M1 Informatique -Sorbonne Université

MU4IN801 - **MLBDA**

Modèles et Langages Bases de Données Avancées

Bernd Amann

COURS 4 – DONNÉES DU WEB
MODÈLE SEMI-STRUCTURÉ ET XML

Plan

Modèles de données semi-structurées

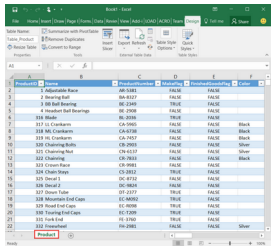
- Besoins
- Caractéristiques générales

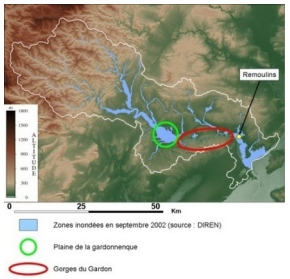
OEM (Object Exchange Model)

XML (Extensible Markup Language)

JSON (Javascript Object Notation)

Gestion de données avant





Recherche d'information

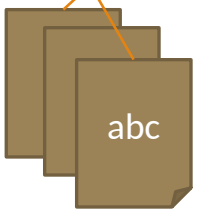
Requêtes Transactions

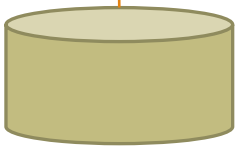
Métadonnées

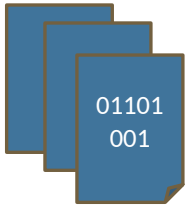
Systèmes documentaires (texte)

Systèmes de Gestion de Bases de Données

Systèmes multimédia





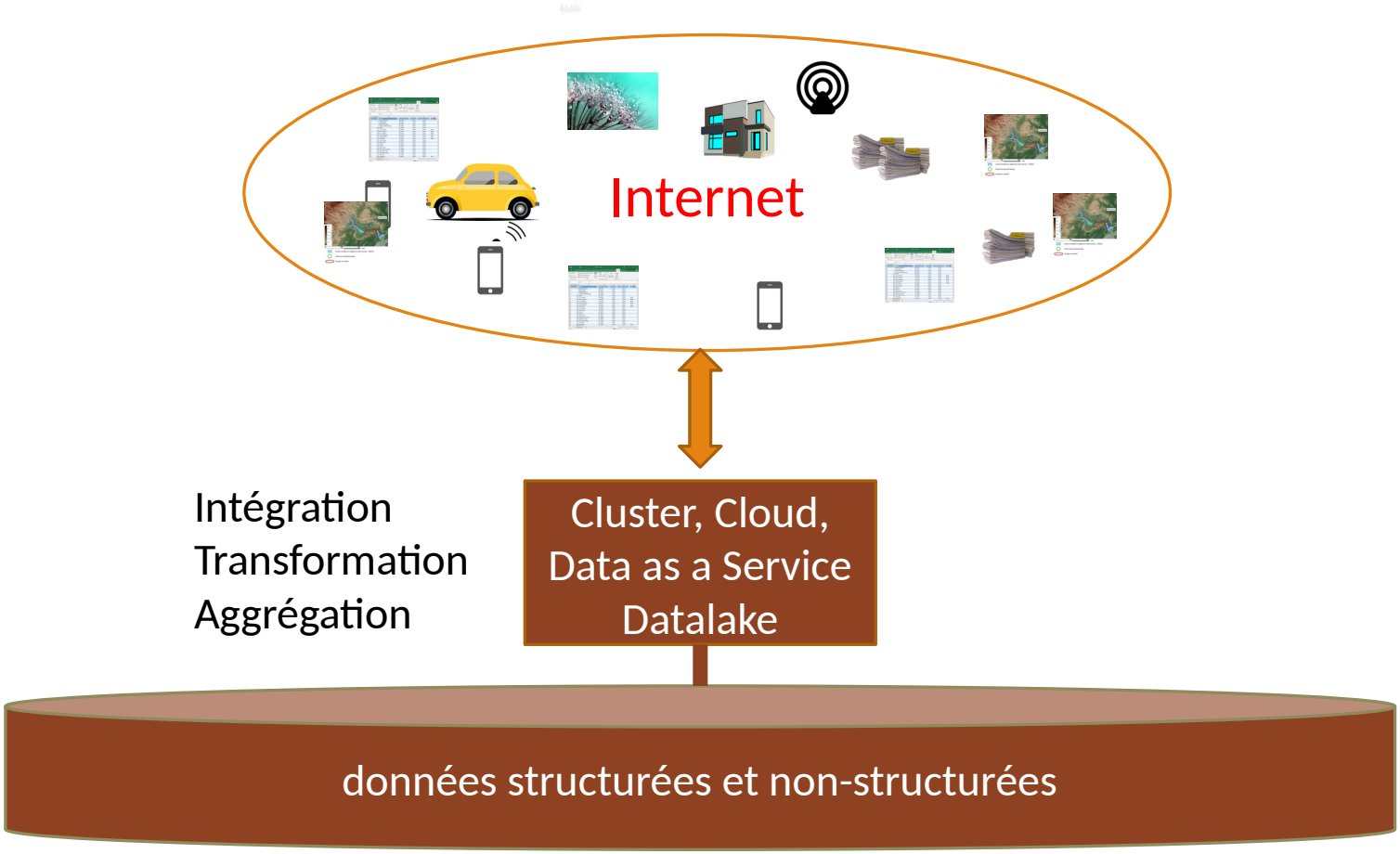


texte

données structurées

audio, images, vidéos

Gestion de données aujourd'hui



Quel modèle de données ?

Utiliser les modèles de données structurées (relationnel, relationnel-objet) pose des problèmes :

- la structure est **trop rigide**
- les données peuvent ne pas être conformes au schéma
- **l'évolution** de la structure de données conduit à des évolutions de schéma difficiles à mettre en œuvre.

Nécessité d'un modèle général et **souple**, **sans trop de contraintes**, avec un *langage de requêtes* associé
⇒ *modèles semi-structurés*

Caractéristiques des Modèles de Données ***Sémi-structurées***

DONNÉES

Structure irrégulière :

- Ex: adresse sous forme de nuplet avec et sans codes postaux et sous forme de chaîne de caractères

Structure « implicite » :

- Ex: documents textuels avec des sections et sous-sections

Données partielles / incomplètes :

- Ex: adresses avec et sans numéros de rues

SCHÉMAS

Schéma descriptifs

- Description et validation de données déjà existantes
- Plusieurs schémas possibles pour les mêmes données (précision versus taille)

Schéma complexes et volumineuses

- Ex: données irrégulières: taille données ~ taille schéma

Schémas évolutifs

- Ex: intégration de données ⇒ modification du schéma

REQUÊTES

Requêtes « flexibles »:

- Interrogation de données et de schémas

Requêtes approximatives

- Ignorer le schéma
- Interroger avec connaissance partielle de la structure

Modèle de données semi-structurées

Principe :

- partir des données et trouver une structure commune, souple.

Les modèles semi-structurés utilisent des graphes étiquetés pour représenter les données.

Les modèles diffèrent par

- le *rôle des étiquettes* : type, valeur, identifiant, …
- l'endroit des étiquettes : arêtes et/ou nœuds
- l'existence ou non d'un ordonnancement
- La représentation du partage d'information (références)

OEM (Object Exchange Model)

Bases de données OEM = objet

Chaque objet a

- une étiquette (chaîne de caractères)
- un identificateur
- une valeur avec son type (*optionnels*).

Objets atomiques

- valeurs atomique : integer, real, string, html, audio, java, etc.

Objets complexe:

- valeur = **collection d'identificateurs** (références) : set, list

Représentation graphique

personne &1 set

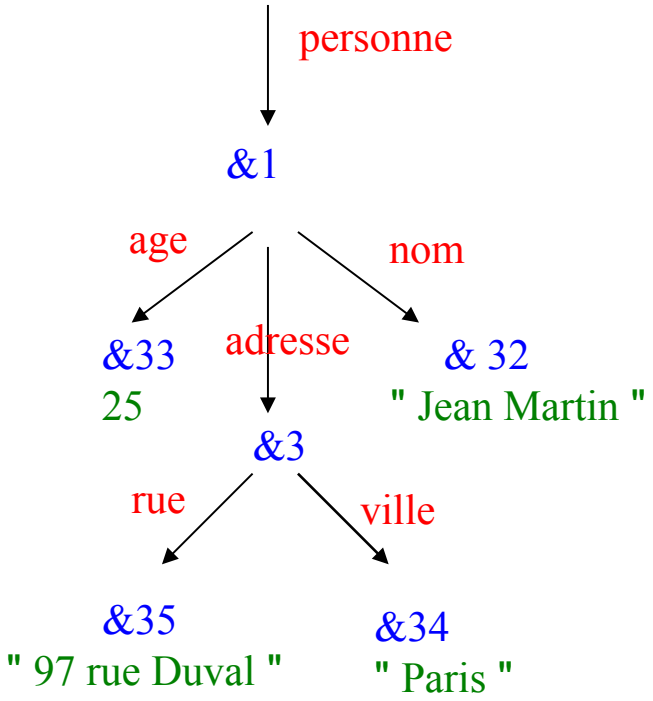
nom &32 string "Jean Martin "

age &33 integer 25

adresse &3 set

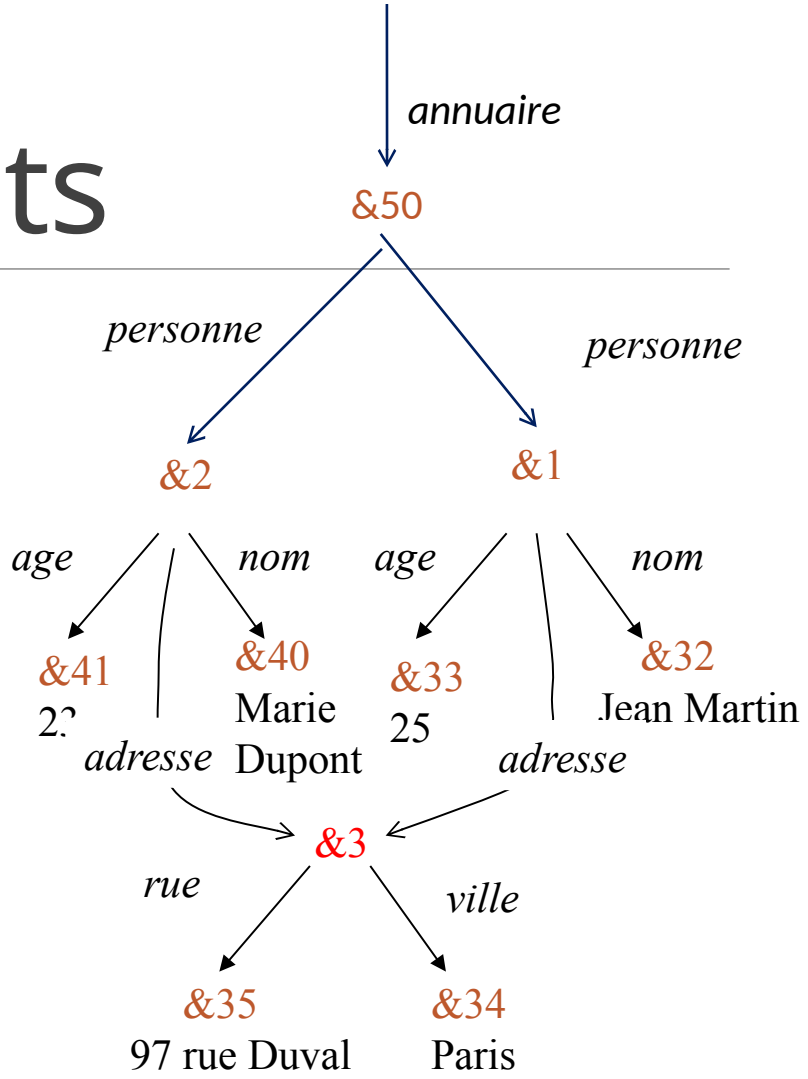
rue &35 string "97 rue Duval"

ville &34 string "Paris"



Partage d'objets

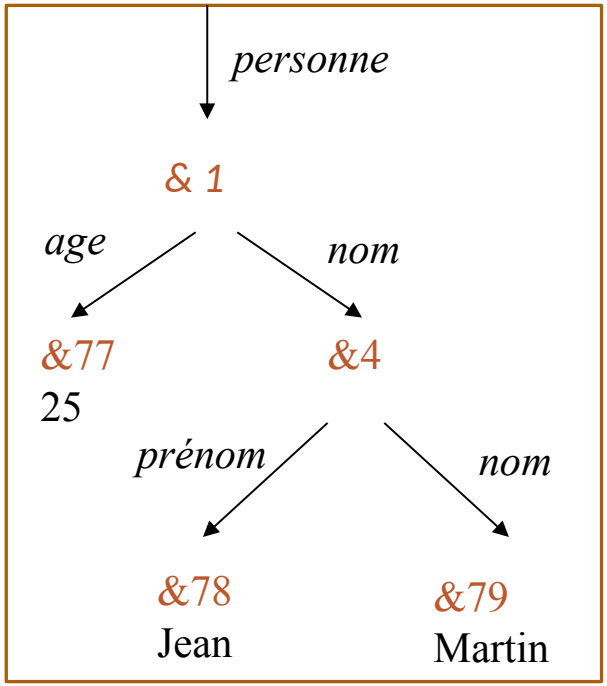
annuaire &50
  personne &1
    nom &32 "Jean Martin "
      age &33 25
      adresse &3
        rue &35 "97 rue Duval"
        ville &34 "Paris"
  personne &2
    nom &40 "Marie Dupont"
    age &41 23
    adresse &3

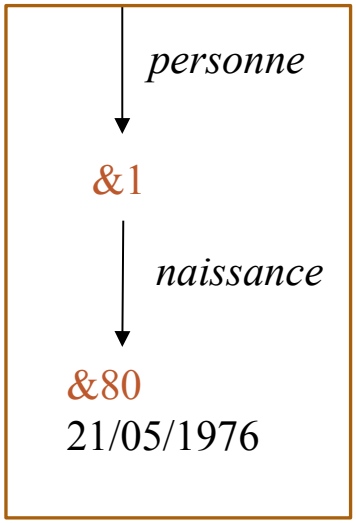


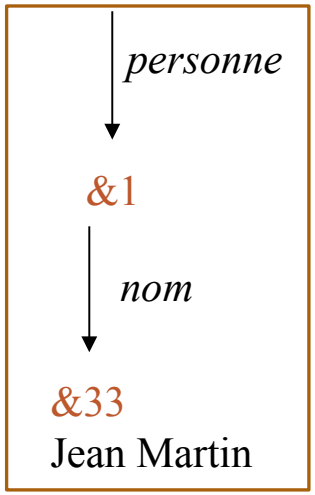
Souplesse de structuration

Co-existence de structures diverses pour les mêmes types d'entités

Exemple : Jean Martin



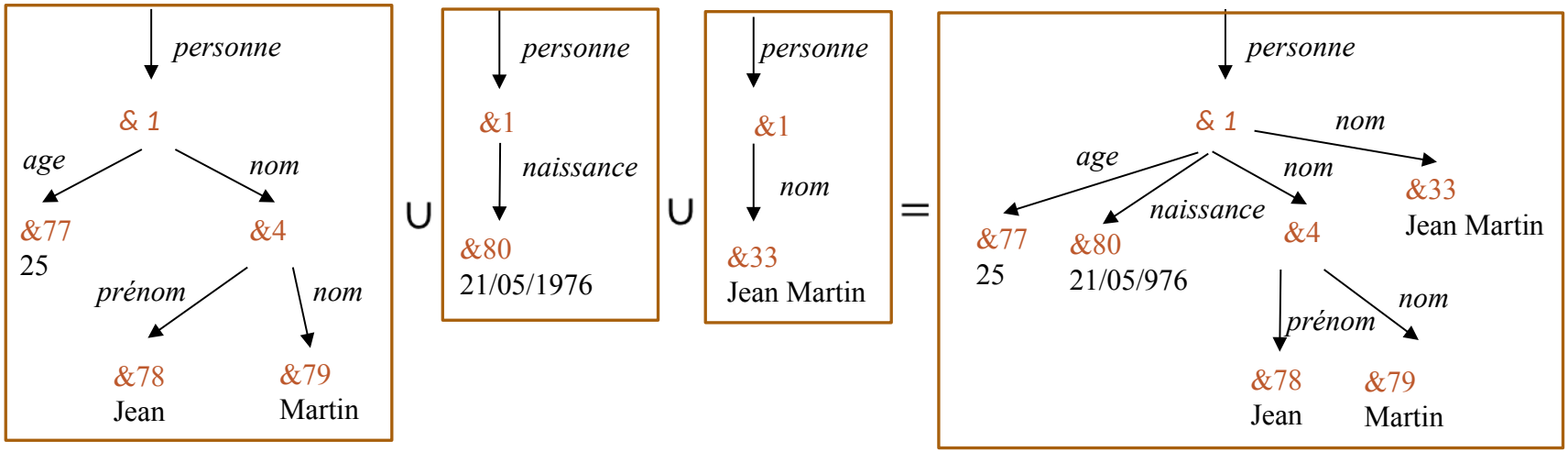




Integration « facile »

Co-existence de structures diverses pour les mêmes types d'entités

Exemple : **personne**



Langages de requêtes semi-structurés

Les langages classiques ne sont plus appropriés:

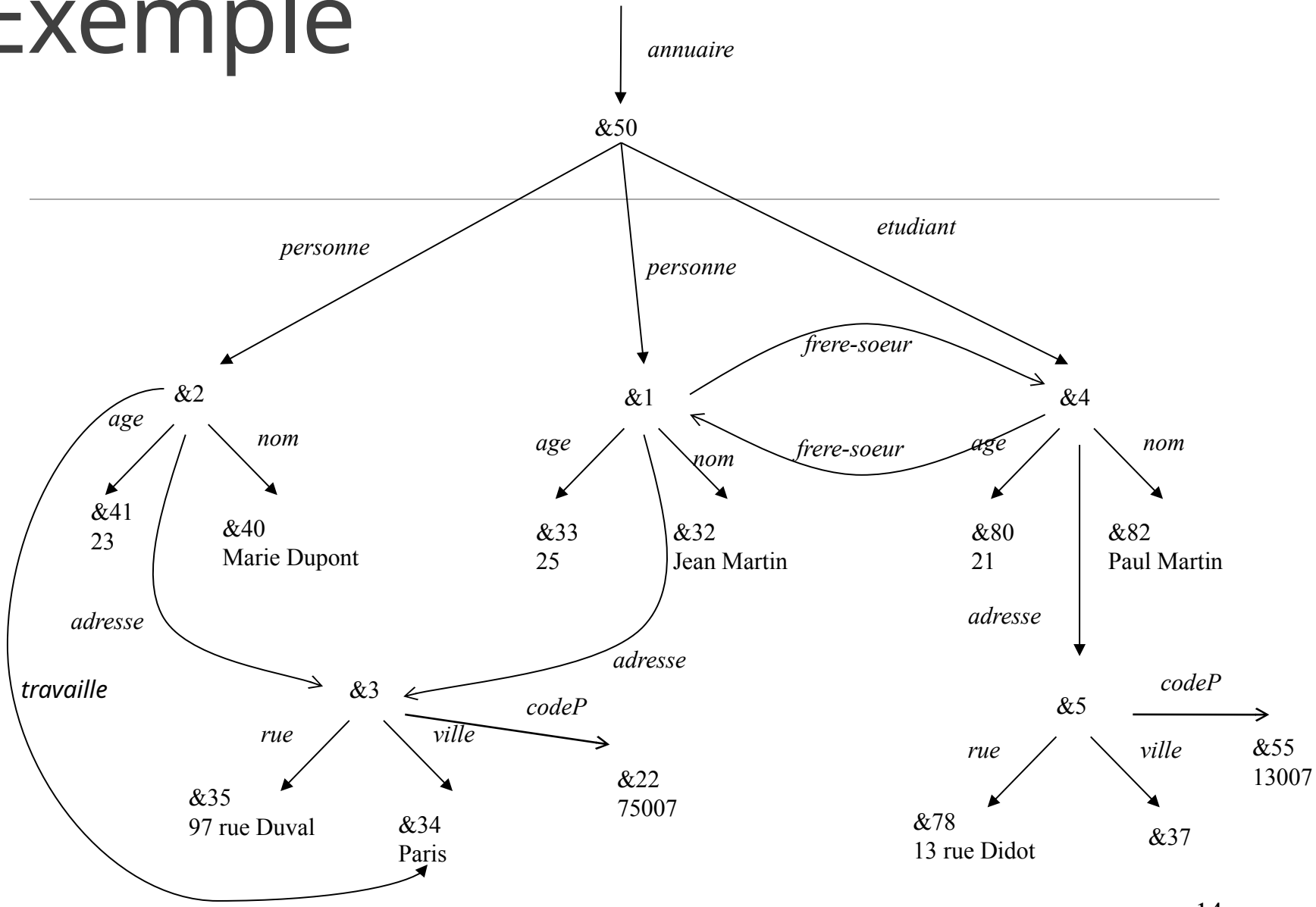
- la structure exacte des données n'est pas connue
- la structure est irrégulière
- des concepts similaires sont représentés différemment
- ensembles hétérogènes

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)

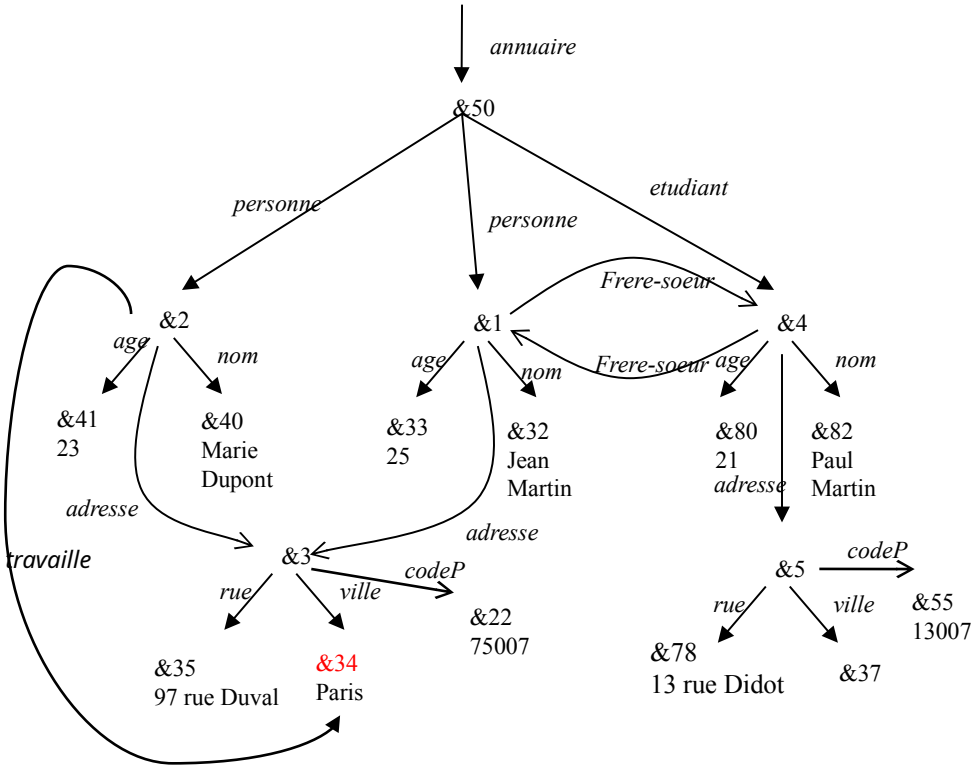
⇒ interrogation *navigationnelle* qui s'appuie sur des *expressions de chemins*.

Exemple


annuaire
&50
personne
personne
etudiant
frere-soeur
frere-soeur
&2
&1
&4
age
nom
age
nom
age
nom
&41
23
&40
Marie Dupont
&33
25
&32
Jean Martin
&80
21
&82
Paul Martin
adresse
adresse
adresse
travaille
&3
&5
codeP
codeP
rue
ville
rue
ville
&35
97 rue Duval
&34
Paris
&22
75007
&78
13 rue Didot
&37
&55
13007

Représentation textuelle

annuaire &50
- Personne &1
  - nom &32 “Jean Martin“
  - age &33 25
  - frere-soeur &4
  - adresse &3
    - rue &35 “97 rue Duval“
    - ville &34 “Paris“
    - codeP &22 75007
- Personne &2
  - nom &40 “Marie Dupont“
  - 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 Didot“
    - ville &37
    - codeP &55 75014



Expression de chemin

Expression de chemin (suite d'étiquettes) :

- Une expression de chemin est une séquence $Z.\,l_1\,.\,l_2\,\ldots\,\mathrm{l_n}$, où les l_i sont des étiquettes et Z est un objet (ou une variable dénotant un objet).

Exemple : annuaire.nom.adresse

Chemin de données :

- Un chemin de données est une séquence $O_0, l_1, O_1, \ldots, \mathrm{l}_n\ , O_n$, où les O_i sont des objets, et pour chaque i, il existe un arc étiqueté li entre O_{i-1} et O_i.

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

Expressions de chemin

Correspondance exacte : personne

Correspondance partielle % : per%

Alternative | : personne | etudiant

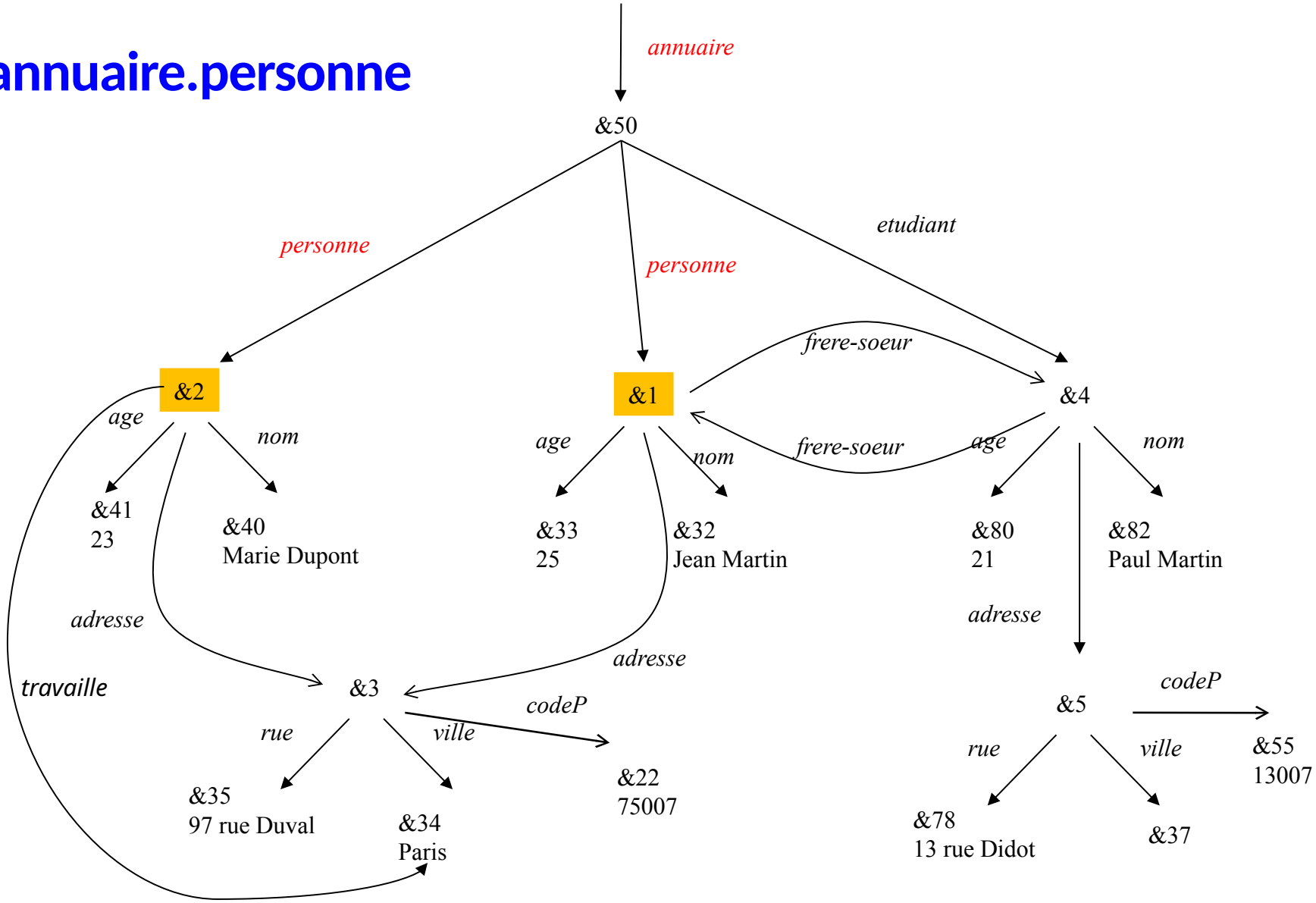
Composition X.Y : personne.nom

Arc optionnel (.X)? : personne (.adresse)?.rue

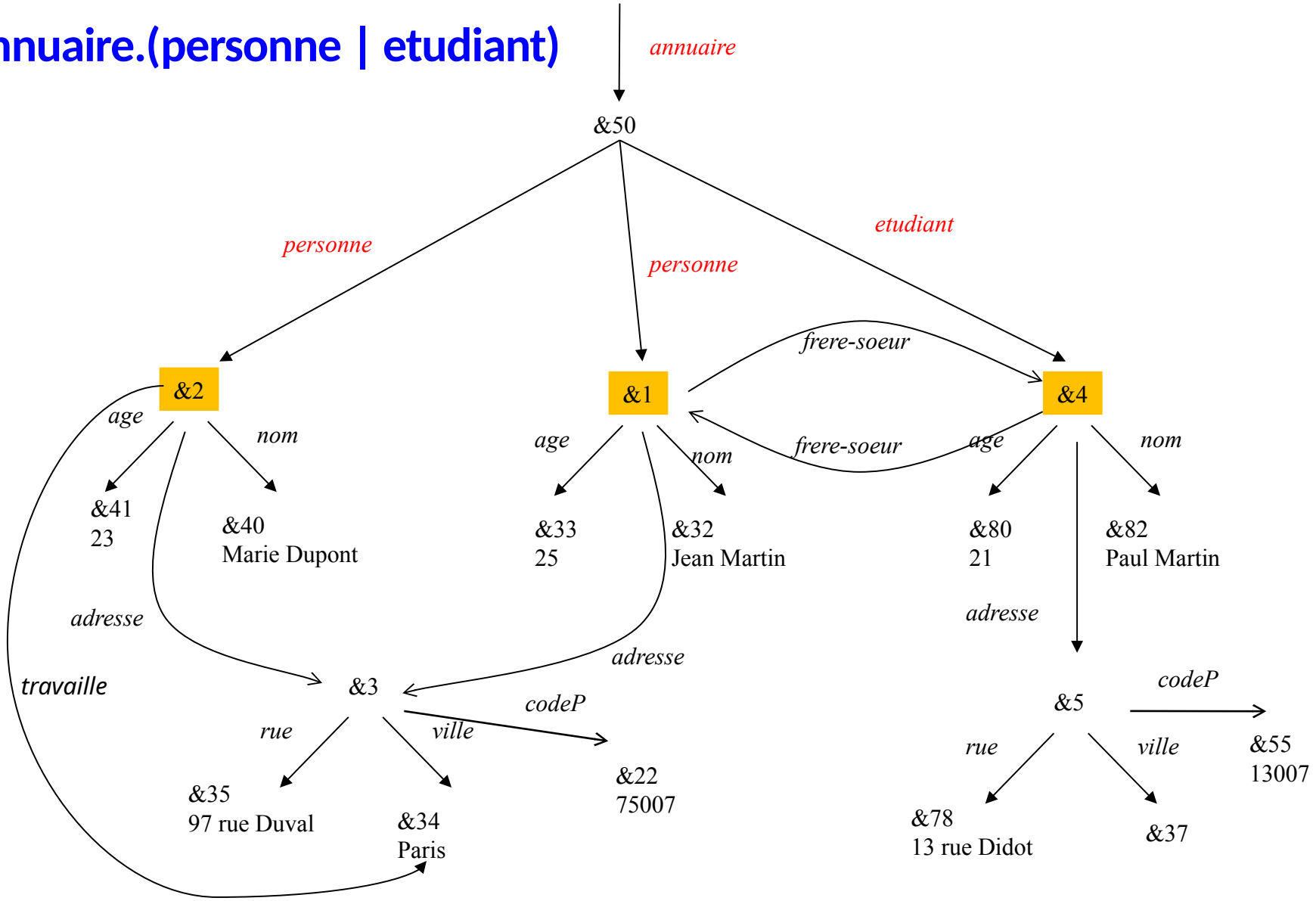
Nombre arbitraire d'arcs (.X)* : personne(.frere-soeur)*.nom

« joker » pour un nombre indéfini d'étiquettes non spécifiées # : personne.#.ville

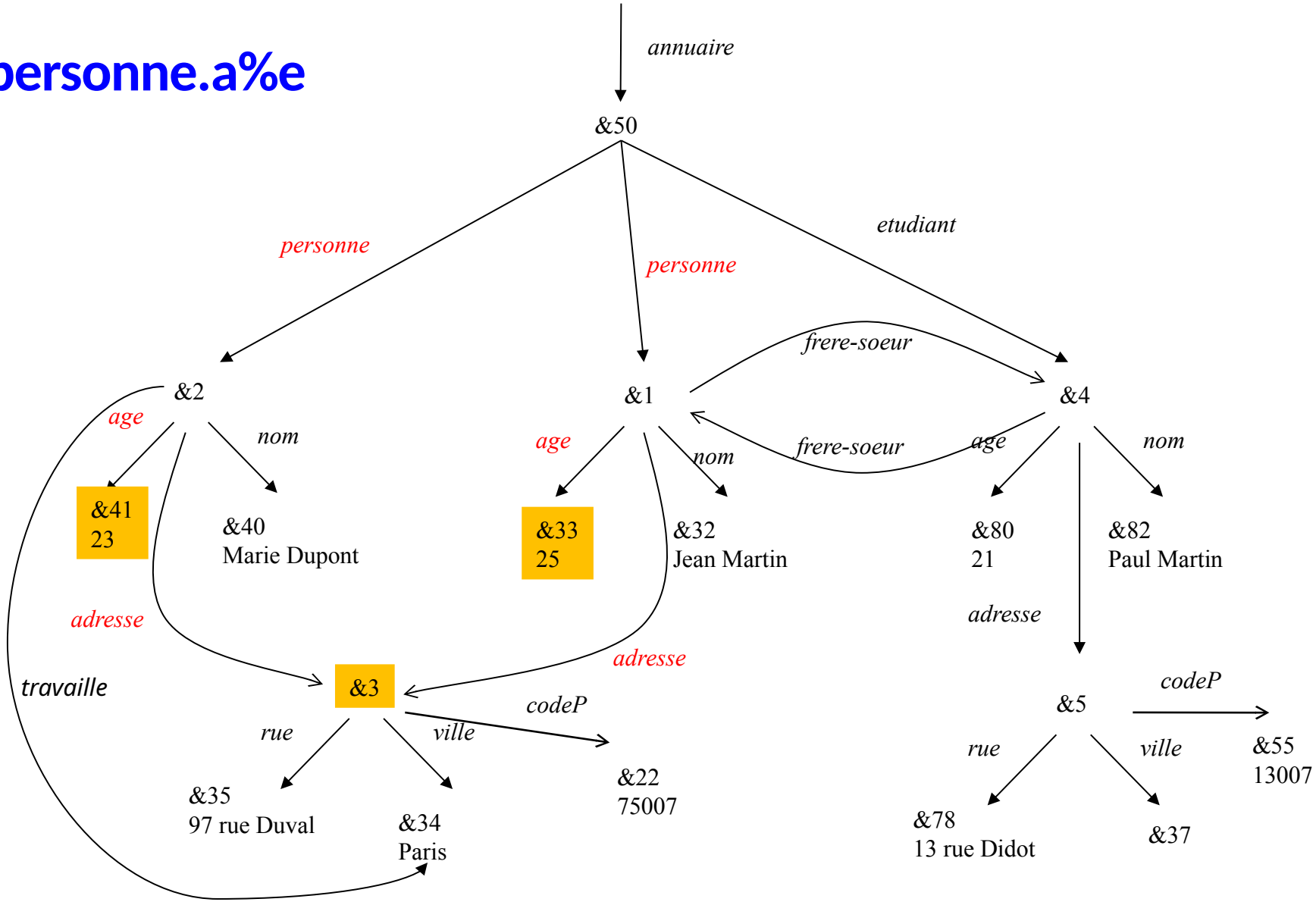
annuaire.personne



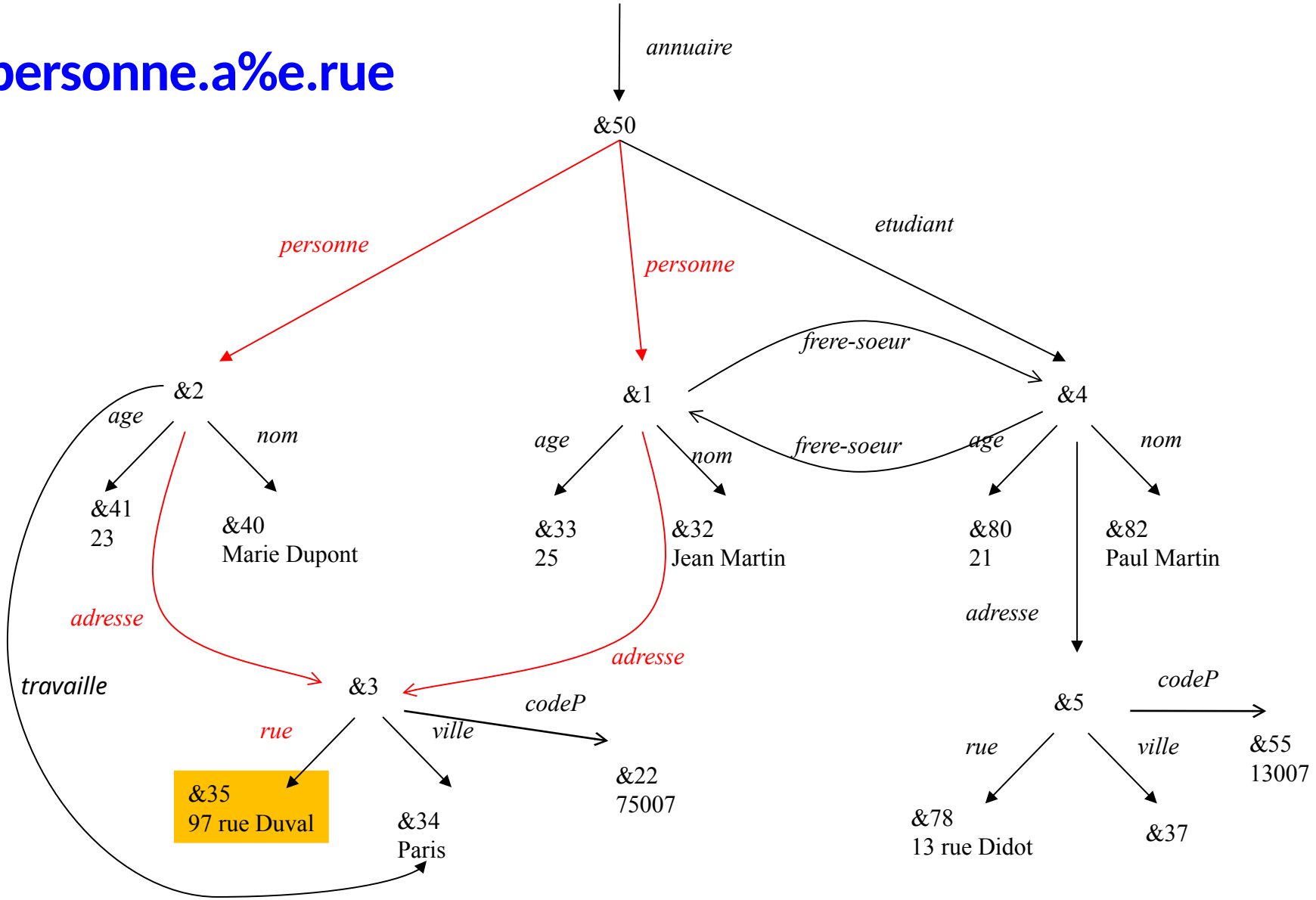
annuaire.(personne | etudiant)



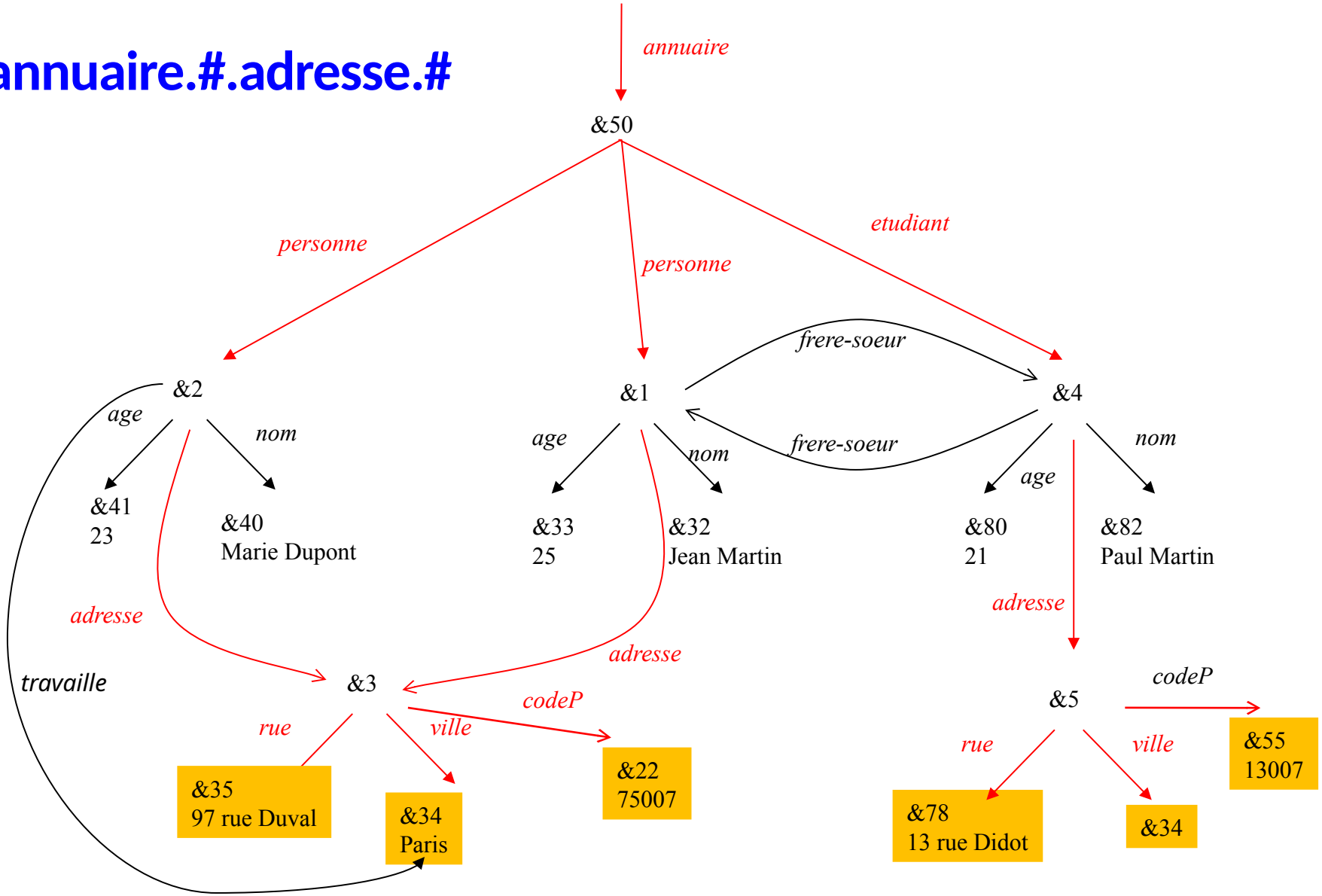
personne.a%e


annuaire
&50
personne
personne
etudiant
&2
&1
&4
frere-soeur
frere-soeur
age
nom
age
nom
age
nom
&41
23
&40
Marie Dupont
&33
25
&32
Jean Martin
&80
21
&82
Paul Martin
adresse
adresse
adresse
travaille
&3
&5
codeP
codeP
rue
ville
rue
ville
&35
97 rue Duval
&34
Paris
&22
75007
&78
13 rue Didot
&37
&55
13007

personne.a%e.rue



annuaire.#.adresse.#



Conclusion sur OEM

OEM : un des premiers modèles de données semi-structurées

Montre l'importance de pouvoir interroger les données et leur structure (« schéma »)

Modèle *auto-descriptif*:

- les étiquettes permettent de décrire l'information et des l'accéder
- schéma : description sans imposer des contraintes
- requêtes :
  - expressions de chemins sur les étiquettes
  - accès structuré sans connaissance à priori du schéma

De la théorie à la pratique ⇒ XML

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

- Échange, 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

Structure hiérarchique :

- un document XML est composé **d'éléments**, structurés en **hiérarchie**. Tout document possède **un élément racine**.

Balisage structurel :

- chaque élément est encadré par des balises (tag) ouvrante et fermante : **<cours>**MLBDA**</cours>**
- les noms des balises sont définies par les auteurs : séparer la structure logique des données de leur présentation (règles de transformation)

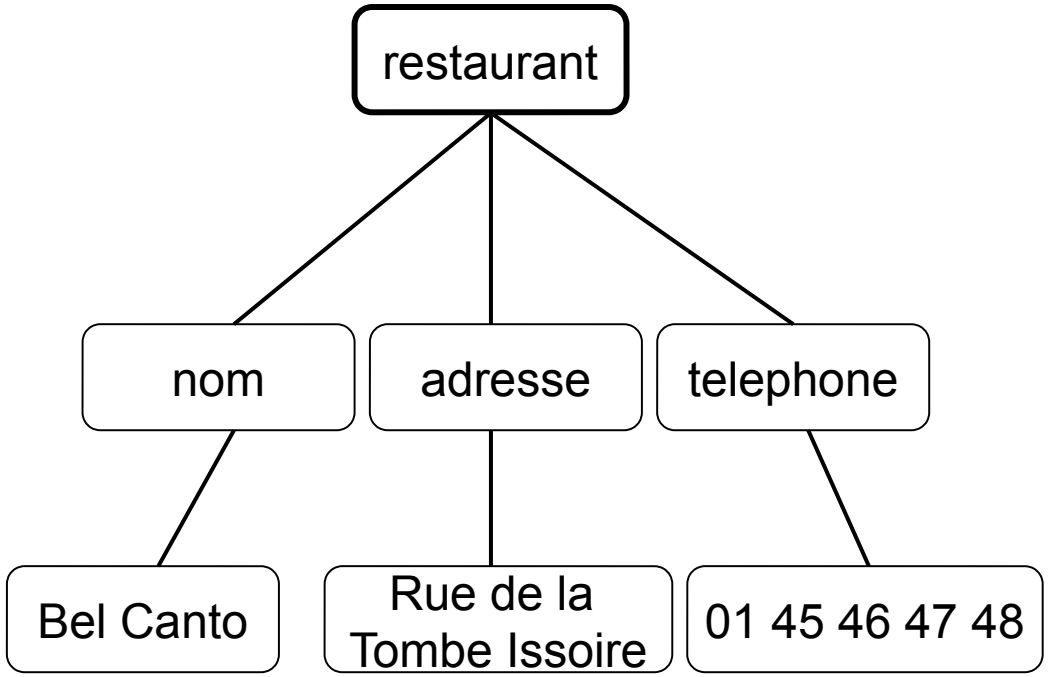
Forme sérialisée et forme arborescente

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

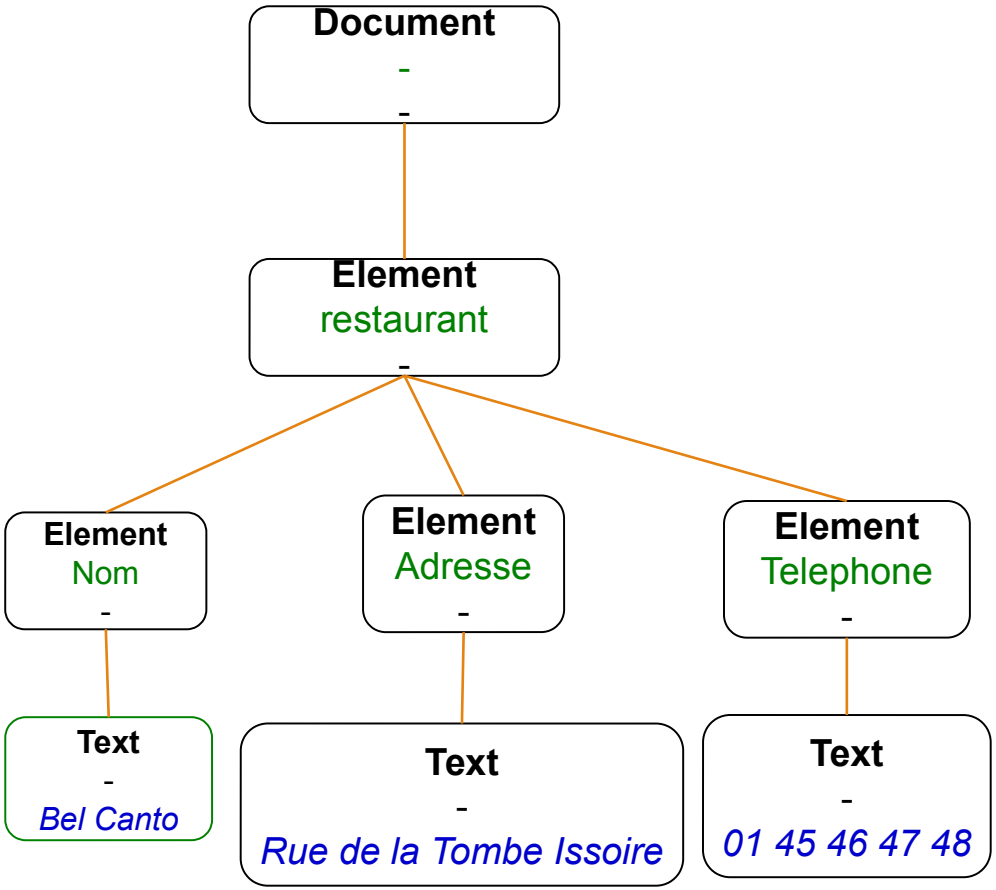
```
<restaurant>
   <nom>Bel Canto </nom>
   <adresse>
         Rue de la Tombe-Issoire
   </adresse>
   <telephone>
         01 45 46 47 48
   </telephone>
</restaurant>
```



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

Arbres DOM

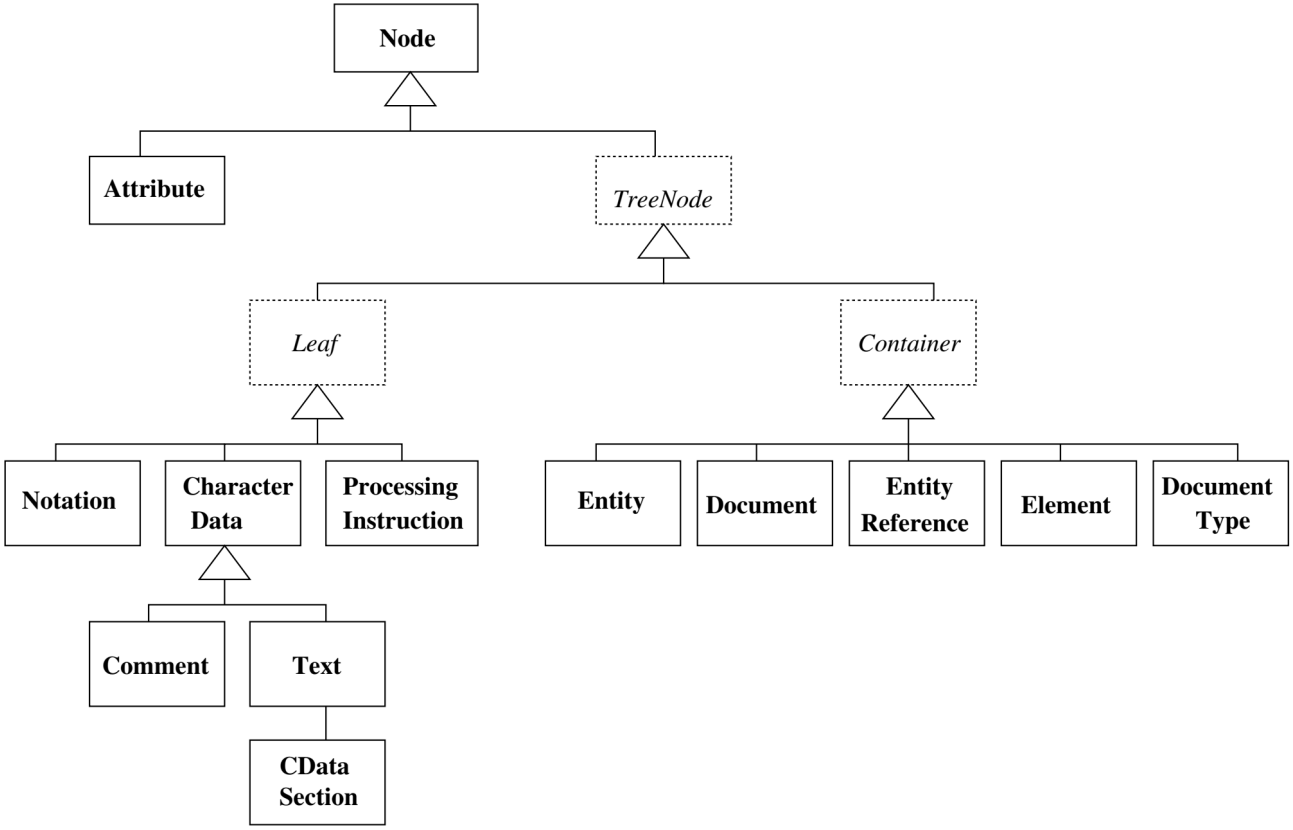
```
<restaurant>
  <nom>Bel Canto </nom>
  <adresse>
        Rue de la Tombe-Issoire
  </adresse>
  <telephone>
        01 45 46 47 48
  </telephone>
</restaurant>
```



Nœud DOM:

- **Type**
- Nom (optionnel)
- *Valeur* (optionnel)

Les Types DOM



Structure d'un document XML (syntaxe)

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 Duval</rue>
            <ville>Paris</ville>
      </adresse>
<!-- COMMENTAIRE  -->
</personne>
```

Déclaration XML (indications au processeur)

- xml version= ''1.0'' : décrit la versionXML 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** qui decrit la structure du contenu **(voir plus loin)**

Arbre d'éléments (syntaxe)

Element : <nom>contenu</nom>

Un document est formé d'une **hiérarchie** d'éléments dénotant la structure 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 être vide ou contenir d'autres éléments, des données, des références à des entités, des sections littérales, des instructions de traitement.

Quelques règles syntaxiques:

- Tout élément doit avoir une balise ouvrante et une balise fermante.
- 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

DTD : Définition de types de documents (Document Type Definition)

Une DTD est composée d'une suite de **déclarations** pour tous les composants utilisés dans un document XML :

- Éléments : <!ELEMENT …>
- Attributs d'éléments : <!ATTLIST …>
- Entités : <!ENTITY …>
- Notations : <!NOTATION … >

Elle peut aussi contenir des commentaires.

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

Déclaration d'Eléments

<!ELEMENT nom modèle_de_contenu>

Modèle de contenu :

- Expression régulières sur les noms des éléments fils et #PCDATA (text)

Expression régulières:

- Chaque nom d'élément est une expression régulière
- Si E1 et E2 sont des expressions régulières, alors
  - E1, E2 : concaténation
  - E1|E2 : choix
  - E1* : fermeture avec zéro ou plusieurs occurrences
  - E1+ : fermeture avec au moins une occurrence
  - E1? : option (0 ou 1 fois)

  sont des expression régulières.

Exemple

```
<!ELEMENT adresse (numero, rue, ville)>
<!ELEMENT coordonnées (portable|email)>
<!ELEMENT enfants (personnes)*>
<!ELEMENT texte-long (section+)>
<ELEMENT section (titre?, (sous-section)*)>
```

Déclaration d'Eléments (suite)

#PCDATA Parsed Character Data :
- Contenu texte
- Ex: <!ELEMENT titre (#PCDATA)>

```
<!ELEMENT paragraphe (#PCDATA | em | exp | ind)*>

<paragraphe>Du texte <em> en évidence </em>,
9 = 3<exp>2</exp></paragraphe>
```

Exemple

Modèle de contenu mixte :
- (#PCDATA | élément1 | élément2 | … | élémentn)*
- Pour mêler contenu textuel et éléments

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'>...</rapport>

Un attribut est défini

- dans la balise ouvrante de l'élément et
- doit toujours avoir un nom et une valeur, encadrée par des apostrophes simples ou doubles.

L'ordre des attributs n'est pas important.

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

Déclaration d'attributs

<!ATTLIST *nom-élément* nom-attribut1 type-attribut1 mode-défaut1
nom-attribut2 type-attribut2 mode-défaut2
…. >

- Type-attribut :
  - CDATA : la valeur de l'attribut est une chaîne de caractères
  - ID : identificateur d'élément, IDREF(S) : renvoi vers un (des) ID
  - NMTOKEN(S) : un ou des noms symboliques (sans blanc)
  - (a | b | c…) : type énumération de valeurs possibles
  - ENTITY(IES) : entités externes non XML
- Mode-défaut :
  - Valeur par défaut
  - Valeur constante : #FIXED
  - Présence obligatoire : #REQUIRED
  - Présence facultative : #IMPLIED

Exemples de déclarations d'attributs

DTD

```
<!ELEMENT date EMPTY >
<!ATTLIST date format (ANSI | ISO | FR) #OPTIONAL
              valeur #CDATA #REQUIRED>
```

DOCUMENT

```
<date valeur='9 octobre 2019'></date>
<date format='ISO' valeur='2019-10-9'/>
```

Exemples de déclarations d'attributs

DTD

```
<!ELEMENT ToDoList (task)*>
<!ELEMENT task (#PCDATA)>
<!ATTLIST task status (important|normal) "normal">
```

DOCUMENT

```
<ToDoList>
  <task status="important">
     This is an important task.
  </task>
  <task>
     This is by default a task that has a normal status.
  </task>
</ToDoList>
```

Exemples de déclarations d'attributs

DTD

```
<!ELEMENT description (#PCDATA)>
<!ATTLIST description xml:lang NMTOKEN #FIXED "en">
```

DOCUMENT

```
<description xml:lang="en">
   The following section of code displays the menu of
   user choices and gets the user's request. </description>
```

Attributs de type ID, IDREF(S)

Type ID :

- Permet d'identifier de façon unique un élément.
- L'unicité s'applique à tous les attributs de type ID du document.

Type IDREF :

- Permet de faire référence à un élément identifié par la valeur de l'attribut ID correspondant.
- La valeur d'un attribut de type IDREF doit correspondre à la valeur d'un attribut de type ID dans le document.

Type IDREFS :

- Permet de faire référence à plusieurs éléments identifiés par la valeur de l'attribut ID correspondant.
- Le séparateur est un espace (ex : att='p1 p2')

Exemple

```
<!ELEMENT DOC (AA)*>
<!ELEMENT AA (BB, CC)>
<!ELEMENT BB (#PCDATA)>
<!ELEMENT CC (#PCDATA)>
<!ATTLIST AA ident ID
#REQUIRED
        refs IDREFS #IMPLIED>
<!ATTLIST BB num ID
#REQUIRED>
<!ATTLIST CC ref IDREF
#IMPLIED>
```

```
<DOC>
    <AA ident='10'>
      <BB num='1'>abc</BB>
      <CC ref='13'>cde</CC>
    </AA>
      <AA ident='99'>
      <BB num='2'>xxx</BB>
      <CC ref='10'/>
    </AA>
    <AA ident='13' refs='1 99'>
      <BB num='3'>ttt</BB>
      <CC/>
    </AA>
    <AA ident='1'>
      <BB num='5'></BB>
      <CC ref=34/>
    </AA>
</DOC>
```

identifiants invalides (doubles)

référence invalide !

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 "&#x00A9; Editions UPMC."> ]>

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

Produira : © Editions UPMC.

Entités externes et notations

Entités externes : inclure d'autres documents (formats) XML

```
<!ENTITY carte-de-visite SYSTEM "cdv.xml">
<message> … &carte-de-visite; </message>
```

Notations : indiquer le traitement pour des entités externes

```
<!NOTATION gif SYSTEM "/usr/bin/xv">
<!ENTITY maphoto
        SYSTEM "./mesphotos/mapomme.gif" NDATA gif>
<photo img='maphoto'>
```

- « Appel » dans un attribut
- La déclaration de notation (NDATA) indique le traitement utilisée

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 "Éditions ToutSavoir" >

<!ENTITY rights "Tous droits réservés" >

<!ENTITY book "J.Martin. Le Web et les BD, %pub. &rights">

Dans le document :

 &book

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

Documents bien formés et documents valides

Un document XML est **bien formé** s'il respecte les règles syntaxiques définies par le standard XML :

- les éléments sont imbriqués (pas de chevauchement de balises)
- chaque attribut d'un élément a un nom unique,
- ....

Un document XML *bien formé* est **valide par rapport à une DTD** s'il respecte les déclarations de la DTD (éléments, entités, ...).

Exemple Document XML

DOCUMENTS XML

```
<?xml version="1.0" standalone="no" ?>
<!—existence et adresse d'une DTD externe -->
<!DOCTYPE article SYSTEM "cours.dtd">
<article>
    <titre>Cours de M1.</titre>
    <nom IDCours='4IN801'>MLBDA</nom>
    <prerequis>
    <nom IDCours='3IN009'>BD</nom>
    </prerequis>
    <controle>examens répartis</controle>
    <contenu>    ODMG. SQL3.XML.XPATH. </contenu>
</article>
```

COURS.DTD

```
<!ELEMENT article  (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**