Modernisation, développement d'applications et DB2 sous IBM i
*Technologies, outils et nouveautés 2013-2014*

13 et 14 mai 2014 – IBM Client Center Paris, Bois-Colombes

# S17 – DB2/SQL : composition, validation et décomposition XML

*Mercredi 14 mai – 11h00-12h30*

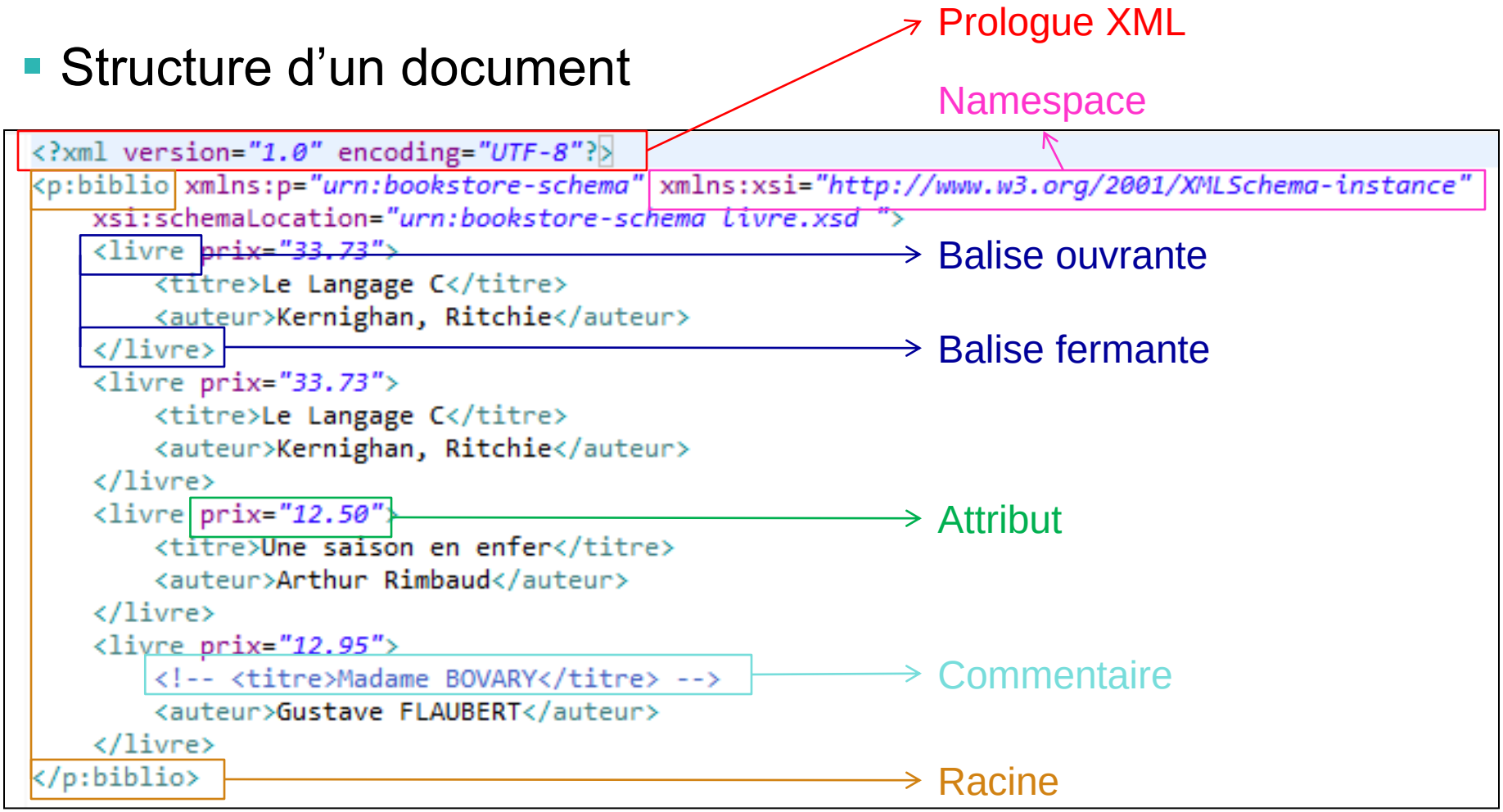
Nathanaël BONNET – Gaïa Mini Systèmes

# Prérequis

- En 7.1, support de XML dans le langage SQL
  - Normalisation SQLXML
    - http://en.wikipedia.org/wiki/SQL/XML
  - Le nom commercial IBM est « PureXML »
    - Sauf pour DB2 for i …
- Fonctions supportées
  - Stockage en format XML natif
  - Fonctions de publication XML
  - Fonctions de validation par un schéma XSD
  - Sérialisation
  - Transformation XSL
  - Décomposition des données XML (shredding et XPath)

# Rappels XML

- Structure d'un document



# Rappels XML

- eXtensible Markup Langage
  - Langage de balisage extensible
  - Balisage
    - Une balise est identifiée par des chevrons < et >
    - Contient une valeur
      - De type primitif ou complexe
  - Extensible
    - Il est possible de définir sa propre grammaire dans un espace de nom personnalisé
      - Ensemble de balises, d'attributs, de contraintes …
- Un document XML
  - Document bien formé
    - Syntaxiquement correct
  - Document valide
    - Respecter les règles grammaticales spécifiques (de l'espace de nom)

# Rappels XML

- XML est aujourd'hui
  - Le langage (encodage) permettant de représenter des données et leur structure
  - Le langage supporté par tous les protocoles de communication car en mode texte
  - Le langage le plus répandu dans le monde informatique
  - Nécessaire pour l'échange de données entre applications

- Il est aujourd'hui indispensable dans nos SI

# Type de colonne XML

- Type XML
  - Taille maxi = 2 Go
    - Rappel : taille maxi d'une ligne = 3,5 Go
  - Encodage par défaut = UTF-8 (CCSID 1208)
    - Sinon se référer à SQL_XML_DATA_CCSID
    - Le CCSID 65535 n'est pas supporté
    - La fonction XMLPARSE s'exécute en UTF-8. L'utilisation d'un autre CCSID force des conversions supplémentaires
  - Le stockage interne est un DBCLOB
- Limites
  - Ne peut pas être une clé primaire, clé unique, clé étrangère
  - Ne peut pas faire partie des clés d'un index, ni de la clause where pour les index dérivés
  - N'est pas admise en clé de partitionnement

# Utilisation

- Base de test

```
call qsys.create_xml_sample('MALIB')
```

- Exemple

select * from nb_xml.customer -

| CID | INFO | HISTORY |
|---|---|---|
| 1000 | <customerinfo Cid="1000"><name>Kathy Smith</name><addr country... | - |
| 1001 | <customerinfo Cid="1001"><name>Kathy Smith</name><addr country... | - |
| 1002 | <customerinfo Cid="1002"><name>Jim Noodle</name><addr country=... | - |
| 1003 | <customerinfo Cid="1003"><name>Robert Shoemaker</name><addr co... | - |
| 1004 | <customerinfo Cid="1004"><name>Matt Foreman</name><addr countr... | - |
| 1005 | <customerinfo Cid="1005"><name>Larry Menard</name><addr countr... | - |

```
                                                Affichage des données

Première ligne à afficher . . _____
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....
                    CID   INFO                                          HISTORY
                  1.000   *POINTER                                      -
                  1.001   *POINTER                                      -
                  1.002   *POINTER                                      -
                  1.003   *POINTER                                      -
                  1.004   *POINTER                                      -
                  1.005   *POINTER                                      -
********  Fin de données  ********
```

# Utilisation

- Avec STRSQL il faut utiliser XMLSERIALIZE

```
SELECT CID,
       xmlserialize(INFO as varchar(1024))
FROM customer
```

- Si le job est en CCSID 65535

```
SELECT CID,
       xmlserialize(INFO as varchar(1024) ccsid 1147)
FROM customer
```

```
                                                Affichage des données

Première ligne à afficher . . _____
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
                    CID   XMLSERIALIZE
                  1.000   <customerinfo Cid="1000"><name>Kathy Smith</name><ad
                  1.001   <customerinfo Cid="1001"><name>Kathy Smith</name><ad
                  1.002   <customerinfo Cid="1002"><name>Jim Noodle</name><add
                  1.003   <customerinfo Cid="1003"><name>Robert Shoemaker</nam
                  1.004   <customerinfo Cid="1004"><name>Matt Foreman</name><a
                  1.005   <customerinfo Cid="1005"><name>Larry Menard</name><a
```

# Utilisation

- Les insertions et modifications se font sur la valeur complète

```
update customer
set info = '<customerinfo Cid="1006"><name>Nathanaël
BONNET</name><addr country="France"><street>69 Rue Gorge
de Loup</street><city>Lyon</city><prov-state>Rhône</prov-
state><pcode-zip>69009</pcode-zip></addr><phone
type="work">04 78 00 00 00</phone></customerinfo>'
where cid = 1006 ;
```

- Il est possible d'insérer une valeur XML contenue dans un fichier stream (IFS)

```
insert into order_records (order_doc)
values (get_xml_file('/home/nbonnet/VehicleOrder.xml'));
```

# Composition XML depuis DB2

- Un ensemble de fonctions SQL a été ajouté et permet la publication de données XML
  - La création de données XML depuis les données relationnelles existantes

# Composition XML depuis DB2

- UDFs

| Fonction | Usage |
|---|---|
| XMLATTRIBUTES | Production d'attribut avec XMLELEMENT |
| XMLCOMMENT | Production de commentaires au format XML |
| XMLCONCAT | Concaténation de deux fragments XML |
| XMLDOCUMENT | Production d'un document XML depuis une valeur XML |
| XMLELEMENT | Production d'un élément XML depuis son nom et sa valeur |
| XMLFOREST | Production d'éléments XML depuis les colonnes d'une table |
| XMLNAMESPACES | Production d'espaces de nom |
| XMLPARSE | Produit un flux XML depuis une chaine de caractères, avec options |
| XMLPI | Production de balises d'instruction |
| XMLROW | Production d'une ligne XML pour les colonnes d'une table |
| XMLSERIALIZE | Sérialise un flux XML sous forme de caractères ou autre |
| XMLTEXT | Retourne une chaine de caractères compatible XML |
| XMLCAST | Retourne la valeur en type XML |

# Composition XML depuis DB2

- Autres fonctions
  - Fonctions de groupe
    - Utilisables avec GROUP BY

| Fonction | Usage |
|---|---|
| XMLAGG | Fonction de groupe : agrégation de valeurs XML |
| XMLGROUP | Fonction de groupe : groupement de valeurs XML |

  - Fonctions utiles

| Fonction | Usage |
|---|---|
| GET_XML_FILE | Retourne un document XML depuis un fichier IFS |

# Composition XML depuis DB2

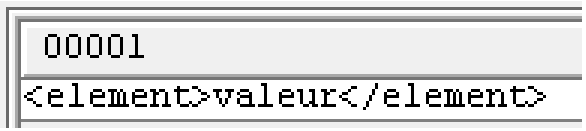
- XMLPARSE, XMLCAST et XMLDOCUMENT

```
select xmlcast(
   xmlparse(document '<element>valeur</element>')
            as xml ccsid 1208 normalized)
from sysibm.sysdummy1 ;
```

– XMLPARSE retourne une valeur XML

– XMLCAST retourne la valeur transmise en type XML

  – Depuis un type XML ou non




```
select xmldocument(
   xmlparse( document '<element>valeur</element>' ) )
from sysibm.sysdummy1 ;
```

– XMLDOCUMENT crée un document XML depuis une valeur XML

  – Correspond au type de colonne XML (contrôles)

# Composition XML depuis DB2

- XMLELEMENT
  - C'est la base de la construction de flux XML
  - Produit un élément XML
    - dont la valeur peut être XML
    - avec des attributs
    - des namespaces

```
select xmlelement( name "emplacement", location)
from inventory ;
```

| 00001 |
| --- |
| <emplacement/> |
| <emplacement>Store</emplacement> |
| <emplacement>Store</emplacement> |
| <emplacement>Warehouse</emplacement> |

# Composition XML depuis DB2

```
select xmlelement( name "inventaire",
          xmlelement( name "pid", pid)  ,
          xmlelement( name "emplacement", location ) )
from inventory ;
```

| 00001 |
|---|
| <inventaire><pid>100-100-01</pid><emplacement/></inventaire> |
| <inventaire><pid>100-101-01</pid><emplacement>Store</emplacement></inventaire> |
| <inventaire><pid>100-103-01</pid><emplacement>Store</emplacement></inventaire> |
| <inventaire><pid>100-201-01</pid><emplacement>Warehouse</emplacement></inventaire> |

```
select xmlelement( name "pid",
  xmlattributes(location as "emplacement",
                quantity as "qte") , pid)
from inventory ;
```

| 00001 |
|---|
| <pid qte="5">100-100-01</pid> |
| <pid emplacement="Store" qte="25">100-101-01</pid> |
| <pid emplacement="Store" qte="55">100-103-01</pid> |
| <pid emplacement="Warehouse" qte="99">100-201-01</pid> |

# Composition XML depuis DB2

- Echappement de caractères
  - XMLTEXT

```
select xmltext('test de base'),
       xmltext('il parait que 2 < 3 & 3 > 2 !')
from sysibm.sysdummy1 ;
```

| 00001 | 00002 |
|---|---|
| test de base | il parait que 2 < 3 & 3 > 2 ! |

- D'autres fonctions utilitaires
  - XMLSERIALIZE, XMLCOMMENT, XMLCONCAT, XMLPI …

# Composition XML depuis DB2

- XMLFOREST et XMLROW
  - XMLFOREST retourne une séquence d'éléments XML
  - XMLROW retourne une valeur XML bien formée (i.e. avec une racine)

```
select xmlforest(pid as "reference",
                 name as "nom",
                 price as "prix",
                 promoprice
                 option null on null)
from product ;
```

| 00001 |
| --- |
| <reference>100-100-01</reference><nom>Snow Shovel, Basic 22 inch</nom><prix>9.99</prix><PROMOPRICE>7.25</PROMOPRICE> |
| <reference>100-101-01</reference><nom>Snow Shovel, Deluxe 24 inch</nom><prix>19.99</prix><PROMOPRICE>15.99</PROMOPRICE> |
| <reference>100-103-01</reference><nom>Snow Shovel, Super Deluxe 26 inch</nom><prix>49.99</prix><PROMOPRICE>39.99</PROMOPRICE> |
| <reference>100-201-01</reference><nom>Ice Scraper, Windshield 4 inch</nom><prix>3.99</prix> |

# Composition XML depuis DB2

```
select xmlrow(pid as "reference",
              name as "nom",
              price as "prix",
              promoprice )
from product ;
```

```
00001
<row><reference>100-100-01</reference><nom>Snow Shovel, Basic 22 inch</nom><prix>9.99</prix><PROMOPRICE>7.25</PROMOPRIC
<row><reference>100-101-01</reference><nom>Snow Shovel, Deluxe 24 inch</nom><prix>19.99</prix><PROMOPRICE>15.99</PROMOP
<row><reference>100-103-01</reference><nom>Snow Shovel, Super Deluxe 26 inch</nom><prix>49.99</prix><PROMOPRICE>39.99</
<row><reference>100-201-01</reference><nom>Ice Scraper, Windshield 4 inch</nom><prix>3.99</prix></row>
```

```
select xmlrow(pid as "reference",
              name as "nom",
              price as "prix",
              promoprice
              OPTION ROW "produit" AS ATTRIBUTES)
from product ;
```

```
00001
<produit reference="100-100-01" nom="Snow Shovel, Basic 22 inch" prix="9.99" PROMOPRICE="7.25"/>
<produit reference="100-101-01" nom="Snow Shovel, Deluxe 24 inch" prix="19.99" PROMOPRICE="15.99"/>
<produit reference="100-103-01" nom="Snow Shovel, Super Deluxe 26 inch" prix="49.99" PROMOPRICE="39.99"/>
<produit reference="100-201-01" nom="Ice Scraper, Windshield 4 inch" prix="3.99"/>
```

# Composition XML depuis DB2

- Fonctions de groupe : XMLAGG et XMLGROUP

  - XMLAGG retourne une séquence d'éléments XML

```
select xmlagg( xmlforest( poid as "identifiant" ,
                          rderdate as "date" )
               order by orderdate desc )
from nb_xml.purchaseorder
group by custid ;
```

```
- <identifiant>5002</identifiant><date>2004-02-29</date>
- <identifiant>5006</identifiant><date>2006-03-01</date>
  <identifiant>5000</identifiant><date>2006-02-18</date>
  <identifiant>5003</identifiant><date>2005-02-28</date>
- <identifiant>5001</identifiant><date>2005-02-03</date>
- <identifiant>5004</identifiant><date>2005-11-18</date>
```

# Composition XML depuis DB2

- XMLGROUP retourne une valeur XML bien formée (avec une racine)

```
select xmlgroup(custid as "nocli",
                status as "status"
                option root "commandes" row "commande")
from purchaseorder
group by custid ;
```

| POID | STATUS | CUSTID | ORDERDATE |
|---|---|---|---|
| 5002 | Shipped | 1001 | 2004-02-29 |
| 5000 | Unshipped | 1002 | 2006-02-18 |
| 5003 | Shipped | 1002 | 2005-02-28 |
| 5006 | Shipped | 1002 | 2006-03-01 |
| 5001 | Shipped | 1003 | 2005-02-03 |
| 5004 | Shipped | 1005 | 2005-11-18 |

- <commandes><commande><nocli>1001</nocli><status>Shipped</status></commande></commandes>
- <commandes><commande><nocli>1002</nocli><status>Unshipped</status></commande><commande><nocli>1002</nocli><status>Shipped</status></commande><commande><nocli>1002</nocli><status>Shipped</status></commande></commandes>
- <commandes><commande><nocli>1003</nocli><status>Shipped</status></commande></commandes>
- <commandes><commande><nocli>1005</nocli><status>Shipped</status></commande></commandes>

# Composition XML depuis DB2

- Espaces de noms

```
select xmlelement(name "ns1:reference",
   xmlnamespaces('http://gaia.fr'  as "ns1",
                 'http://gaia2.fr' as "ns2"),
   xmlattributes( name  as "ns1:nom",
                  price as "ns2:prix") ,
       pid )
from product ;
```

```
– <ns1:reference xmlns:ns1="http://gaia.fr" xmlns:ns2="http://gaia2.fr"
  ns1:nom="Snow Shovel, Basic 22 inch" ns2:prix="9.99">100-100-01</ns1:reference>
– <ns1:reference xmlns:ns1="http://gaia.fr" xmlns:ns2="http://gaia2.fr"
  ns1:nom="Snow Shovel, Deluxe 24 inch" ns2:prix="19.99">100-101-
  01</ns1:reference>
– <ns1:reference xmlns:ns1="http://gaia.fr" xmlns:ns2="http://gaia2.fr"
  ns1:nom="Snow Shovel, Super Deluxe 26 inch" ns2:prix="49.99">100-103-
  01</ns1:reference>
– <ns1:reference xmlns:ns1="http://gaia.fr" xmlns:ns2="http://gaia2.fr"
  ns1:nom="Ice Scraper, Windshield 4 inch" ns2:prix="3.99">100-201-
  01</ns1:reference>
```

# Validation XML

- En complément des fonctions de publication XML, DB2 permet également la validation
  - Valider un document XML par un schéma XSD
    - XML Schema Definition
    - Cf http://fr.wikipedia.org/wiki/XML_Schema et http://www.w3.org/XML/Schema
  - XSD est un fichier XML décrivant la structure d'un document XML
    - Nom des éléments, des attributs, compositions
    - Cardinalités, contraintes de taille, de valeur …
- La validation par fichier DTD (Document Type Definition) n'est pas supportée
  - Avantageusement remplacée par XSD
    - Cf http://fr.wikipedia.org/wiki/Document_Type_Definition et http://www.w3.org/TR/html401/sgml/dtd.html

# Validation XML

- Exemple de xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="urn:bookstore-schema" targetNamespace="urn:bookstore-schema"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:complexType name="livre_type">
        <xs:sequence>
            <xs:element name="titre" type="xs:string" minOccurs="1" maxOccurs="1"></xs:element>
            <xs:element name="auteur" minOccurs="0" maxOccurs="1">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:maxLength value="25"></xs:maxLength>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
        </xs:sequence>
        <xs:attribute name="prix" type="xs:decimal"></xs:attribute>
    </xs:complexType>

    <xs:complexType name="biblio_type">
        <xs:sequence>
            <xs:element name="livre" type="livre_type" minOccurs="0" maxOccurs="unbounded"></xs:element>
        </xs:sequence>
    </xs:complexType>

    <xs:element name="biblio" type="biblio_type"></xs:element>
</xs:schema>
```

C'est un document XML

Espace de nommage cible

Espace de nommage XSD

Définition de structure

Définition de balise avec contrainte d'occurrence et de taille

Définition d'attribut

# Validation XML

- DB2 supporte l'usage de XSD au travers d'un référentiel de XSD nommé XSR
  - XML Schema Repository
- Un ensemble de fonctionnalités est fourni
  - Procédures cataloguées dans SYSPROCS

| Nom | Nature | Rôle |
|---|---|---|
| XMLVALIDATE | Fonction | Valide un schéma XML depuis un XSD enregistré dans XSR |
| XSR_ADDSCHEMADOC | Procédure | Ajoute un schéma XSD |
| XSR_COMPLETE | Procédure | Finalise l'enregistrement d'un schéma XSD |
| XSR_REGISTER | Procédure | Débute l'enregistrement d'un schéma XSD |
| XSR_REMOVE | Procédure | Supprime un schéma XSD |

  - L'enregistrement de XSD produit des objets natifs de type *SQLXSR
    - SQL Extensible Markup Language Schema Repository

# Validation XML

- Enregistrer le XSD (1 fois)
  - XSR_REGISTER
  - XSR_ADDSCHEMADOC, si nécessaire
  - XSR_COMPLETE
  - **Remarque :** le catalogage des XSD doit se faire sous contrôle transactionnel
- Valider
  - XMLVALIDATE (n fois)

# Validation XML

- Exemple

```
-- XSR_REGISTER doit être sous contrôle transactionnel
set transaction isolation level read uncommitted ,
                                read write ;

-- enregistrer le schéma xsd
CALL SYSPROC.XSR_REGISTER ('GMS_NB', 'livre', null,
                           GET_XML_FILE('/home/NB/ex1/livre.xsd'), null);
CALL SYSPROC.XSR_COMPLETE ('GMS_NB', 'livre', null, 0);

commit ;

-- valider le flux
select XMLVALIDATE( XMLPARSE(
    DOCUMENT GET_XML_FILE( '/home/NB/ex1/biblio.xml' ) )
    ACCORDING TO XMLSCHEMA ID GMS_NB.livre )
from sysibm.sysdummy1 ;
```

Bibliothèque

Nom du schéma

Valeur du schéma

# Validation XML

- Cas de XSD avec include

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="urn:bookstore-schema"
    targetNamespace="urn:bookstore-schema"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">

    <xs:include schemaLocation="dvd.xsd"></xs:include>


    <xs:redefine schemaLocation=""></xs:redefine>
    <xs:complexType name="livre_type">
        <xs:sequence>
            <xs:element name="titre" type="xs:string" minOccurs="1"
                maxOccurs="1">
            </xs:element>
            <xs:element name="auteur" minOccurs="0" maxOccurs="1">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:maxLength value="25"></xs:maxLength>
                    </xs:restriction>
                </xs:simpleType>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="urn:bookstore-schema" elementFormDefault="qualified"
    xmlns="http://www.w3.org/2001/XMLSchema" xmlns:tns="urn:bookstore-schema">

    <complexType name="dvd_type">
        <sequence>
            <element name="titre" type="string" minOccurs="1" maxOccurs="1"></element>
            <element name="realisateur" type="string" minOccurs="0" maxOccurs="2"></element>
        </sequence>
        <attribute name="annee" type="date"></attribute>
    </complexType>

</schema>
```

# Validation XML

- Exemple

```
CALL SYSPROC.XSR_REGISTER (
   'GMS_NB',      -- Bibliothèque
   'livre_dvd',   -- Nom interne
   'livre.xsd',   -- schemaLocation
   GET_XML_FILE('/home/NB/ex2/livre.xsd'), -- XSD sur l'IFS
   NULL);         -- Propriétés

CALL SYSPROC.XSR_ADDSCHEMADOC (
   'GMS_NB',      -- Bibliothèque
   'livre_dvd',   -- Nom interne
   'dvd.xsd',     -- schemaLocation
   GET_XML_FILE('/home/NB/ex2/dvd.xsd'),   -- XSD sur l'IFS
   NULL);         -- Propriétés
```

# Décomposition XML par shredding

- La décomposition (« shredding », littéralement « déchiquetage »), permet
  - d'insérer les valeurs contenues dans un document XML dans les tables existantes de la BD, en format non XML
- Cette technique est utile car
  - Nos applications fonctionnent avec une BD relationnelle et sont conçues pour une BD relationnelle
  - La plupart des échanges d'informations sont réalisés en format XML
  - Le shredding permet d'automatiser la transposition
    - données XML -> données relationnelles

# Décomposition XML par shredding

- Il faut enregistrer une XSD
  - Permet de décrire le document XML
  - A laquelle on ajoute des attributs et éléments (annotations) spécifiques pour contrôler la décomposition
    - db2-xdb:rowSet
    - db2-xdb:table
    - db2-xdb: …
- Ces attributs et éléments supplémentaires sont définis
  - Dans un espace de nom **`http://www.ibm.com/xmlns/prod/db2/xdb1`**
  - Documentés ici : infocentre 7.1

# Décomposition XML par shredding

- La plupart de ces annotations peuvent être indiquées sous forme d'attribut ou sous forme d'élément
  - Attribut

```
<xs:element name="nom" minOccurs="1" maxOccurs="1"
            db2-xdb:rowSet="XML_CLIENT" db2-xdb:column="NAME">
```

  - Elément

```
<xs:element name="nom" minOccurs="1" maxOccurs="1">
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:rowSetMapping>
        <db2-xdb:rowSet>XML_CLIENT</db2-xdb:rowSet>
        <db2-xdb:column>NAME</db2-xdb:column>
      </db2-xdb:rowSetMapping>
   </xs:appinfo>
</xs:annotation>
```

# Décomposition XML par shredding

- Exemple

Les tables à alimenter sont recherchées dans la bibliothèque GMS_NB

- L'élément XML « client_type\nom » alimentera la colonne NAME de la table XML_CLIENT
- L'élément XML « client_type\prenom » alimentera la colonne FIRST_NAME de la table XML_CLIENT
- L'attribut « idClient » de l'entité XML « client_type » alimentera la colonne ID de la table XML_CLIENT

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:db2-xdb="http://www.ibm.com/xmlns/prod/db2/xdb1">

    <!-- Indique le schéma SQL pour le shredding -->
    <xs:annotation>
        <xs:appinfo>
            <db2-xdb:defaultSQLSchema>GMS_NB</db2-xdb:defaultSQLSchema>
        </xs:appinfo>
    </xs:annotation>
```

```
    <xs:complexType name="client_type">
        <xs:sequence>
            <xs:element name="nom" minOccurs="1" maxOccurs="1"
                db2-xdb:rowSet="XML_CLIENT" db2-xdb:column="NAME">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:minLength value="3"></xs:minLength>
                        <xs:maxLength value="25"></xs:maxLength>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="prenom" minOccurs="0" maxOccurs="1"
                db2-xdb:rowSet="XML_CLIENT" db2-xdb:column="FIRST_NAME">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:minLength value="3"></xs:minLength>
                        <xs:maxLength value="25"></xs:maxLength>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="adresse" type="adresse_type" minOccurs="0"
                maxOccurs="2"></xs:element>
        </xs:sequence>
        <xs:attribute name="idClient" type="xs:int"
            db2-xdb:rowSet="XML_CLIENT" db2-xdb:column="ID"></xs:attribute>
    </xs:complexType>
```

# Décomposition XML par shredding

- Exemple

```
-- enregistrer le schéma xsd
CALL SYSPROC.XSR_REGISTER ('GMS_NB', 'client', null,
                GET_XML_FILE('/home/NB/shred/client.xsd'),
                null);
-- compléter l'enregistrement du schéma
CALL SYSPROC.XSR_COMPLETE ('GMS_NB', 'client', null, 1);

-- appel de la procédure de décomposition
call sysproc.xdbdecompxml( 'GMS_NB', 'client',
GET_XML_FILE('/home/NB/shred/clients.xml'), null ) ;

-- visualiser le résultat
select * from xml_client ;
```

| ID | NAME | | FIRST_NAME | | LAST_UPD |
|---|---|---|---|---|---|
| 1 | DEBLOUZE | ... | Agathe | ... | 2014-03-21 17:41:41.643017 |
| 2 | EPI | ... | Fanny | ... | 2014-03-21 17:41:41.643017 |
| 3 | OUZI | ... | Jacques | ... | 2014-03-21 17:41:41.643017 |

# Décomposition XML par shredding

- Il est possible de stocker certaines valeurs sous forme XML dans la BD
  - Pratique pour les cardinalités non définies
- Exemple
  - Flux XML

```
<?xml version="1.0" encoding="UTF-8"?>
<clients>
    <client idClient="1">
        <nom>DEBLOUZE</nom>
        <prenom>Agathe</prenom>
        <adresse idAdresse="1">
            <ligne1>Allee de la musique</ligne1>
            <CP>69009</CP>
        </adresse>
    </client>
</clients>
```

  - Extrait XSD

```
<xs:element name="clients" type="clients_type"
            db2-xdb:rowSet="XML_DATA"
            db2-xdb:column="DONNEES"
            db2-xdb:contentHandling="serializeSubtree"></xs:element>
```

    - **db2-xdb:contentHandling="serializeSubtree"** indique le mode de gestion de la valeur

```
DONNEES
<clients><client idClient="1"><nom>DEBLOUZE</nom><prenom>Agathe</prenom><adresse idAdre...
```

# Décomposition XML par shredding

- Décomposition d'un flux vers plusieurs tables
  - Indiquer l'ordre de traitement des tables pour respecter les contraintes d'intégrité référentielle

```
<xs:annotation>
    <xs:appinfo>
        <db2-xdb:rowSetOperationOrder>
            <db2-xdb:order>
                <db2-xdb:rowSet>XML_CLIENT</db2-xdb:rowSet>
                <db2-xdb:rowSet>XML_ADRESS</db2-xdb:rowSet>
            </db2-xdb:order>
        </db2-xdb:rowSetOperationOrder>
    </xs:appinfo>
</xs:annotation>
```

  - Insérer la même valeur dans deux tables distinctes

```
<xs:attribute name="idClient" type="xs:int">
    <xs:annotation>
        <xs:appinfo>
            <db2-xdb:rowSetMapping>
                <db2-xdb:rowSet>XML_CLIENT</db2-xdb:rowSet>
                <db2-xdb:column>ID</db2-xdb:column>
            </db2-xdb:rowSetMapping>
            <db2-xdb:rowSetMapping>
                <db2-xdb:rowSet>XML_ADRESS</db2-xdb:rowSet>
                <db2-xdb:column>ID_CLIENT</db2-xdb:column>
            </db2-xdb:rowSetMapping>
        </xs:appinfo>
    </xs:annotation>
</xs:attribute>
```

# Décomposition XML par shredding

- Résultat
  - Table XML_CLIENT

| ID | NAME | FIRST_NAME | LAST_UPD |
|---|---|---|---|
| 1 | DEBLOUZE | Agathe | 2014-03-24 11:54:53.538576 |
| 2 | EPI | Fanny | 2014-03-24 11:54:53.538576 |
| 3 | OUZI | Jacques | 2014-03-24 11:54:53.538576 |

  - Table XML_ADRESS

| ID | ID_CLIENT | LIGNE1 | LIGNE2 | CP | LAST_UPD |
|---|---|---|---|---|---|
| 2 | 3 | 3 Boulevard de l'auto-défense ... | ... | 75010 | 2014-03-24 11:54:53.538576 |
| 1 | 3 | 3 Allée de la piscine ... | ... | 69003 | 2014-03-24 11:54:53.538576 |
| 1 | 2 | Lotissement bel air ... | 2 Rue du calendrier ... | 69009 | 2014-03-24 11:54:53.538576 |
| 1 | 1 | Allée de la musique ... | ... | 69009 | 2014-03-24 11:54:53.538576 |

  - Contrainte entre XML_CLIENT et XML_ADRESSE

```
create table XML_ADRESS (
...
constraint xml_adress_client_fk foreign key ( id_client)
   references xml_client ( id ) ) ;
```

# Décomposition XML par shredding

- Pour palier le cas où le type défini dans la XSD n'est pas directement compatible avec le type de le colonne SQL correspondante
  - Soit ID de la table XML_CLIENT et ID_CLIENT de la table XML_ADRESS de type SQL int
  - XSD

```
<xs:attribute name="idClient" type="xs:string">
    <xs:annotation>
        <xs:appinfo>
            <db2-xdb:rowSetMapping>
                <db2-xdb:rowSet>XML_CLIENT</db2-xdb:rowSet>
                <db2-xdb:column>ID</db2-xdb:column>
                <db2-xdb:expression>CAST(INT($DECOMP_CONTENT) AS INT) + 10</db2-xdb:expression>
            </db2-xdb:rowSetMapping>
            <db2-xdb:rowSetMapping>
                <db2-xdb:rowSet>XML_ADRESS</db2-xdb:rowSet>
                <db2-xdb:column>ID_CLIENT</db2-xdb:column>
                <db2-xdb:expression>CAST(INT($DECOMP_CONTENT) AS INT) + 10</db2-xdb:expression>
            </db2-xdb:rowSetMapping>
        </xs:appinfo>
    </xs:annotation>
</xs:attribute>
```

# Décomposition XML par shredding

- Limites : ne sont pas gérés
  - <xs:any> ou <xs:anyAttribute>
    - Les éléments ou attributs correspondants ne sont pas pris en charge (sauf si fils d'éléments avec db2-xdb:contentHandling set to serializeSubtree ou stringValue)
  - xsi:type
    - Provoque une erreur de décomposition
  - Eléments récursifs
    - Ne sont pas décomposables
  - Seule l'insertion est gérée
    - Mises à jour et suppression dans les tables cibles non permises
  - ENTITY et NOTATION
    - Seul le nom est pris en compte

# Décomposition XML par XMLTABLE

- Prérequis
  - 7.1 et SI46631
- Fonctionnement
  - XMLTABLE retourne une table depuis l'évaluation d'une expression Xpath
  - XPath est un langage permettant d'accéder à une partie d'un flux XML
- Exemple XML

```
<?xml version="1.0"?>
<racine>
    <encyclopedie nom="Wikipedia" site="http://fr.wikipedia.org/">
        <article nom="XPath">
            <auteurs>
                <auteur>
                    <nom>Dupont</nom>
                </auteur>
                <auteur>
                    <nom>Dubois</nom>
                </auteur>
            </auteurs>
        </article>
    </encyclopedie>
</racine>
```

# Décomposition XML par XMLTABLE

| Expression | Résultat |
|---|---|
| / | Sélectionne l'ensemble du document |
| /root | sélectionne le nœud vide, puisqu'il n'y a pas d'élément "root" (mais "racine") |
| //article | sélectionne tous les éléments "article" du document où qu'ils soient |
| /racine/encyclopedie | sélectionne l'unique élément "encyclopedie" puisqu'il est ici le seul fils de "racine" portant ce nom |
| //article[@nom='XPath'] | sélectionne tous les éléments "article" du document où qu'ils soient, ayant un attribut "nom" dont la valeur est "XPath" |

- Ressources
  - http://fr.wikipedia.org/wiki/XPath
  - http://www.w3.org/TR/xpath/
  - http://www.w3.org/TR/xpath20/

# Décomposition XML par XMLTABLE

- Exemple XML
  - Table EMP
    - `CREATE TABLE EMP (DOC XML) ;`
  - Données de la table EMP : les deux lignes suivantes

```
<dept id="101">
    <employe id="901">
        <nom>
            <prenom>John</prenom>
            <famille>Doe</famille>
        </nom>
        <bureau>344</bureau>
        <salaire devise="EUR">35000</salaire>
    </employe>
    <employe id="902">
        <nom>
            <prenom>Peter</prenom>
            <famille>Pan</famille>
        </nom>
        <bureau>216</bureau>
        <tel>06.01.02.03.04</tel>
    </employe>
</dept>
```

```
<dept id="114">
    <employe id="903">
        <nom>
            <prenom>Mary</prenom>
            <famille>Jones</famille>
        </nom>
        <bureau>415</bureau>
        <tel>01.00.00.01.10</tel>
        <tel>06.22.33.44.55</tel>
        <salaire devise="EUR">47000</salaire>
    </employe>
</dept>
```

# Décomposition XML par XMLTABLE

- Par exemple

```
SELECT X.*
FROM emp,
   XMLTABLE ('$d/dept/employe' PASSING emp.doc AS "d"
              COLUMNS empID     INTEGER     PATH '@id',
                      prenom    VARCHAR(20) PATH 'nom/prenom',
                      patronyme VARCHAR(25) PATH 'nom/famille') AS X
```

| EMPID | PRENOM | PATRONYME |
|---|---|---|
| 901 | John | Doe |
| 902 | Peter | Pan |
| 903 | Mary | Jones |

  - Expression XPath de génération des lignes
    - `'$d/dept/employe'`
      - Indique que l'on génère une ligne par élément XML dept/employe trouvé dans chaque colonne XML de la table EMP
    - `PASSING emp.doc AS "d"`
      - Indique que `$d` référence la colonne DOC de la table EMP dans l'expression Xpath
  - Expressions de génération de colonnes

# Décomposition XML par XMLTABLE

- Cardinalités
  - Les données au format XML peuvent avoir un format variable
    - L'ensemble des informations n'est pas obligatoirement présent
      - La donnée générée est alors NULL
    - Il est possible de gérer des valeurs par défaut
      - Afin d'éviter les valeurs nulles
      - Exemple

```
salaire   INTEGER DEFAULT 0  PATH 'salaire‘
```

    - Il peut exister plusieurs occurrences
      - Il faut alors choisir le moyen de traiter

# Décomposition XML par XMLTABLE

- Limiter à 1 ligne résultat

```
XMLTABLE('$d/dept[@id="114"]/employe' PASSING doc AS "d"
```

- Seulement la 1ère valeur

```
tel         VARCHAR(15)  PATH 'tel[1]'
```

- Une valeur XML contenant l'ensemble des valeurs

```
tel         XML          PATH 'tel'
```

- Une colonne pour chaque valeur

```
tel1        VARCHAR(15)  PATH 'tel[1]',
tel2        VARCHAR(15)  PATH 'tel[2]'
```

# Décomposition XML par XMLTABLE

- Toutes les valeurs (autant de lignes)

```
XMLTABLE ('$d/dept/employe/tel' PASSING doc AS "d"
            COLUMNS   prenom     VARCHAR(20)  PATH '../nom/prenom',
                      patronyme  VARCHAR(25)  PATH '../nom/famille',
                      tel        VARCHAR(50)  PATH '.' ) AS X
```

- Gérer les valeurs inexistantes

```
XMLTABLE ('$d/dept/employe[fn:not(tel)]' PASSING doc AS "d"
```

- Numéroter les lignes résultat

```
 XMLTABLE ('$d/dept/employe' PASSING doc AS "d"
                  COLUMNS
                      seqno      FOR ORDINALITY,
                      empID      INTEGER     PATH '@id',
```

| SEQNO | EMPID | PRENOM | PATRONYME |
|---|---|---|---|
| 1 | 901 | John | Doe |
| 2 | 902 | Peter | Pan |
| 1 | 903 | Mary | Jones |

# Décomposition XML par XMLTABLE

- Il est également possible de gérer les espaces de noms
  - Sans tenir compte des namespaces

```
XMLTABLE ('$d/*:dept/*:employe' PASSING doc AS "d"
          COLUMNS
            empID     INTEGER       PATH '@*:id',
            prenom    VARCHAR(20)   PATH '*:nom/*:prenom',
            patronyme VARCHAR(25)   PATH '*:nom/*:famille' ) AS X
```

  - En spécifiant le namespace au niveau du document XML et à chaque colonne …
  - En spécifiant un namespace par défaut

```
XMLTABLE (XMLNAMESPACES(DEFAULT 'http://www.ibm.com/xmltable'),
                  '$d/dept/employe' PASSING doc AS "d"
```

  - En spécifiant des préfixes …
- Les fichiers XML depuis l'IFS sont supportés
  - **GET_XML_FILE**

# Décomposition XML par XMLTABLE

- XMLTABLE vs XDBDECOMPXML
  - XMLTABLE est beaucoup plus souple que XDBDECOMPXML
    - Ne nécessite pas les XSD modifiées et cataloguées dans DB2
    - Ne produit pas un résultat stocké directement dans des tables DB2
  - Cela permet d'extraire des données depuis du XML, soit stocké en BD soit sur l'IFS, par une requête SQL
    - Utilisable en SQL embarqué
    - Eventuellement construite dynamiquement pour des flux dont la structure n'est pas entièrement connue à l'avance

# Transformation XSL

- DB2 intègre un moteur de transformation XSL
  - Le langage XSL permet de décrire des transformations à appliquer sur des données XML
    - XSL : XML Style Sheet (feuille de style XML)
  - Pour générer des données dans un autre format
    - Format à base de balises : XML, HTML … Usage prévu de la norme
  - Ressources
    - http://fr.wikipedia.org/wiki/Extensible_Stylesheet_Language_Transformations
    - http://www.w3.org/TR/xslt
- Nécessite
  - IBM XML Toolkit for i 5733-XT2
  - Java 5770-JV1
  - Portable App Solutions Environment 5770-SS1

# Transformation XSL

- Exemple

```
SELECT XSLTRANSFORM (XML_DOC USING XSL_DOC AS CLOB(1M))
FROM XML_TAB;
```

  - XML_DOC
    - Document XML à transformer (chaine de caractères ou document XML)
  - XSL_DOC
    - Document XSL (chaine de caractères ou document XML)
  - WITH xsl-parameters
    - Admet des paramètres (chaine de caractères ou document XML)

```
<params xmlns="http://www.ibm.com/XSLTransformParameters">
  <param name="parm1" value="valeur1"/>
  <param name="parm2">valeur2</param>
</params>
```

# Conclusion

- DB2 intègre totalement XML
  - Dans le respect des standards de l'industrie
- Cela nous permet de disposer de fonctions XML
  - Dans tous nos développements !
    - SQL embarqué, script SQL, QM …
  - De façon uniforme
  - Avec support IBM, sans produit supplémentaire

# Nous contacter

- Par mail
  - nbonnet@gaia.fr
  - contact@gaia.fr
- Nos sites
  - www.gaia.fr
  - www.know400.fr
  - www.as400.fr