Bases de données et XML

Quelques questions

- XML, format de bases de données ?
  - Pas vraiment : stockage inefficace, pas très structuré, pas (encore) de langage de requête
- Doit-on associer une BD à XML ?
  - Oui, pour l'indexation, les transactions, les **mises à jour**, etc

=> encore une fois, XML est plutôt un format d'échange et d'intégration

Données et documents

- Dans une base, il y a des données :
  - Très structurées
  - Granularité fine
  - Typées
- Et dans un document XML ?
  - Pas de typage
  - Beaucoup de texte
  - Une structure très souple

Situations mixtes

Systèmes d'information où la séparation données/document n'est pas très nette.

- Un part de description textuelle est de nature « document »

  Exemple : tout le texte présentant un cinéma, ses activités, etc

- Une part est contenue dans une base de données

  Exemple : les films

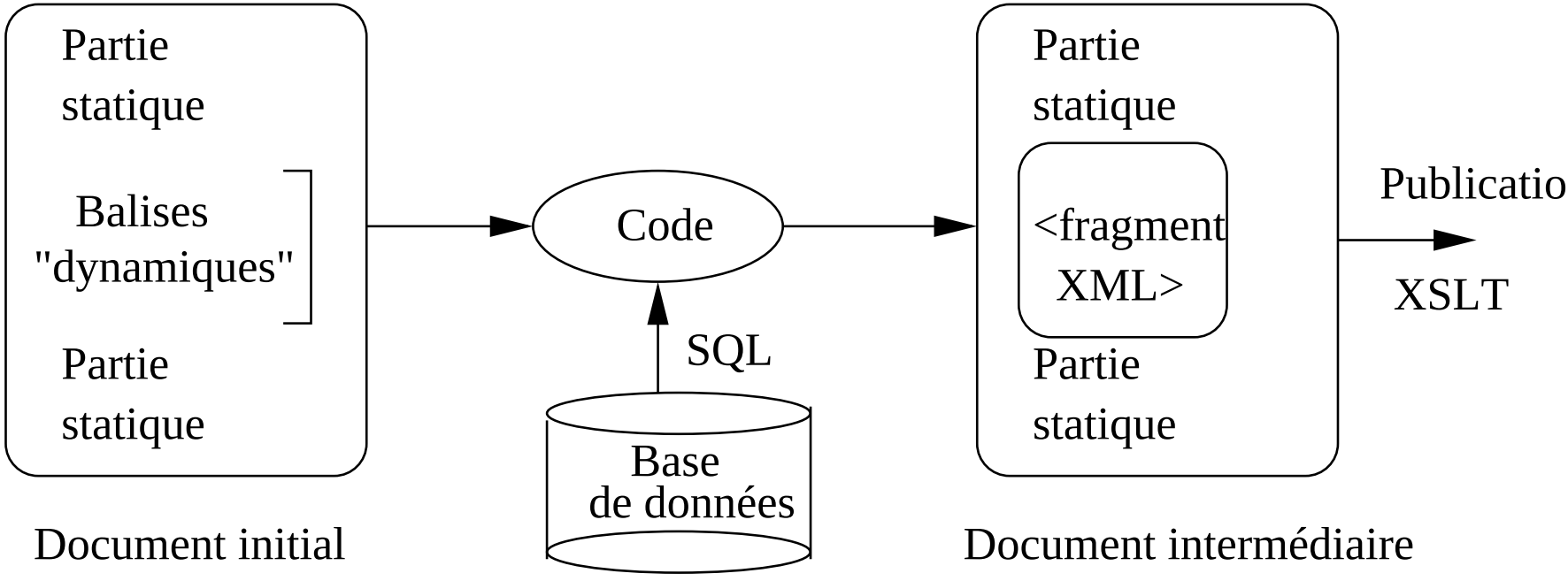
XML dynamique

- XML : moyen de rendre publique (= *publier*) tout ou partie d'une BD.
- Besoin : intégrer du XML statique, et des parties de code effectuant du SQL
- Mieux : comment éviter le mélange programmation/gestion du contenu (*séparation des points de vue*)

=> même problème que HTML + (PHP, ASP, JSP)...

Architecture générale

Déclenchement du code par « balises dynamiques »



Première étape : transformation BD relationnelle vers XML

Notre base de données

Des stations où séjournent des clients.

- Station (**nomStation**, capacité, lieu, région, tarif)
- Activité (***nomStation***, **libellé**, prix)
- Client (**id**, nom, prénom, ville, région, solde)
- Séjour (***idClient***, **station**, **début**, nbPlaces)

$\Rightarrow$ un schéma avec clés primaires et clés étrangères

Autres aspects d'un schéma relationnel

On peut décrire un ensemble de **contraintes** :

- des contraintes de types : certains attributs sont numériques, d'autres alphanumériques, avec ou sans décimale, etc ;
- des contraintes d'existence : certains attributs ont toujours une valeur ;
- des contraintes d'appartenance à un ensemble énuméré : exemple une codification des régions (« Antilles », « Europe », « Amérique », « Asie », etc).

Une instance

nomStation	capacité	lieu	région	tarif
Venusa	350	Guadeloupe	Antilles	1200
Passac	400	Alpes	Europe	1000

La table *Station*

nomStation	libellé	prix
Venusa	Voile	150
Venusa	Plongée	
Passac	Ski	200
Passac	Piscine	20

La table *Activité*

Une instance

id	nom	prénom	ville	région	solde
10	Fogg	Phileas	Londres	Europe	12465
30	Kerouac	Jack	New York	Amérique	9812

idClient	station	début	nbPlaces
30	Santalba	2001-08-14	5
30	Passac	2001-08-15	3
30	Venusa	2001-08-03	3
30	Farniente	2002-06-24	5
10	Farniente	2002-09-05	3

Table = ensemble

Une table est un **ensemble** de lignes : l'ordre n'est pas important.

```
SELECT nomStation, lieu, région
FROM Station
WHERE capacité > 100
```

Le résultat de cette requête est indépendant de l'ordre.

Résumé : le modèle relationnel

- la base est décrite par un *schéma* séparé (physiquement) des données ;
- les caractéristiques d'une table sont :
  - son nom ;
  - le nom de ses attributs (ou colonnes) ;
  - la *clé primaire* ;
  - une ou plusieurs *clés étrangères*
  - le *type* de chaque attribut
  - les contraintes (`NOT NULL`, ensemble énuméré)

Réprésentation en XML : avec éléments

Trois niveaux d'éléments dans l'arbre : table, lignes, attributs

```
<?xml version='1.0' encoding='ISO-8859-1?>
<Stations>
  <Station>
    <nomStation>Venusa</nomStation>
    <capacite>350</capacite>
    <lieu>Guadeloupe</lieu>
    <region>Antilles</region>
    <tarif>1200.00</tarif>
  </Station>
</Stations>
```

Tous représentés par des éléments

Réprésentation en XML : avec attributs

Deux niveaux d'éléments tables et lignes :

```
<?xml version='1.0' encoding='ISO-8859-1?>
<Stations>
<Station nomStation='Venusa'
    capacite='350'
    lieu='Guadeloupe'
    region='Antilles'
    tarif='1200.00'
/>
</Stations>
```

Attributs relationnels $\equiv$ attributs XML

Petite discussion

La représentation avec attributs est plus proche du relationnel car :

- Absence d'ordre sur les attributs en XML comme en relationnel.
- On ne peut pas avoir deux fois un attribut avec le même nom.
- On peut, avec une DTD, donner la liste des valeurs acceptées pour un attribut.

De plus, moins volumineux

Représentation des associations

L'équivalent du relationnel en XML

```
<?xml version='1.0' encoding='ISO-8859-1?>
<Stations>
  <Station nomStation='Venusa'
     capacite='350'
     lieu='Guadeloupe'
     region='Antilles'
     tarif='1200.00' />
  <Activite nomStation='Venusa'
     libelle='Voile'
     prix='150.00' />
   <Activite nomStation='Venusa'
     libelle='Plongee' />
</Stations>
```

Pourquoi prendre cette représentation

En relationnel :

- tout est représenté « à plat »
- on associe les lignes par le système clé primaire/clé étrangère
- on fait le rapprochement par un calcul (la jointure) assez coûteux

Intérêt : simple, raisonnablement efficace, évite les redondances

Et en XML ?

Le modèle est plus puissant

- même représentation possible
- toujours aussi coûteux
- **pas nécessaire de garder la représentation « à plat »**

On peut exploiter une représentation imbriquée quand c’est judicieux.

Représentation imbriquée

On exploite le fait qu'une activité est relative à une seule station

```
<?xml version='1.0' encoding='ISO-8859-1?>
<Stations>
 <Station nomStation='Venusa'
  capacite='350'
  lieu='Guadeloupe'
  region='Antilles'
  tarif='1200.00'>
  <Activite libelle='Voile' prix='150.00'/>
  <Activite libelle='Plongee'/>
 </Station>
</Stations>
```

Associations plusieurs à plusieurs

Beaucoup moins évident à gérer. Exemple avec l'association *Client/Station*

- Un client peut aller dans plusieurs stations
- Un station peut accueillir plusieurs clients

En relationnel, on crée une table intermédiaire (*Les séjours*)

`Séjour (`***`idClient`***`, `**`station`**`, `**`début`**`, nbPlaces)`

Exemple

```
<?xml version='1.0' encoding='ISO-8859-1?>
<Station nomStation='Venusa'
    region='Antilles' tarif='1200.00'>
  <Sejour idClient='30' debut='2001-08-03'
     nbPlaces='3'/>
</Station>
<Station nomStation='Farniente'
    region='Océan Indien' tarif='1500.00'>
   <Sejour idClient='30' debut='2002-06-24'
      nbPlaces='5'/>
</Station>
<Client id='30'  nom='Kerouac'
     prenom='Jack' ville='New York'/>
```

Solution ultime

On choisit une racine (par exemple les stations) et on imbrique tout

- les séjours sont des éléments dans les stations
- les clients sont des éléments dans les séjours

On introduit une certaine redondance : les clients apparaissent plusieurs fois

Solution ultime : exemple

```
<?xml version='1.0' encoding='ISO-8859-1?>
<Station nomStation='Venusa'
    region='Antilles' tarif='1200.00'>
  <Sejour  nom='Kerouac'
           prenom='Jack' ville='New York'/>
           debut='2001-08-03'
           nbPlaces='3'/>
</Station>
<Station nomStation='Farniente'
    region='Océan Indien' tarif='1500.00'>
   <Sejour  nom='Kerouac'
            prenom='Jack' ville='New York'
            debut='2002-06-24'
            nbPlaces='5'/>
</Station>
```

Déclaration de la DTD

Pourquoi une DTD ?

- En général on exporte une base de données pour fournir une « vue » XML à d'autres utilisateurs.
- Il faut donc fournir une description de la structure XML.
- On utilise (pour l'instant) les *Document Type Definition*

NB : la DTD peut aussi servir à **valider** des documents (en utilisant un parseur validant)

DTD de la base Station

On suppose que :

- les colonnes sont représentées par des attributs XML ;
- le chemin d'accès principal est la station ;
- pour chaque station on trouve, imbriqués, les séjours de la station, et dans chaque séjour les clients qui ont séjourné dans la station ;
- les activités de la station sont représentées par des éléments indépendants, avec un lien de navigation.

Qu'est-ce qu'une DTD ?

Un description du contenu (attributs et fils) de chaque élément. Exemple général :

```
<!ELEMENT Stations (Station*)>

<!ELEMENT Station (Sejour*)>
<!ATTLIST Station
          nomStation ID #REQUIRED
          capacite   CDATA #IMPLIED
          lieu       CDATA #REQUIRED
          tarif      CDATA #REQUIRED
          region     (Antilles Europe
                      Amérique Asie) #REQUIRE
>
```

Contenu d'un élément

La forme générale de définition d'un élément est :
`<!ELEMENT nom structure>`
La structure décrit le modèle de contenu avec le type, l'ordre et le nombre d'occurrences des fils.

- `A, B` : un A suivi d'un B
- `A, B*` : un A suivi de 0 ou plusieurs B
- `A, B?, C+`: un A, suivi de 0 ou 1 B, suivi d'au moins un C
- `A, (B|C)*` : un A suivi de plusieurs B ou C dans n'importe quel ordre

Types et contraintes dans une DTD

Les principaux types sont :

- `CDATA` (pour les attributs) et `#PCDATA` (pour les éléments) : chaînes de caractères
- `ID` : identifiant d'un nœud dans le document.
- `IDREF` : réf. un nœud du même document

Les contraintes :

- `#IMPLIED` pour les attributs facultatifs
- `#REQUIRED` pour les attributs obligatoires

Suite de la DTD Station

```
<!ELEMENT Sejour (Client)>
<!ATTLIST Sejour
          debut    CDATA #REQUIRED
          nbPlaces CDATA #REQUIRED
>

<!ELEMENT Client  EMPTY>
<!ATTLIST Client
          id       ID    #REQUIRED
          nom      CDATA #REQUIRED
          prenom   CDATA #REQUIRED
          ville    CDATA #REQUIRED
          region   CDATA #REQUIRED
          solde  CDATA #REQUIRED
>
```

Fin de DTD Station

On déclare une référence depuis *Activité* vers *Station*.

```
<!ELEMENT Activite  EMPTY>
<!ATTLIST Activite
          nomStation IDREF #REQUIRED
          libelle   CDATA #REQUIRED
          prix      CDATA #IMPLIED
>
```

NB : identifiants et références ne tiennent pas compte du

type des éléments.

Architectures

Problématique

- une base de données, contenant des informations gérées par des applications de gestion ou autre
- des informations textuelles et informelles relatives à l'activité de l'entreprise
- nécessité de publier toutes ces informations dans différents formats (le Web, une plaquette, un rapport d'activité...)

Problème : comment éviter de manipuler **tous** les outils à la fois

Séparation des points de vue

- la base de données est relationnele : efficacité, simplicité, services avancés de traitement des données (transactions, sécurité, etc)
- une « vue » XML des données, intégrant celles issues de la base, et les informations textuelles diverses
- des traitements de mise en forme, s'appuyant sur la vue XML, chacun à destination d'un support particulier

Les solutions techniques (1)

Reprise des solutions pour HTML dynamique (servlets, JSP, PHP)

- Permettent une connexion à la base et un traitement des paramètres CGI
- Conçus pour HTML, donc
  - on ne respecte pas la séparation des points de vue (mélange XML + instructions Java ou PHP)
  - pas commode d'enchaîner avec une transformation XSLT

Les solutions techniques (2)

Solution plus satisfaisante :

- inclure directement des instructions d'interrogation de la base de données dans un document XML
- évaluer ces instructions au moment où on accède au document

Deux exemples :

- XSP, les JSP adaptées à XML
- XSQL, d'Oracle

L'utilitaire XSQL d'ORACLE

Oracle propose un *XML Development Kit* avec : DOM, SAX, processeur XSLT et nombreux utilitaires (en java). XSQL est un SQL mis à la sauce XML. Il permet :

- de se connecter à la base
- d'effectuer une requête
- de mettre en forme XML le résultat, selon certains paramètres
- d'enchaîner avec une transformation XSLT

Une page XSQL minipale

C’est un document XML.

```
<?xml version="1.0"?>
<xsql:query
        connection="maConnexion"
        xmlns:xsql="urn:oracle-xsql">
  SELECT ’Hello World’ AS TEXT FROM Dual
</xsql:query>
```

Les instructions XSQL sont identifiées par *l’espace de nom*

```
xsql:
```

Le résultat

C’est encore document XML, avec la représentation en éléments du résultat de la requête.

```
<?xml version="1.0"?>
<ROWSET>
  <ROW num=’1’>
    <TEXT>Hello World</TEXT>
  </ROW>
</ROWSET>
```

Pour effectuer une transformation

Très simple : on ajoute une instruction indiquant le programme de transformation à appliquer.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
          href="HelloWorld.xsl"?>
<xsql:query connection="maConnexion"
        xmlns:xsql="urn:oracle-xsql">
  SELECT 'Hello World' AS TEXT FROM Dual
</xsql:query>
```

NB : la requête est effectuée **avant** la transformation.

Utilisation plus générale de XSQL

```
<Promotion NO="1">
  <Description>
   Cette semaine nous proposons une
   promotion exceptionnelle sur bla bla bla
  </Description>
  <xsql:query connection="connexionBase"
                  xmlns:xsql="urn:oracle-xsql">
    SELECT * FROM Station
    WHERE nomStation='Passac'
  </xsql:query>
</Promotion>
```

Utilisable dans les contextes Servlets, JSP, etc.

Le format de sortie

Par défaut le format de sortie est le suivant :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ROWSET>
 <ROW num="1">
   <NOMSTATION>Passac</NOMSTATION>
   <CAPACITE>400</CAPACITE>
   <LIEU>Alpes</LIEU>
   <REGION>Europe</REGION>
   <TARIF>1000</TARIF>
 </ROW>
</ROWSET>
```

Mais tout est paramétrable

Ce qu'on peut faire avec XSQL

On fait les choses simples simplement, et on peut faire des choses compliquées.

- Utiliser des paramètres dans la requête
- Placer plusieurs requêtes dans une page
- Contrôler le format XML de sortie
- Produire plusieurs formats de sortie avec XSLT

Voir l'URL *http://www.ie2.u-psud.fr:8080/xsql*

Exemple de paramètres

Des paramètres (HTTP, Cookies, autre) peuvent être exploités dans la requête.

```
<?xml version="1.0"?>
<xsql:query
        connection="maConnexion"
        xmlns:xsql="urn:oracle-xsql">
  SELECT * FROM Station
  WHERE nomStation='{@nom}'
</xsql:query>
```

Contrôle du résultat XML

On peut remplacer `ROWSET` et `ROW` par des noms explicites.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsql:query
        connection="maConnexion"
        xmlns:xsql="urn:oracle-xsql"
        rowset-element='STATIONS'
        row-element='STATION'>
  SELECT * FROM Station
  WHERE nomStation='Passac'
</xsql:query>
```

Le résultat

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<STATIONS>
 <STATION num="1">
   <NOMSTATION>Passac</NOMSTATION>
   <CAPACITE>400</CAPACITE>
   <LIEU>Alpes</LIEU>
   <REGION>Europe</REGION>
   <TARIF>1000</TARIF>
 </STATION>
</STATIONS>
```

NB : on peut aussi indiquer l'identifiant, gérer les valeurs nulles, etc.

Avec paramètres

Les paramètres peuvent aussi être utilisés dans les atttributs de <xsql:query>. Exemple :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <xsql:query connection="connexionVisiteur"
                  xmlns:xsql="urn:oracle-xsql"
                  rowset-element="{@racine}"
                  row-element="{@element}">
    SELECT * FROM Station
    WHERE nomStation='{@nomStation}'
  </xsql:query>
```

Et l'imbrication

Par défaut la représentation est « à plat », comme en relationnel. On peut spécifier la création de XML imbriqué de la manière suivante :

```
SELECT nomStation, capacite,
     CURSOR(SELECT libelle, prix
            FROM Activite
            WHERE A.nomStation = S.nomStatio
            AS activites
FROM  Station S
```

Le résultat

```
<ROWSET>
  <ROW num='1'>
    <nomStation>Passac</nomStation>
    <capacite>400</capacite>
    <activites>
     <activites_row num='1'>
      <libelle>Ski</libelle>
      <prix>200.00</prix>
     </activites_row>
     <activites_row num='2'>
       <libelle>Piscine</libelle>
       <prix>20.00</prix>
     </activites_row >
    </activites>
  </ROW>
</ROWSET>
```

Transformations XSLT

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="Film.xsl" type="text/
<FILM>
    <TITRE>Alien</TITRE>
    <AUTEUR>Ridley Scott</AUTEUR>
    <ANNEE>1979</ANNEE>
    <GENRE>Science-fiction</GENRE>
    <PAYS>Etats Unis</PAYS>
    <RESUME>Près d'un vaisseau spatial échou
       planète, des Terriens en mission déco
       "oeufs". Ils en ramènent un à bord, i
       d'introduire parmi eux un huitième pa
       féroce et meurtrier.
    </RESUME>
</FILM>
```

Avec `xsql:include-xml`

On inclut le document XML.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="Cinema.xsl"
                 type="text/xsl"?>
<page xmlns:xsql="urn:oracle-xsql">
  <xsql:include-xml href="Epee.xml"/>
</page>
```

Pour la version WML

Avantage : on peut lui appliquer une autre transformation XSLT.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="CinemaWML.xsl"
                 type="text/xsl"?>
<page xmlns:xsql="urn:oracle-xsql">
  <xsql:include-xml href="Epee.xml"/>
</page>
```