www.mas-ict.ch

Hes·so
Haute Ecole Spécialisée de Suisse occidentale

# Module: XML et les bases de données

## Mapping XML- BDs

***Houda Chabbi Drissi***

houda.chabbi@hefr.ch

Source: XML and Databases de Ronald Bourret

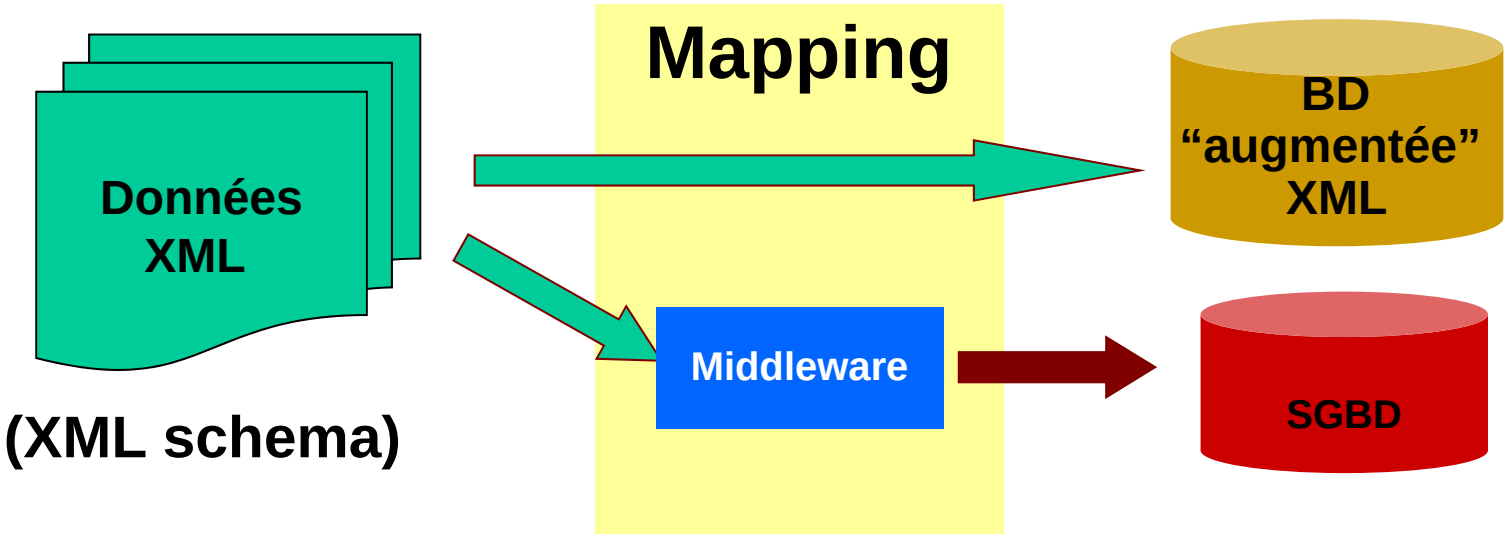
# Trois types de besoins XML-BDs

- Générer du XML
- Stocker/extraire des documents XML
- Stocker/extraire/requêter des documents XML

# Les solutions pour gérer du XML

- Solutions avec SGBDs existants:
  - Middleware (Mécanisme de mapping)
- Solutions avec SGBDs augmentés XML
  - $(SQL/XML)^{+}$ & Mécanisme de mapping
- Solutions avec XBDs

# Mécanisme de mapping




Mapper = transformer des documents XML en tables et de tables en documents XML.

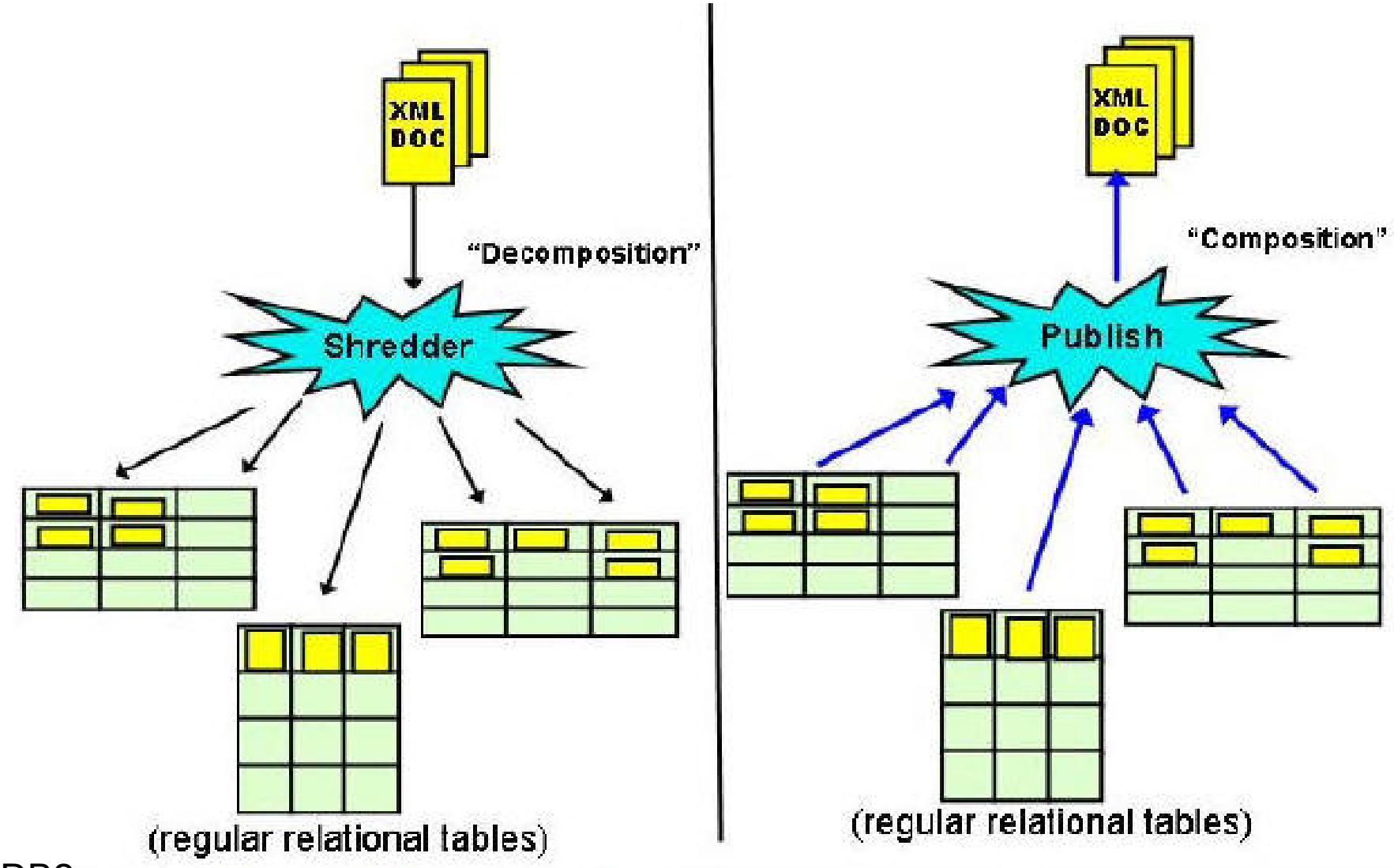
Permet l'utilisation d'un langage d'interrogation XML

# Le mapping: définition

Le mapping consiste en un stockage fragmenté *(shredding)* d'un document après ventilation de son contenu en différentes tables

**Nécessite un mapping entre les types XML et les types SQL**

# Le mapping: utilisation



# SGBD relationnelle étendu

- Utilise le ***mapping***
- Stocke en tant que type de données ***CLOB***

Certains combinent ces deux techniques ce qui accroît la souplesse du SGBD.

Nouveaux outils pour le stokage et le traitement de documents XML.

Importer du XML (dans tables) / exporter du XML

# Mapper XML dans le schéma de la base

- Le mapping concerne (cf. suite)
  - Les éléments, attributs et textes
  - Ignore la structure physique (CDATA sections, encoding,…)
  - Ignore les structures logiques (processing instructions, comments, order,…)

**Un ”import et un export” ne crée pas le même fichier (mais les mêmes données)**

# Approches du mapping

Deux approches:

- Table-Based mapping
- Object-relational mapping

# Remarques sur le mapping

- Mapping des noms: aucune contrainte pour conserver exactement les même chaines de caractères, au contraire:
  - Si un attribut et un même sous-éléments ont le même nom et qu'il doivent être représenté par deux attributs…
- Mapping data types
- Bidirectionel

# Approches du mapping

Deux approches:

- Table-Based mapping
- Object-relational mapping

# Table-Based mapping

- Mapping générique :
  - Deux tables :
    - ✓une table pour stocker l'ordre, les balises (label) et la relation parent/enfant,
    - ✓une table pour les valeurs.
  - Plusieurs tables
- Mapping guidé par la DTD / XSD

# Mapping générique avec 2 tables

Relation enfant-parent

| IdParent | Position | Label | Type | id |
|---|---|---|---|---|
| | | | | |

Relation valeur

| Id | Valeur |
|---|---|
| | |

→ Dénormalisé: Valeur peut prendre +/- de place

# Exemple : document XML

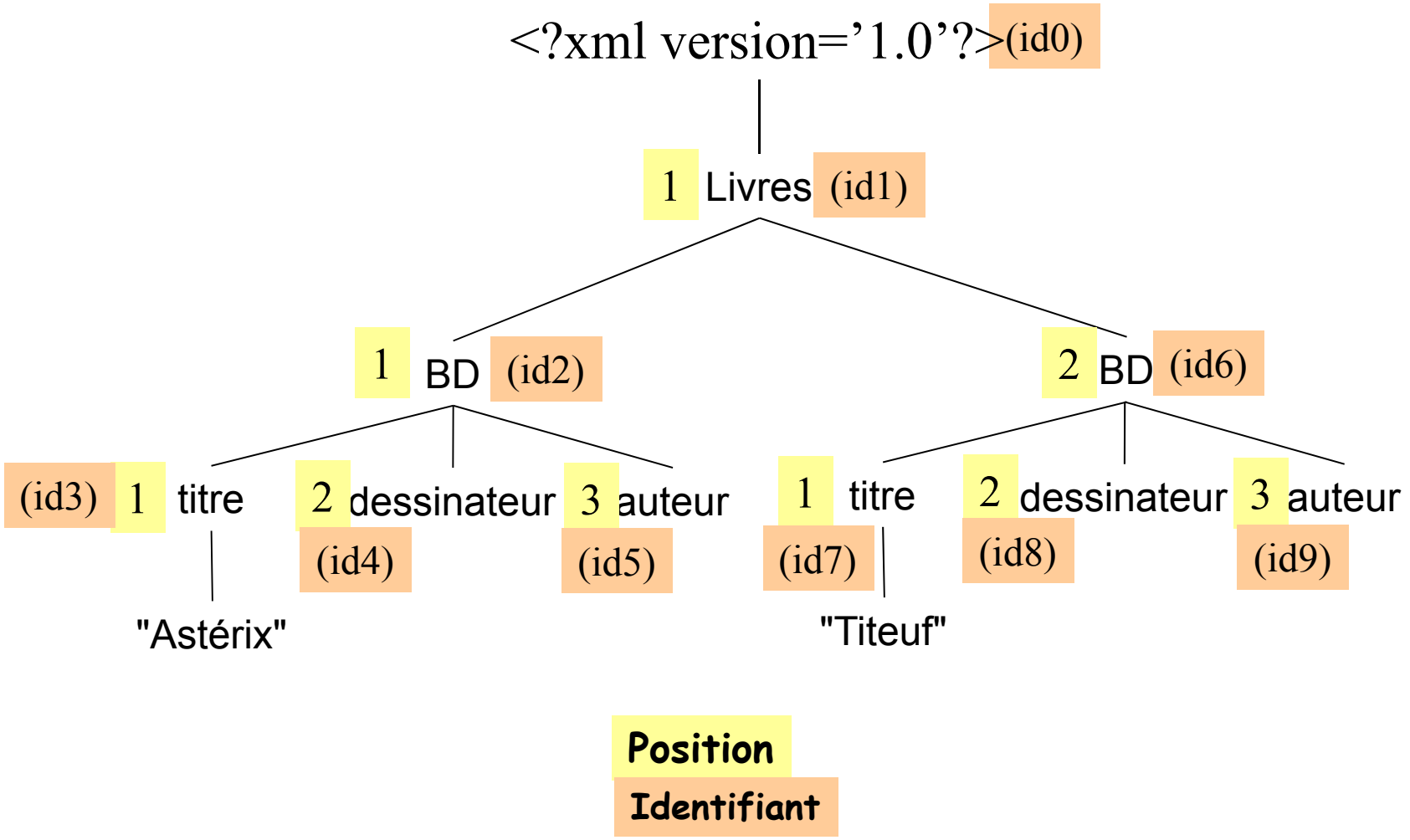
```
<?xml version='1.0'?>                                   (id0)
1 <Livres>                                              (id1)
  1 <BD>                                                (id2)
      1 <titre>Asterix</titre>                          (id3)
      2 <dessinateur>Uderzo</dessinateur>               (id4)
      3 <auteur>Gossigny</auteur>                       (id5)
  </BD>
  2 <BD>                                                (id6)
      <titre>Titeuf</titre>                             (id7)
      <dessinateur>Zep</dessinateur>                    (id8)
      <auteur>Zep</auteur>                              (id9)
  </BD>
</Livres>
```

**Position**

**Identifiant**

Relation enfant-parent

| IdParent | Position | Label | Type | id |
|---|---|---|---|---|
| | | | | |


<?xml version='1.0'?>(id0)
1 Livres (id1)
1 BD (id2)
2 BD (id6)
(id3) 1 titre
2 dessinateur
3 auteur
(id4)
(id5)
1 titre
2 dessinateur
3 auteur
(id7)
(id8)
(id9)
"Astérix"
"Titeuf"
Position
Identifiant

T1

| IdParent | Position | Label | Type | id |
|---|---|---|---|---|
| id0 | 1 | livre | ref | id1 |
| id1 | 1 | BD | ref | id2 |
| id1 | 2 | BD | ref | id6 |
| id2 | 1 | titre | cdata | id3 |
| id2 | 2 | dessinateur | cdata | id4 |
| id2 | 3 | auteur | cdata | id5 |

T2

| id | Valeur |
|---|---|
| id3 | Asterix |
| id5 | Gossigny |

*Ref: {element,attribut,id, idref}

# Requêtes

- XQuery : Les titres des BDs de Gossini:

```
For $l in document('Livres.xml')/livres/BD,
Where $l/auteur = 'Gossigny'
Return $l/titre
```

- SQL : 4 sélections et 4 jointures :

```
select Vtit2.val
from T1 Liv, T1 Aut, T1 Tit, T2 Vaut1, T2 Vtit2
where Liv.label = 'BD' and Aut.label = 'auteur'
        and Tit.label = 'titre'
        and Liv.id = Aut.par and Liv.id = Tit.par
        and Aut.id = Vaut1.id and Tit.id = Vtit2.id
        and Vaut1.val = 'Gossigny';
```

# Query: Les titres des BDs de Gossini

T1

| IdParent | Position | Label | Type | id |
|---|---|---|---|---|
| id0 | 1 | livre | ref | id1 |
| id1 | 1 | BD | ref | id2 |
| id1 | 2 | BD | ref | id6 |
| **id2** | **1** | **titre** | **cdata** | **id3** |
| id2 | 2 | dessinateur | cdata | id4 |
| id2 | 3 | auteur | cdata | id5 |

T2

| id | Valeur |
|---|---|
| id3 | Asterix |
| id5 | Gossigny |

```
select Vtit2.val
from T1 Liv, T1 Aut, T1 Tit, T2 Vaut1, T2 Vtit2
where and Aut.label = 'auteur' and Aut.id = Vaut1.id
      and Vaut1.val = 'Gossiny' and Liv.label = 'BD'
      and Liv.id = Aut.par and Liv.id = Tit.par
      and Tit.label = 'titre' and Tit.id = Vtit2.id
```

# Synthèse

- Avantages :
  - ☺ Généricité: Fonctionne sur tous les documents
  - ☺ +/- Normalisé: Espace utilisé est faible
- Inconvénients :
  - ☹ Le nombre de jointures

# Table-Based mapping

- Mapping générique :
  - Deux tables :
    - ✓une table pour stocker l'ordre, les balises et la relation parent/enfant,
    - ✓une table pour les valeurs.
  - Plusieurs tables: 2 approches
- Mapping guidé par la DTD / schema

# 1ere approche: exemple

Un document XML est «vu» comme suit:

```
<A>
    <B>
        <C>ccc</C>
        <D>ddd</D>
        <E>eee</E>
    </B>
    <B>
        <C>fff</C>
        <D>ggg</D>
        <E>hhh</E>
    </B>
</A>
```

Table B

| C | D | E |
|---|---|---|
| ccc | ddd | eee |
| fff | ggg | hhh |

# 1ere approche

Le document XML est «vu» comme suit:

```
<Tables>
 <table_1>
   <Row>
       <Column_1>...</Column_1>
       ...
       <Column_n>...</Column_n>
   </Row>
    ...
   <Row>
       <Column_1>...</Column_1>
        ...
       <Column_n>...</Column_n>
   </Row>
 </table_1>
 <table_p>
    ...
 </table_p>
</Tables>
```

Le document est « vu » comme:

- une table unique
- ou un ensemble de tables.

# Synthèse

☺ Approche très simple à mettre en œuvre

☹Approche trop simpliste: Les documents ciblés forment un sous ensemble très petits: que faire des entités? Commentaires? Déclaration de DTD, les PI?

☹Ne préserve pas la structure originale du XML

Assez répandu! (XSU oracle)
Approche middleware

# 2ieme approche

**Principe:** Une table pour chaque type de chemin:

- Une table par niveau dans l'arbre

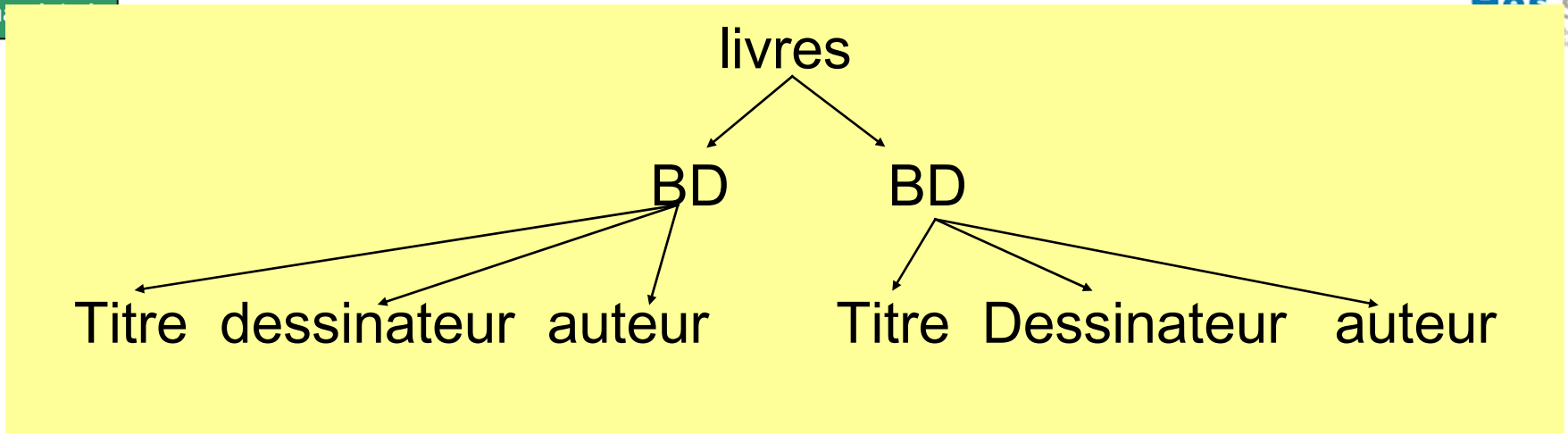
- Informations stockées :
  - père/fils
  - nœud/valeur
  - nœud/attribut
  - nœud/position

# Exemple : document XML

```
<?xml version='1.0'?>                                   (id0)
1 <Livres>                                              (id1)
  1 <BD>                                                (id2)
      1 <titre>Asterix</titre>                          (id3)
      2 <dessinateur>Uderzo</dessinateur>               (id4)
      3 <auteur>Gossigny</auteur>                       (id5)
    </BD>
  2 <BD>                                                (id6)
      <titre>Titeuf</titre>                             (id7)
      <dessinateur>Zep</dessinateur>                    (id8)
      <auteur>Zep</auteur>                              (id9)
    </BD>
</Livres>
```

**Position**

**Identifiant**




Livres.BD

| Parent | Id |
|---|---|
| id1 | id2 |
| id1 | id6 |

Livres.BD.titre

| Parent | Id |
|---|---|
| id2 | id3 |
| id6 | id7 |

Livre.BD.titre.val

| Id | Val |
|---|---|
| id3 | Asterix |
| id7 | …. |

…

# Requête

- XQuery : Les titres des BDs de Gossigny:

```
For $l in document('Livres.xml')/livres/BD
Where $l/auteur = 'Gossigny'
Return $l/titre
```

- SQL : 1 sélection et 3 jointures

```
Select V2.val
From livres.BD.titre A, livres.BD.auteur B,
livres.BD.auteur.val V1,livres.BD.titre.val V2
Where A.Par = B.par and
          A.Id = V1.Id and B.Id = V2.Id and
          V2.val = 'Gossigny';
```

# Synthèse

- Avantages :
  - ☺ Requêtes avec expressions de chemins
  - ☺ Petites relations
  - ☺ Classification des nœuds
- Inconvénients :
  - ☹ Le nombre de jointures

# Table-Based mapping

- Mapping générique :
  - Deux tables :
    - ✓une table pour stocker l'ordre, les balises et la relation parent/enfant,
    - ✓une table pour les valeurs.
  - Plusieurs tables:
    - ✓Une table pour chaque type de chemin

➢Mapping guidé par la DTD / schema

# Stockage guidé par la DTD/XSD

- Utiliser la DTD/XSD pour créer le schéma de la base.
- Décider quand un élément est “mis” dans la table de son parent et quand il faut créer une table séparée.
- Types d’éléments peuvent être partagés: redondance
- Approche: utilise l’Object-relational mapping (suite) et transforme en relationnel le résultat.

Les implémentations dans les SGBDR combinent les 2 étapes en 1

# Approches du mapping

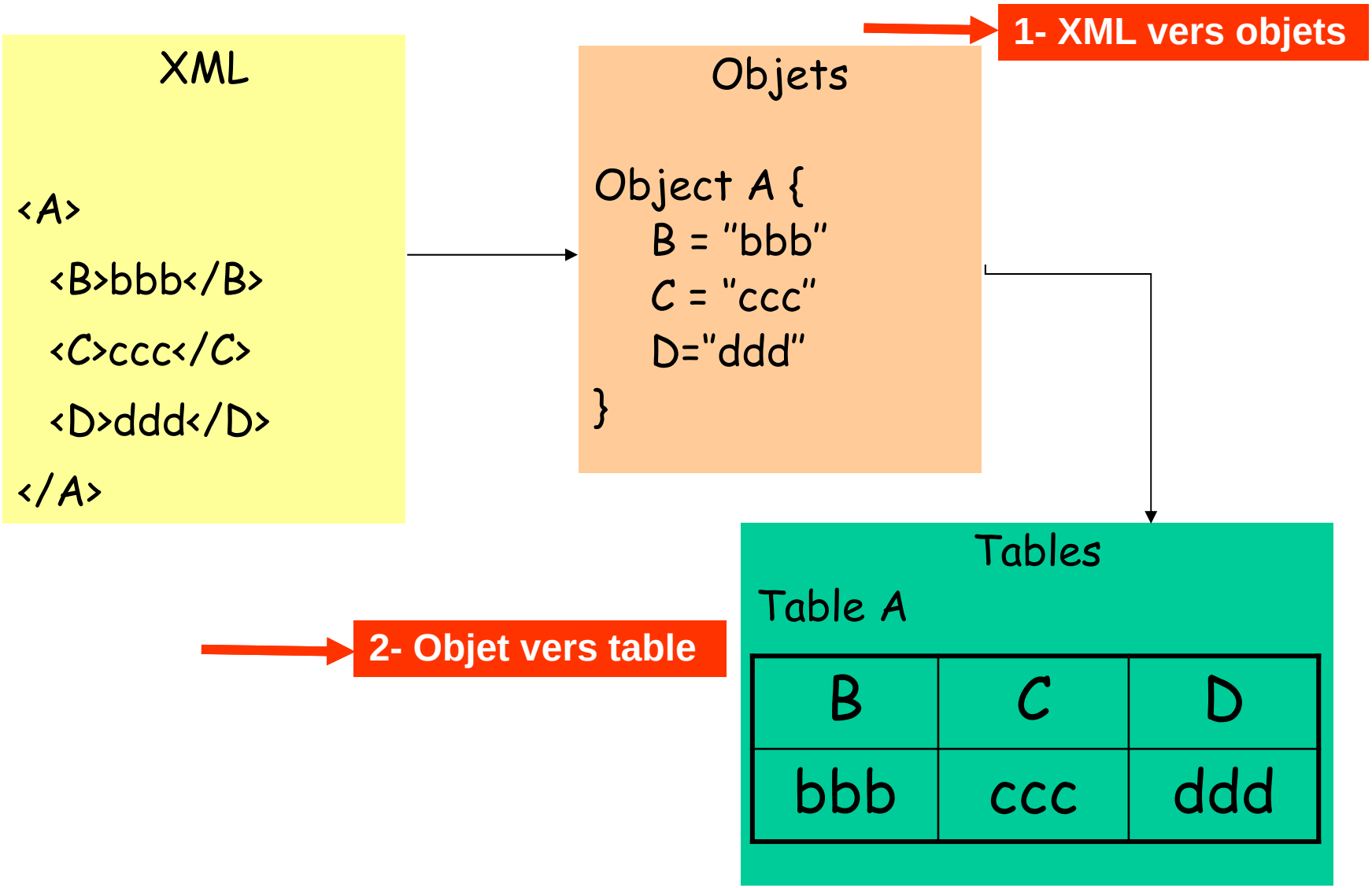
Deux approches:

- Table-Based mapping

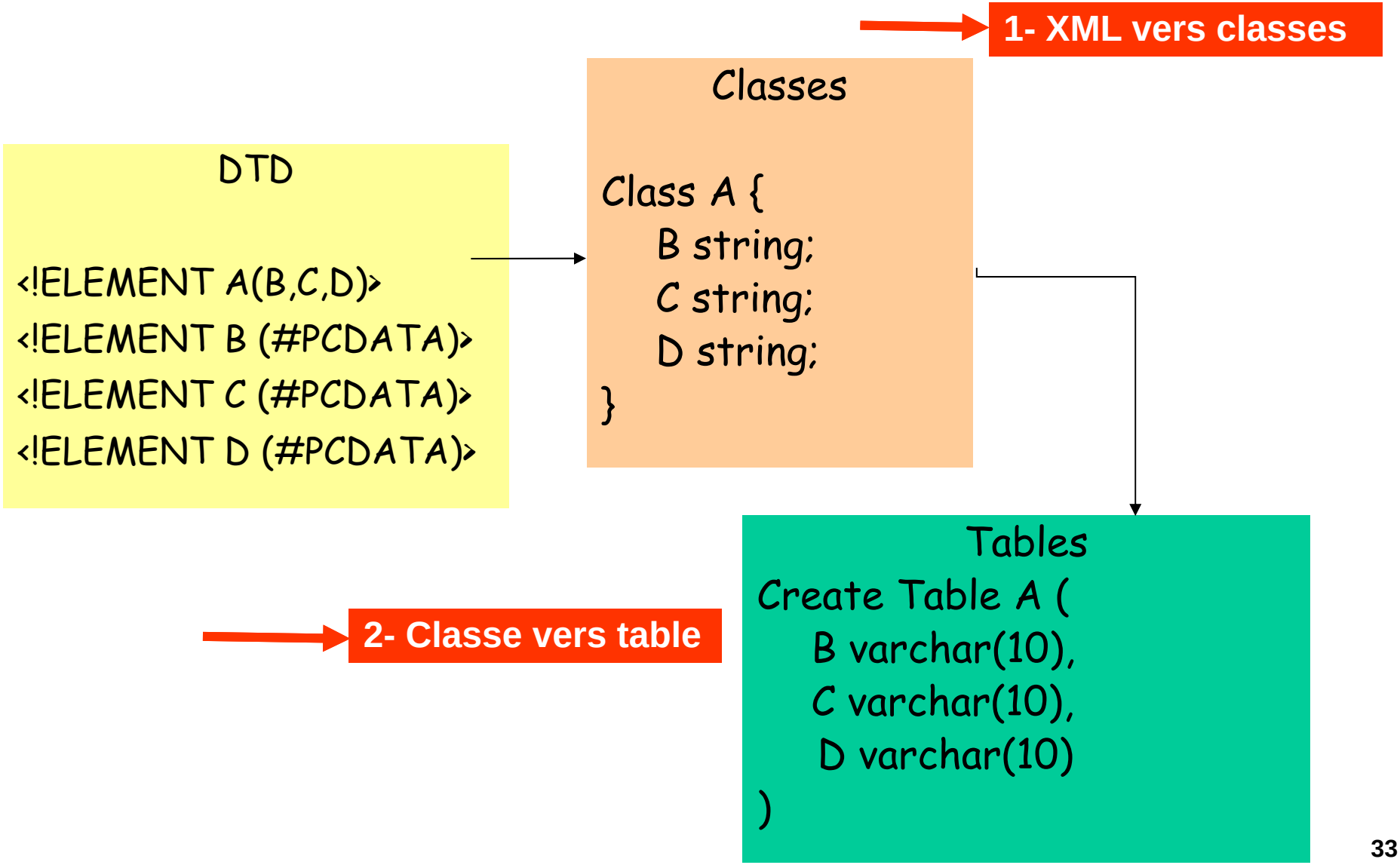
➢ Object-relational mapping

*

# Un exemple simple (données)



# Un exemple simple (modèle)



# Mapping de séquences et hiérarchies

**1- XML vers classes**

DTD

```
<!ELEMENT A(B,C)>
<!ELEMENT B (#PCDATA)>
<!ATTLIST A
        F CDATA #REQUIRED)>
<!ELEMENT C(D,E)>
<!ELEMENT D (#PCDATA)>
<!ELEMENT E (#PCDATA)>
```



Classes

```
Class A {
    B string;
    c       C;//type objet
    F_att string;
}
Class C {
    D string;
    E string;
}
```

## 2- Classe vers table

**Classes**

```
Class A {
    B string;
    c       C; //type objet
    F_att string;
}
Class C {
    D string;
    E string;
}
```

**Tables**

```
Create Table A (
    B       varchar(10),
    c_fk   FK(C(c_pk)),
    F_att varchar(10)
)
Create Table C (
    c_pk int;
    D varchar(10),
    E varchar(10),
)
```

→ Utilisation des paradigmes PK et FK

# Mapping du facultatif ?

**1- XML vers classes**

DTD

```
<!ELEMENT A(A?,B,C)>
<!ELEMENT B (#PCDATA)>
<!ELEMENT C (#PCDATA)>
```

Classes

```
Class A {
    a  A; //nullable
    B String;
    C string;
}
```

**2- Classe vers table**

Tables

```
Create Table A (
    a_pk int;
    B varchar(10),
    C varchar(10),
    a_fk FK(A(a_pk)) NULL,
)
```

→ Utilisation du NULL

# Mapping du choix

**1- XML vers classes**

DTD

```
<!ELEMENT A(B | C)>
<!ELEMENT B (#PCDATA)>
<!ELEMENT C(D,E)>
<!ELEMENT D (#PCDATA)>
<!ELEMENT E (#PCDATA)>
```

Classes

```
Class A {
    B string; //nullable
    c       C;//nullable
}
Class C {
    D string;
    E string;
}
```

→ Utilisation du NULL

## 2- Classe vers table

**Classes**

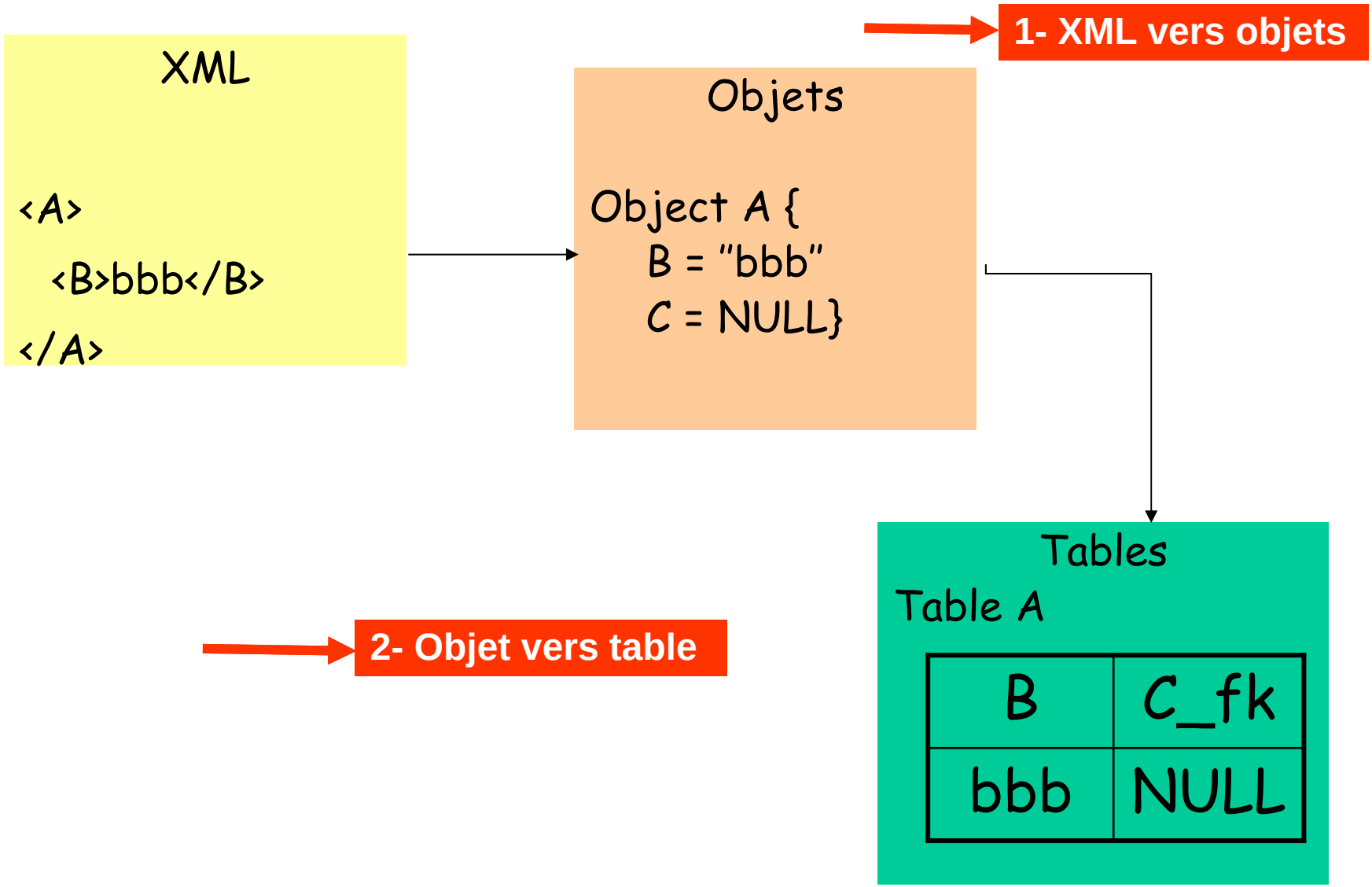
```
Class A {
    B string; //nullable
    c       C;//nullable
}
Class C {
    D string;
    E string;
}
```

**Tables**

```
Create Table A (
    B       varchar(10) NULL,
    c_fk   FK(C(c_pk)) NULL,
)
Create Table C (
    c_pk int   NOT NULL;
    D varchar(10) NOT NULL,
    E varchar(10) NOT NULL,
)
```

→ Utilisation des paradigmes PK et FK

# Exemple de données



# Mapping de la répétition *

**1- XML vers objets**

DTD

```
<!ELEMENT A(B, C*)>
<!ELEMENT B (#PCDATA)>
<!ELEMENT C (#PCDATA)>
```

Classes

```
Class A {
    B   string;
    C    string[];//collection NULL
}
```

Tables

```
Create Table A (
    B       varchar(10),
    a_pk  int,
)
Create Table C (
    a_fk int  FK(A(a_pk));
     C varchar(10)
)
```

**2- Objet vers table**

→ Relation 0-N

# Mapping de la répétition +

**1- XML vers objets**

DTD

```
<!ELEMENT A(B, C+)>
<!ELEMENT B (#PCDATA)>
<!ELEMENT C (#PCDATA)>
```

Classes

```
Class A {
    B   string;
    C1  string;
    C    string[];//  NULL
}
```

**2- Objet vers table**

Tables

```
Create Table A (
    B       varchar(10),
    C       varchar(10),
    a_pk  int,
)
Create Table C (
    a_fk int  FK(A(a_pk));
    C varchar(10)
)
```

→ Relation 1-N

# Mapping du mixed content

DTD

```
<!ELEMENT
  A(#PCDATA|B|C)*>
<!ELEMENT B (#PCDATA)>
<!ELEMENT C (#PCDATA)>
```

Classes

```
Class A {
    pcdata  string[];
    B       string[];
    C       string[];
}
```

Tables

```
Create Table A (
    a_pk  int,
)
Create Table PCDATA (
    a_fk int  FK(A(a_pk));
    pcdata   varchar(10)
)
```

```
Create Table B (
    a_fk int  FK(A(a_pk));
    b   varchar(10)
)
Create Table C (
    a_fk int  FK(A(a_pk));
    c   varchar(10)
)
```

# Exemple de données

## XML

```
<A>voila du txt<B>b</B>
  pour <C>c</C> voir ce
  qui se passe <C>cc</C>
  <B>bb</B>
</A>
```

## Objets

```
Object A {
    pcdata = {"voila du txt","pour",
    "voir ce qui se passe"}
    B = {"b", "bb"}
    C = {"c", "cc"}
```

## Tables

Table A (1):

| A_fk | B |
|---|---|
| 1 | b |
| 1 | bb |

| A_fk | C |
|---|---|
| 1 | c |
| 1 | cc |

| A_fk | pcdata |
|---|---|
| 1 | Voila du txt |
| 1 | pour |
| 1 | voir ce qui se passe |

# Remarques

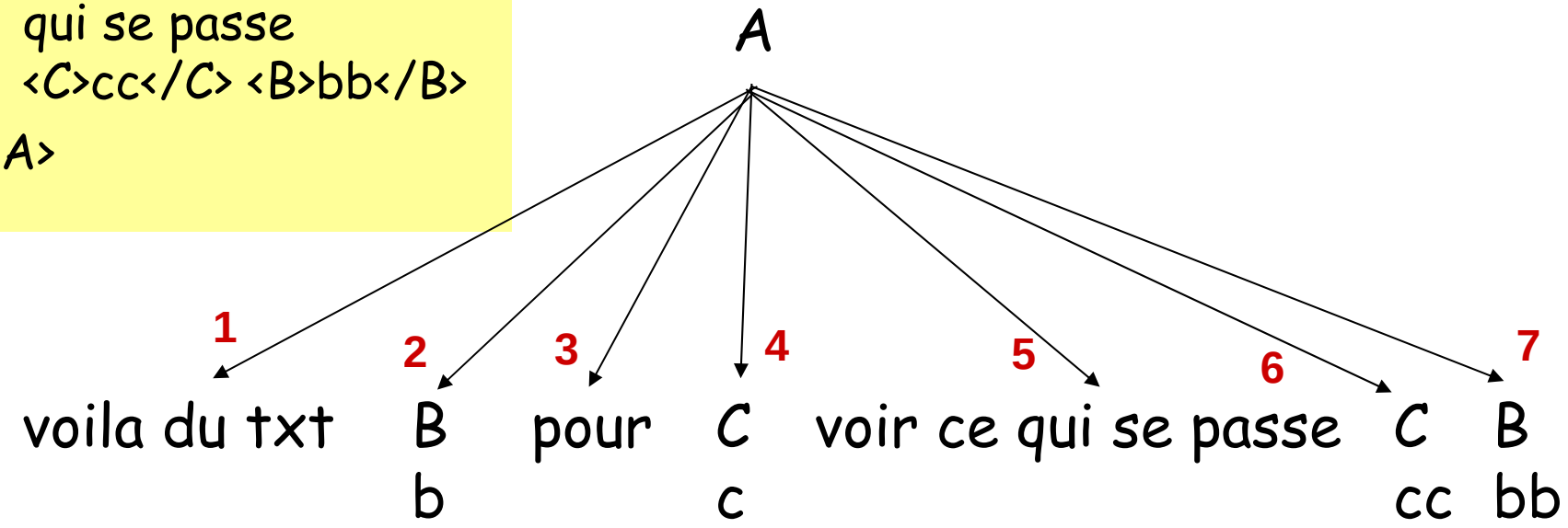
- Du « mixed data » est plutôt du « document centric data » sa transformation tel que montrée est peu efficace.

- Reste une question primordiale: « et l'ordre dans tout ça? »

# Ordre: sibbling - hiérarchie

XML

```
<A>voila du txt<B>b</B>
    pour <C>c</C> voir ce
    qui se passe
    <C>cc</C> <B>bb</B>
</A>
```



# Prise en compte de l'ordre

- Attention pour des « data centric » l'ordre n'est pas important ☺
- Le mapping actuel supporte la relation de la hiérarchie (FK, PK) ☺
- Sinon, ajout de l'information de l'ordre

# Mapping du mixed content + ordre

DTD

```
<!ELEMENT A(#PCDATA|B|C)*>
<!ELEMENT B (#PCDATA)>
<!ELEMENT C (#PCDATA)>
```

Tables

```
Create Table A (
    a_pk  int,
)
Create Table PCDATA (
    a_fk int  FK(A(a_pk)),
    pcdataordre int;
    pcdata   varchar(10)
)
Create Table B (
    a_fk int  FK(A(a_pk)),
    b   varchar(10),
    bordre int
)
Create Table C (
    a_fk int  FK(A(a_pk)),
    c   varchar(10),
    cordre int
)
```

# Exemple de données

## XML

```
<A>voila du txt<B>b</B>
   pour <C>c</C> voir ce
   qui se passe <C>cc</C>
   <B>bb</B>
</A>
```

## Objets

Object A {
pcdata = {"voila du txt","pour", "voir ce qui se passe"}
pcdataordre = {1,3,5}
B = {"b", "bb"}, Bordre = {2,7}
C = {"c", "cc"},Cordre = {4,6}

## Tables

Table A (1);

| A_fk | B | Bordre |
|---|---|---|
| 1 | b | 2 |
| 1 | bb | 7 |

| A_fk | C | Cordre |
|---|---|---|
| 1 | c | 4 |
| 1 | cc | 6 |

| A_fk | pcdata | ordre |
|---|---|---|
| 1 | Voila du txt | 1 |
| 1 | pour | 3 |
| 1 | voir ce qui se passe | 5 |

# Exemple-exercice

```
<!ELEMENT BD (titre, auteur*,dessinateur*, editeur?) >
<!ELEMENT caricature (titre, dessinateur*) >
<!ELEMENT titre #PCDATA >
<!ELEMENT auteur #PCDATA >
<!ELEMENT dessinateur #PCDATA >
<!ELEMENT editeur #PCDATA >
```

Schéma de la base?

<!ELEMENT BD (titre, auteur*,dessinateur*, editeur?) >

<!ELEMENT caricature (titre, dessinateur*) >

```
BD(id int,
     parId int FK(parent(id)),
     titre varchar(100),
     editeur varchar(100) NULL)
BD_auteur(id int,
               parId int FK(BD(id)),
               auteur varchar(100))
BD_dessinateur(id int,
               parId int FK(BD(id)),
               dessinateur varchar(100))
```

Caricature(id, parId, titre)

Caricature_dessinateur(id, parId, dessinateur)

# Une autre variante possible

<!ELEMENT BD (titre, auteur*,dessinateur*, editeur?) >

<!ELEMENT caricature (titre, dessinateur*) >

BD(id int, parId, titre, editeur)

BD_auteur(id, parId, auteur)

BD_dessinateur(id, parId, dessinateur)

Caricature(id, parId, titre)

Caricature_dessinateur(id, parId, dessinateur)

titre(id,parid,titre)

editeur(id, parId, editeur)

auteur(id, parId, auteur)

dessinateur(id, parId, dessinateur)

Apparition de tables en plus (jointure en plus) mais valide des éléments def en dehors de la racine

# Requêtes (1)

Sur 1ere version

- XQuery : BD de Zep :

```
for $l in document("biblio.xml")/biblio/livre
fhere $l/auteur = 'Zep'
feturn $l/titre
```

- SQL : 1 jointure et 1 sélection

```
select titre
from BD, BD_auteur
where BD.id = BD_auteur.parId
and auteur = ''Zep''
```

# Requêtes (2)

Sur 2ieme version

- XQuery : Noms des dessinateurs :

```
For $a in $Livres//dessinateur
Return $a
```

- SQL :

```
select dessinateur from dessinateur
union
select dessinateur from BD_dessinateur
union
select dessinateur from caricature_dessinateur;
```

# Manque

- Entité
- Notations
- Attributs et leur différents types

- Mapping du relationnel vers une DTD
- Mapping XSD vers relationnel

# Synthèse

- Avantages :
  - ☺ Requêtes avec expressions de chemins
- Inconvénients :
  - ☹ Non normalisé: problèmes de redondance et de maintenance
  - ☹ Trop de tables

Souvent des variantes (hybrides) mises en place par:

- les solutions « middleware »
- la majeur partie des « SGBDs augmentés XML

# SGBD objet-relationnelle étendu

Pour stocker du XML dans un SGBDRO, les données du document sont modélisées en tant que type de données ***natif XML*** ou **mapper dans une table OR:**

- Les types d'éléments ayant des attributs, les contenus d'éléments, ainsi que les contenus mixtes (les *types d'éléments complexes)* sont généralement modélisés comme des **objets**
- Les types d'éléments contenant seulement des PCDATA (les *types d'éléments simples)*, les attributs et les PCDATA elles-mêmes sont modélisés comme des **attributs**

# Exemple de mapping

```
<Livres>
  <BD>
    <titre>Asterix</titre>
    <dessin>Uderzo</dessin>
    <auteur>Gossigny</auteur>
  </BD>
  <BD>
    <titre>Titeuf</titre>
    <dessin>Zep</dessin>
    <auteur>Zep</auteur>
  </BD>
</Livres>
```

```
create type BDType
as OBJECT (
titre VARCHAR2(32)
dessin VARCHAR2(32)
Auteur Varchar2(32)
);
create table livres (
  id int;
   BD BDType
);
```

# Synthèse

- Avantages :
  - ☺ Requêtes avec expressions de chemins
  - ☺ Petites relations
  - ☺ Classification des nœuds
- Inconvénients :
  - ☹ Le nombre de jointure

# Implémentation

- SS2012:
  - Le mapping plutôt de type « générique table-based » via la création du XML primary key

- Ora12c:
  - Le mapping plutôt «  Object-relational mapping  » diriger par une XSD.