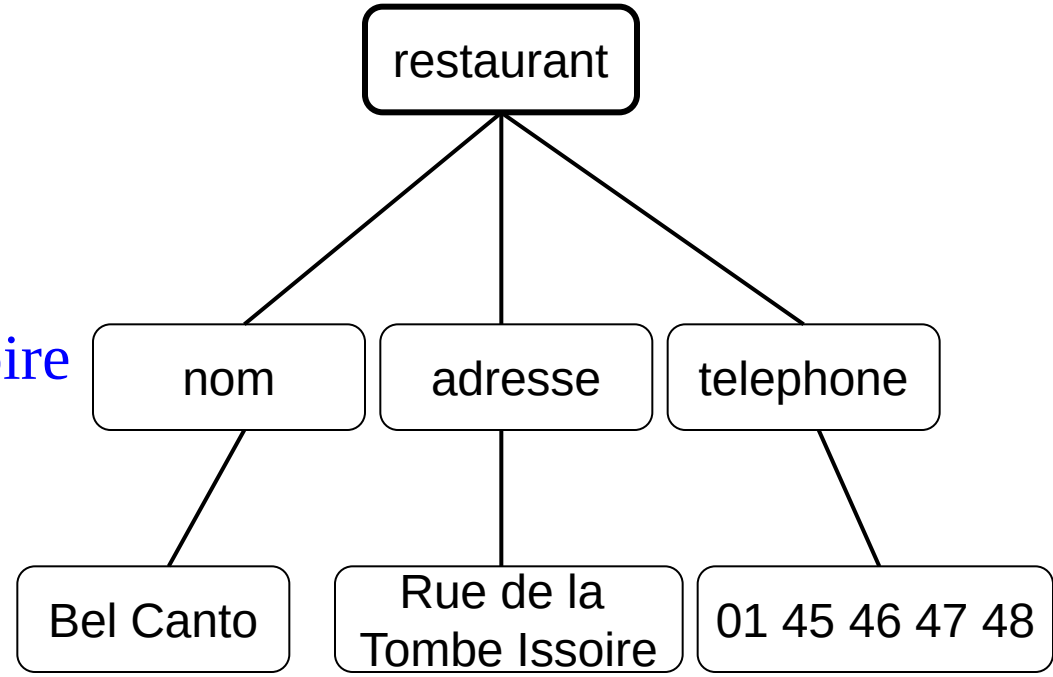
- Séparer la *structure logique* des données de leur *présentation*
- Feuille de style (XSL) :
  - ensemble de règles pour la réalisation sur un médium particulier
- Structure hiérarchique :
  - un document XML est composé d'éléments, structurés en hiérarchie. Tout document possède un élément racine.
- Balisage structurel (SGML) :
  - chaque élément est encadré par des balises (tag) ouvrante et fermante : <cours>MLBDA</cours>
- Balisage défini par les auteurs :
  - souplesse

Forme sérialisée et forme arborescente

- Un document XML peut se représenter sous deux formes:
  - **La forme sérialisée** est la forme courante (contenu marqué par des balises). Elle est utilisée pour
    - Stocker un document dans un fichier
    - Echanger des documents
  - **La forme arborescente** met en évidence la structure du document (facilite la conception des traitements).
    - Elle permet de spécifier des manipulations de données XML
    - Elle peut être utilisée par certaines applications qui gèrent les documents en mémoire (éditeurs XML).

Exemple

```
<restaurant>
    <nom>Bel Canto </nom>
    <adresse>
        Rue de la Tombe-Issoire
    </adresse>
    <telephone>
        01 45 46 47 48
    </telephone>
</restaurant>
```



La structure des documents est définie par le *Document Object Model* (DOM)

Structure d’un document

- Un document XML est composé
  - d’un prologue (facultatif)
  - d’un arbre d’éléments (obligatoire : décrit le contenu)
  - de commentaires et instructions de traitement (facultatif)

- Un document peut être découpé en entités enregistrées dans un ou plusieurs fichiers

Exemple

```
<?xml version="1.0" encoding='ISO-8859-1'
  standalone='yes'>
<personne>
  <nom>Martin</nom>
  <prenom>Jean</prenom>
  <adresse>
     <rue>rue du Bac</rue>
     <ville>Paris</ville>
  </adresse>
<!-- pas d'autre information disponible -->
</personne>
```

Prologue

- Déclaration XML (indications au processeur)

  `xml version= ‘‘1.0’’` : décrit la version utilisée

  `encoding=‘ISO-8859-1’:` codage de caractères utilisés dans le document

  `standalone=‘‘yes’’` : existence de déclarations extérieures (yes = toutes les déclarations nécessaires au document sont incluses)

- Le prologue peut contenir la déclaration d’une DTD

Arbre d’éléments

`<nom>contenu</nom>`

- Un document est formé d’une hiérarchie d’éléments dénotant la structure sémantique du contenu.
- Tout élément fils est complètement inclus dans son père.
- L’élément racine est unique et contient tous les autres éléments.
- Un élément peut contenir d’autres éléments, des données, des références à des entités, des sections littérales, des instructions de traitement. Il peut aussi être vide.

Quelques règles de syntaxe

- Tout élément doit avoir une balise ouvrante et une balise fermante.
- La hiérarchie doit être respectée (un élément totalement inclus dans l'élément père)
- Un nom de balise doit commencer par une lettre, un _, ou : (pas de chiffre, de caractères réservés ?, /, %, …). Il ne doit pas contenir d'espace.
- On distingue les minuscules des majuscules

Structures types de documents
DTD (Document Type Definition)

La DTD est composée d'une suite de déclarations pour *tous* les

- éléments,
- attributs,
- entités,
- notations

utilisés dans le document.

Elle peut aussi contenir des commentaires.

Elle peut être incluse dans le document ou externe (standalone = 'no').

Elément

<!ELEMENT nom modele>

Modèle de contenu :

- Nom de sous-élément (fils) (ordonnés ou non)

  Séquence (fils1, fils2,…), choix (fils1 | fils2…)

  – Ex: **<!ELEMENT chapitre (titre, intro, (texte-long | résumé))>**

  Indicateur d'occurrence :

  * : nombre quelconque

  + : 1 ou plusieurs fois

  ? : 0 ou 1 fois

  – Ex : **<!ELEMENT chapitre (titre, intro?, paragraphe+)>**

- **#PCDATA** *Parsed Character Data* : du texte à l'intérieur d'un élément (pas d'éléments fils)

  - Ex: **<!ELEMENT titre (#PCDATA)>**

Modèles de contenu (suite)

- Modèle de contenu mixte

  (#PCDATA | élément$_1$ | élément$_2$ | … | élément$_n$)*

  – Mêler données et éléments

  – Ex : **<!ELEMENT paragraphe (#PCDATA | em | exp | ind)*>**
  <paragraphe>Du texte en évidence , 9 = 3<exp>2</exp></paragraphe>
- Modèle de contenu vide ***EMPTY***
  – Obligatoirement vide (infos uniquement dans attributs)
  – Ne peut pas être composé (pas d'éléments fils)
  – Syntaxe <element attribut=… />
- Modèle de contenu libre ***ANY***
  – Contenu quelconque (autres éléments et données)
  – Sert à prototyper des DTD complexes

Attributs d'un élément

- Les attributs sont un **autre** moyen de représenter l'information

```
<rapport langue='français'
  date='2014'>
```

- L'ordre des attributs n'est pas important.
- Un attribut doit toujours avoir une valeur, encadrée par des apostrophes simples ou doubles.
- Il ne peut pas y avoir deux attributs de même nom dans un élément.

Les attributs ne sont pas ordonnés alors que les éléments le sont

Attributs d'un élément

- Déclaration :

  <!ATTLIST nom-élément nom-attribut type-attribut mode-défault>

  – On peut grouper la déclaration des attributs d'un même élément

  – Type-attribut :
    - CDATA : la valeur de l'attribut est une chaîne de caratères
    - ID : identificateur d'élément, IDREF(S) : renvoi vers un (des) ID
    - NMTOKEN(S) : un ou des noms symboliques (sans blanc)
    - (a | b | c…) : type énuméré.
    - ENTITY(IES) : entités externes non XML

  – Mode :
    - Valeur par défault
    - Obligatoire : #REQUIRED
    - Facultatif : #IMPLIED
    - Constante : #FIXED

Exemples d’attributs

Dans la DTD :

<!ELEMENT date (#PCDATA) >

<!ATTLIST date format (ANSI | ISO | FR) #REQUIRED>

Dans le document :

<date format=‘FR’> 8 octobre 2014 </date>

<date format=‘ISO’>2014-10-08</date>

Exemples d'attributs ID et IDREF

Dans la DTD :

```
<!ELEMENT personne (nom, …)>
<!ATTLIST personne num ID #REQUIRED>
<!ELEMENT livre (titre,auteur+,…)>
<!ELEMENT auteur EMPTY>
<!ATTLIST auteur ref IDREF #REQUIRED>
. . .
```

Dans le document :

```
<personne num='p1'><nom>Toto</nom></personne>
<personne num='p2'><nom>Titi</nom></personne>
<livre>
    <titre>Les bases de données en s'amusant</titre>
    <auteur ref='p1'/>
    <auteur ref='p2'/>
</livre>
```

La référence se fait à un élément quelconque du document (pas forcément une personne) ayant un attribut ID de valeur 'p1'

Entités

- Caractère, chaîne de caractères, fragment, fichier externe…
  - Déclarée par un nom : ***nom-entité*** et une valeur ***val***
  - Appelée dans le document par ***&nom-entité***
  - Interprétation : remplacer ***&nom-entité*** dans le document par ***val***
- Permet de réutiliser quelque chose défini ailleurs (macro, raccourci)
- Plusieurs types d'entités :
  - **Entités internes** : définies localement, comme une chaîne de caractères
  - **Entités externes** : font référence à des fichiers externes
  - **Entités prédéfinies** et **entités caractères** : référencent des caractères réservés en XML et des caractères qui ne sont pas sur le clavier

Entités prédéfinies et entités internes

- Entités prédéfinies :
  - amp (&), apos (’), quot (’’), gt (>), lt (<)
  - #code-unicode
- Entités internes définies par l’utilisateur dans la DTD :

<!ENTITY nom-entité ‘‘valeur’’>

Exemple :

<!DOCTYPE mon-document [
<!ENTITY copyright ‘‘© Editions UPMC.’’>]>

<mon-document> &copyright </mon-document>

Produira : © Editions UPMC.

Entités externes xml

- Déclaration et appel

  <!ENTITY carte-de-visite SYSTEM ‘‘cdv.xml’’>

  <message> … &carte-de-visite; </message>
- Les entités externes doivent être bien formées

Rappel : un document xml est bien formé si :

- ses éléments sont correctement imbriqués (pas de chevauchement)
- chaque attribut a une valeur unique

En d’autres termes, un document xml est bien formé s’il représente un arbre étiqueté.

NE PAS CONFONDRE : document **bien formé** (/xml) et document **valide** (/DTD)

Notations et entités non-xml

- Déclaration :

  <!NOTATION gif SYSTEM ‘‘/usr/bin/xv’’>

  <!ENTITY maphoto

  SYSTEM ‘‘./mesphotos/mapomme.gif’’ NDATA gif>

  <photo img=‘maphoto’>

  – Uniquement appel dans un attribut (pas besoin de &)
  – La déclaration de notation associée au NDATA indique que xml ne traitera pas cette entité.
  – La notation indique de plus quelle application utiliser (xv)

Entité paramètre

Entité paramètre : entité déclarée dans une DTD pour être utilisée dans cette DTD.

Déclarée dans la partie interne (dans la partie DOCTYPE)

<!ENTITY % nom-entite "valeur entite" >

La référence à l'entité (dans la DTD) se fait par %nom-entité.

```
<!-- Dans la DTD -- >
<!ENTITY %pub "&#xc9;ditions ToutSavoir"  >
<!ENTITY rights "Tous droits réservés" >
<!ENTITY book "J.Martin. Le  Web et les BD, %pub. &rights">
...
<!-- Dans le  document -- >
<p> &book </p>
```

On obtient :

J. Martin. Le Web et les BD, Éditions ToutSavoir. Tous droits réservés.

Rôles des DTD

- Modèle selon une organisation hiérarchique (définition des éléments, attributs, contenus)
- Spécifie la structure des instances de documents : cet élément contient ces éléments, ces attributs, etc.
- Spécifie le type de données de chaque élément et attribut
- Définition d'entités : mécanisme d'inclusion (interne, externe, paramètre) utile pour les opérations de modularisation et de réutilisation

Exemple
Document XML

```
<?xml version="1.0" standalone="no" ?>
<!–existence et adresse d’une DTD externe -->
<!DOCTYPE document SYSTEM "cours.dtd">
<document>
   <titre>Cours de M1.</titre>
    <nom IDCours=‘MI005’>MABD</nom>
     <prerequis>
           <nom IDCours=‘LI341’>Bases de Données</nom>
     </prerequis>
   <controle>examens répartis</controle>
    <contenu>ODMG. SQL3.XML.XPATH.QUERY</contenu>
</document>
```

Exemple
DTD correspondante

Fichier cours.dtd

```
<!ELEMENT document
  (titre*, nom,prerequis?,controle,contenu)>
<!ELEMENT prerequis (nom)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT nom (#PCDATA)>
<!ATTLIST nom IDCours ID #REQUIRED>
<!ELEMENT controle (#PCDATA)>
<!ATTLIST controle type-controle (examen | partiel) #IMPLIED>
<!ELEMENT contenu (#PCDATA)>
```

Limites des DTD

- Syntaxe spécifique : une syntaxe pour les documents, une pour leur définition
- Pas de possibilité de typer les contenus (types limités)
- Typage faible des valeurs d'attributs

Les DTD ne sont pas suffisantes pour l'échange de données structurées (BD, commerce électronique, etc.).

⇨ Schéma XML