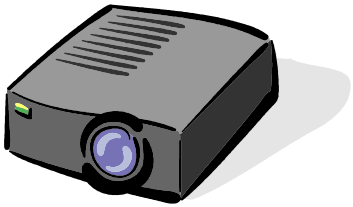
Université IBM i – 2017

17 et 18 mai 2017 – IBM Client Center, Bois-Colombes

VOLUBIS
DEVENEZ UN EXPERT

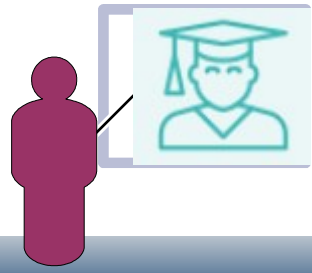


Volubis.fr

Conseil et formation sur OS/400, I5/OS puis IBM *i* depuis 1994 !

Dans nos locaux, vos locaux ou par Internet

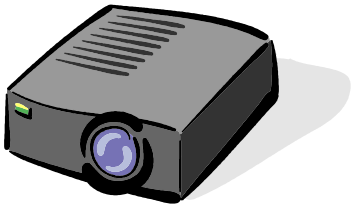
Christian Massé - cmasse@volubis.fr



Université IBM i – 2017

17 et 18 mai 2017 – IBM Client Center, Bois-Colombes

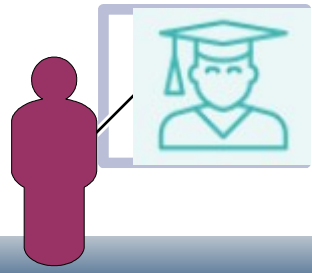
VOLUBIS
DEVENEZ UN EXPERT



Volubis.fr

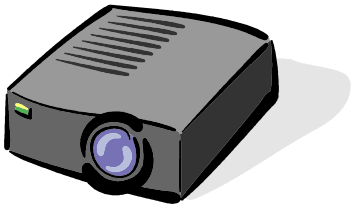
Base de connaissance depuis 1995 (plus de 500 cours)

Cours en ligne (accessibles en mode « replay »)

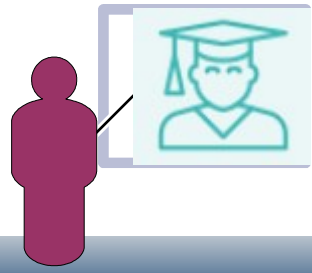


Université IBM i – 2017

17 et 18 mai 2017 – IBM Client Center, Bois-Colombes



Session 34 : XML vs JSON



XML vs JSON

- Format XML

  - Les règles du jeu XML

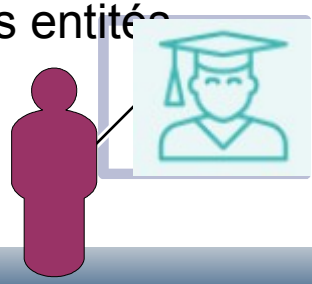
  - Elles sont extrêmement simples. Les informations doivent être :

  - soit encadrées par des balises ouvrantes(ex. <LIVRE>) et fermantes (ex. </LIVRE>) (contrairement à HTML où ses ces dernières n'étaient pas toujours obligatoires). On parle alors **d'éléments**. Les éléments doivent s'imbriquer proprement les uns dans les autres : aucun chevauchement n'est autorisé.

    Les éléments vides sont permis, selon le format <ELEMENTVIDE/>.

  - soit incluses à l'intérieur même des balises : on parle alors **d'attributs**.

    Exemple : <LIVRE SUJET="XML">. Ici l'attribut SUJET de l'élément LIVRE a la valeur "XML" . En XML, contrairement à HTML, les valeurs des entités doivent toujours être encadrées par des guillemets.



XML vs JSON

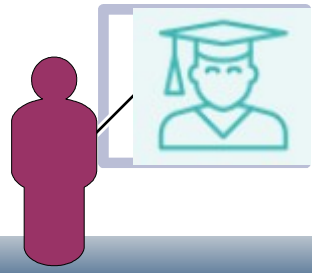
- Format XML

  - Les règles du jeu XML

  - Soit encore définies sous forme d'entités. Les entités sont des abréviations.

    Par ex; si "Extensible Markup Language" est déclaré comme une entité associée à la notation "xml"; cette chaîne de caractères pourra être abrégée en "&xml;" dans tout le fichier XML.

  - Les caractères < , > , & et " doivent être remplacés par les entités **< , > , &** et **&quote;** dans les valeurs de type texte.



XML vs JSON

- Format XML

  - le document peut enfin contenir :

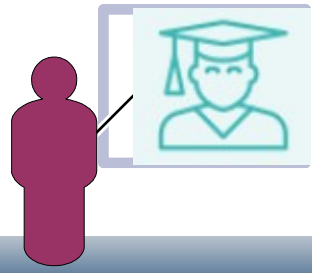
  - des commentaires sous la forme <!-- le commentaire →

  - des processing instructions (PI) c'est à dire des instructions destinées à une application particulière sous la forme
    <?nom-application instructions-spécifiques?>

    par exemple, une "processing instruction" destinées à XML lui même commence souvent les documents XML.

    <?xml version="1.0" encoding="ISO-8859-1" ?>

    (version de XML et jeu de caractères des données, ici "latin-1", c.a.d le nôtre)



XML vs JSON

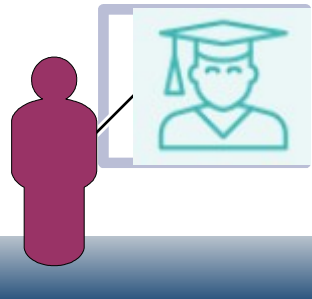
- Format XML

  - La structure arborescente du document XML (intitulé des balises, imbrications des balises, caractère obligatoire ou facultatif des balises et de leur ordre de succession) peut être déclarée formellement dans le corps du document XML ou dans un fichier séparé.

  - Avant DTD
  - Aujourd'hui schéma **XSD** (lui même en XML)

  Lorsqu'un document XML possède une DTD ou un schéma associé et le respecte, on dit qu'il est **valide**.

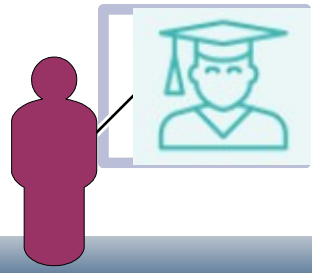
  Lorsqu'il respecte seulement les règles de la grammaire XML (balises fermées, correctement imbriquées) on dit qu'il est **bien formé**.



XML vs JSON

- Format XML

```
<producteurs>
  <producteur>
    <numero>45</numero>
   <commune>Reims</commune>
    <appellation>13</appellation>
  </producteur>
</producteurs>
```



XML vs JSON

- Format JSON

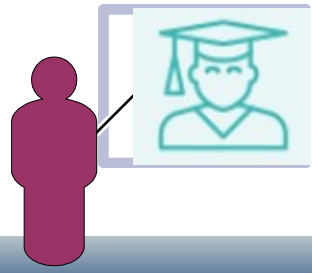
JSON (JavaScript Object Notation) est un format de données textuelles dérivé de la notation des objets du langage JavaScript

Les éléments sont marqués par **{** et **}**

le nom de l'élément est donné au début de l'élément, pas à la fin

Les tableaux d'éléments sont notés par **[** et **]**

Les commentaires ne sont pas prévus par la RFC

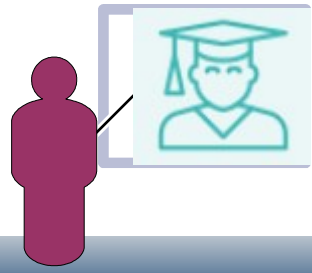


XML vs JSON

- Format JSON

ce qui s'écrit comme cela en XML		s'écrit comme cela en JSON

<producteurs>
<producteur>
<numero>45</numero>
<commune>Reims</commune>
<appellation>13</appellation>
</producteur>
</producteurs>
	->	{
		"producteurs": {
		"producteur": {
		"numero":45,
		"commune":"Reims",
		"appellation":13
		}
		}
		}



XML vs JSON

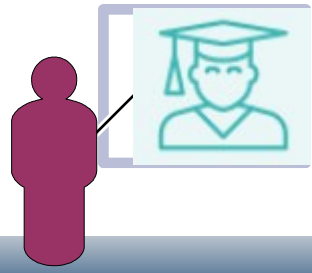
- Vous trouverez ici

  http://www.mcpressonline.com/programming-other/general/techtip-json-and-xml-conversion-in-db2-for-i

  deux fonctions bien pratiques Xml2Json / Json2Xml

```
1 values af4test.xml2json('<producteur>
2   <nom>Bruno Cormerais</nom><appellation>muscadet</appellation>
3  </producteur>')
```

```
00001
{"producteur":{"appellation":"muscadet","nom":"Bruno Cormerais"}}
```



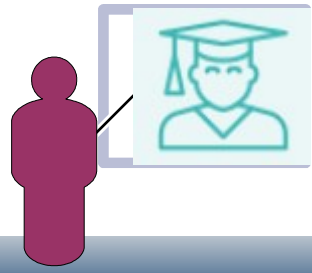
XML vs JSON

- Format JSON

```xml
<PO>
 <id>103</id>
 <orderDate>2014-06-20</orderDate>
 <customer>
   <cid>888</cid>
 </customer>
 <items>
  <item>
   <partNum>872-AA</partNum>
   <shipDate>2014-06-21</shipDate>
   <productName>Lawnmower</productName>
   <USPrice>749.99</USPrice>
   <quantity>1</quantity>
 </item>
 <item>
  <partNum>837-CM</partNum>
  <productName>Digital Camera</productName>
  <USPrice>199.99</USPrice>
  <quantity>2</quantity>
  <comment>2014-06-22</comment>
  </item>
 </items>
</PO>
```

```json
{
 "PO":{
 "id": 103,
 "orderDate": "2014-06-20",
 "customer": {"@cid": 888},
 "items": {
   "item": [
            { "partNum": "872-AA",
              "productName": "Lawnmower",
              "quantity": 1,
              "USPrice": 749.99,
              "shipDate": "2014-06-21"
            },
            { "partNum": "837-CM",
              "productName": "Digital Camera",
              "quantity": 2,
              "USPrice": 199.99,
              "comment": "2014-06-22"
            }
          ]
        }
    }
}
```

Tableau



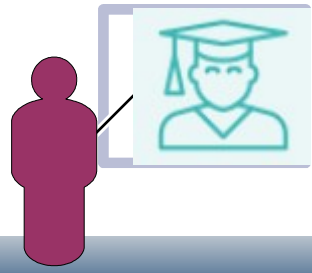
XML vs JSON

- Format BSON

  Http://bsonspec.org

BSON *[bee · sahn]*, short for Binary JSON, is a binary-encoded serialization of JSON-like documents. Like JSON, BSON supports the embedding of documents and arrays within other documents and arrays. BSON also contains extensions that allow representation of data types that are not part of the JSON spec. For example, BSON has a Date type and a BinData type.

BSON is designed to be traversed easily. This is a vital property in its role as the primary data representation for MongoDB.



XML vs JSON

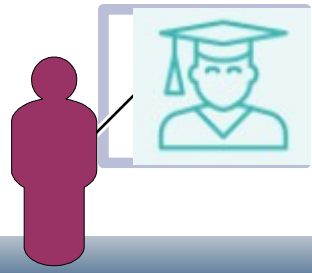
- Format BSON

  DB2 for I propose un accès noSQL (développé avec MongoDB)

  DB2nosql -- accès "nosql" en mode ligne de commandes

  la première fois utilisez impérativement l'option -setup enable

cela créé une table SYSJSON_INDEX



XML vs JSON

- Format BSON

- db.sqlUpdate

  passe un ordre SQL (par exemple create schema)

- use schema

  défini le schéma (bibliothèque) en cours

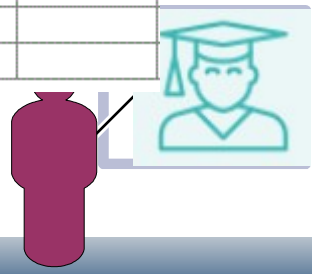
- db.createCollection

  créé une collection JSON (soit une table dans le schéma en cours)

JSONBIB.JSONCOL - As400.volubis.intra(As400)

Table | Colonnes | Contraintes de clé | Contraintes de clé associée | Contraintes de vérification | Requête matérialisée | Partitionnement

Nom de colonne	Nom de système	Type de données	Longu...	Val...	Implicite...	Valeur par défaut	Texte	CCSID
ID	ID	CHARACTER	12	Non		Aucune valeur p...		65535
DATA	DATA	BLOB	16777...	Oui		Valeur indéfinie		
ROWCREATED	ROWCREATED	TIMESTAMP	12	Non		Horodatage en ...		



XML vs JSON

- Format BSON

- db.collection.insert

- insert des données JSON dans une collection

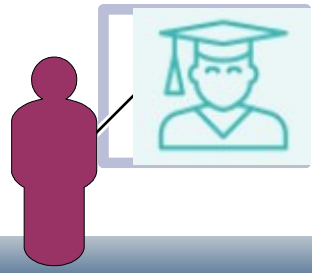
```
db.JSONCOL.insert({nom:"Miss Terre",appellation:"Muscadet",commune:"La sénéchalière"})
db.JSONCOL.insert({nom:"Miss Terre",appellation:"Muscadet",commune:"La s?chalinre"})
CDJSN1000I La commande a abouti.
nosql>
db.JSONCOL.insert({nom:"Montus",appellation:"Madiran",commune:"Maumusson"})
db.JSONCOL.insert({nom:"Montus",appellation:"Madiran",commune:"Maumusson"})
CDJSN1000I La commande a abouti.
nosql>
```

Les données sont au format BSON

Contenu de JSONBIB.JSONCOL - As400.volubis.intra(As400)

	ID	DATA	ROWCREATED
1	55C0C3DD54C205A6C6444B23	0350000000026E6F6D000B0000004D69737320546572726500026170706 56C6C6174696F6E00090000004D75736361646574000 2636F6D6D756E6500100000004C612073EFBFBD6368616C696E72650000	2015-08-04 15:53:33.352972527832
2	55C0C43554C205A6C6444B24	0345000000026E6F6D00070000004D6F6E74757300026170706 56C6C6174696F6E00080000004D61646972616E0002636F6D6D756E65000A0000004D61756D7573736F6E0000	2015-08-04 15:55:01.620842320800

(voir les fonctions SQL Bson2Json et Json2Bson)

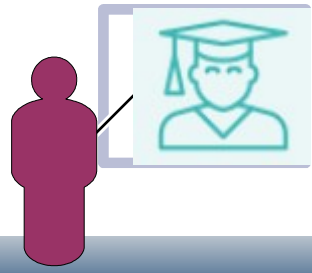


XML vs JSON

- Format BSON

- de nombreuses commandes sont disponibles ensuite, comme **find**

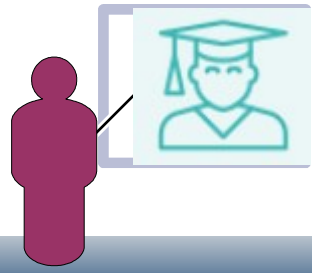
```
db.JSONCOL.find()
nosql>Ligne 1 :
nosql> {
"_id":{"$oid":"55c0c3dd54c205a6c6444b23"},
"nom":"Miss Terre",
"appellation":"Muscadet",
"commune":"La s?chalinre"
}
nosql>Ligne 2 :
nosql> {
"_id":{"$oid":"55c0c43554c205a6c6444b24"},
"nom":"Montus",
"appellation":"Madiran",
"commune":"Maumusson"
}
nosql>CDJSN1206I "2" lignes(s) renvoyÃ©e(s) en "31" millisecondes.
nosql>
```



XML vs JSON

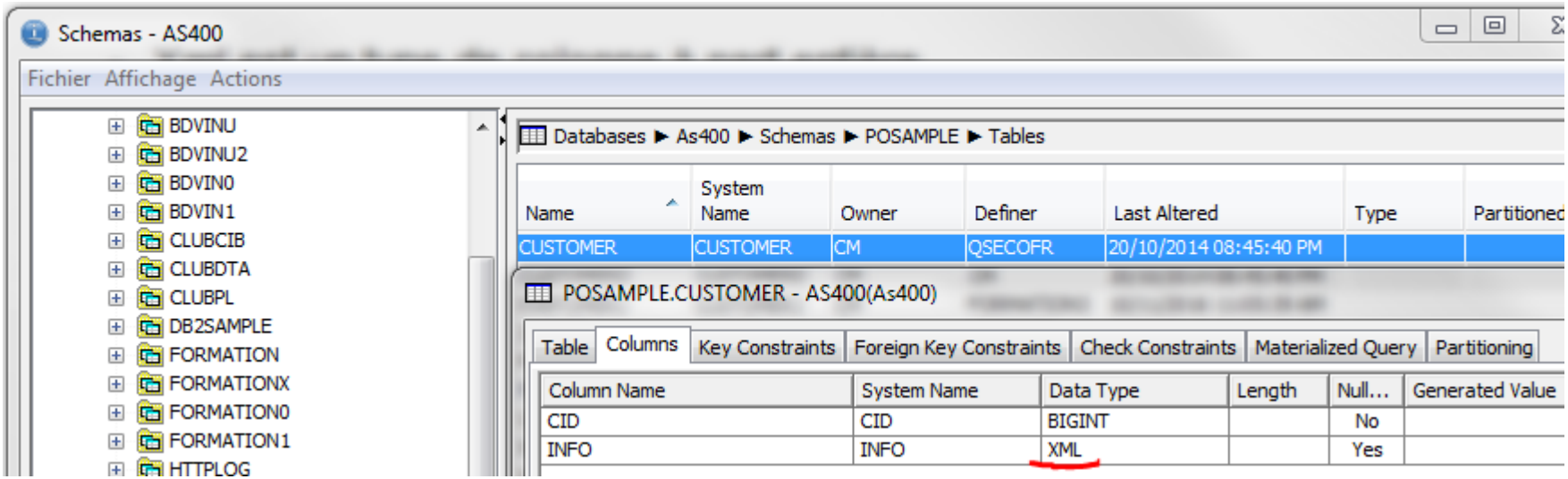
- Mais JSON comme XML peut être utilisé directement par le langage SQL
  - Stocké dans une table DB2
  - Vérifiez avant d'être stocké
  - Manipulé par un simple Select (à l'aide de fonctions table)
- Les deux formats peuvent être générés par le serveur de web services (Liberty), quand il expose vos programmes historiques

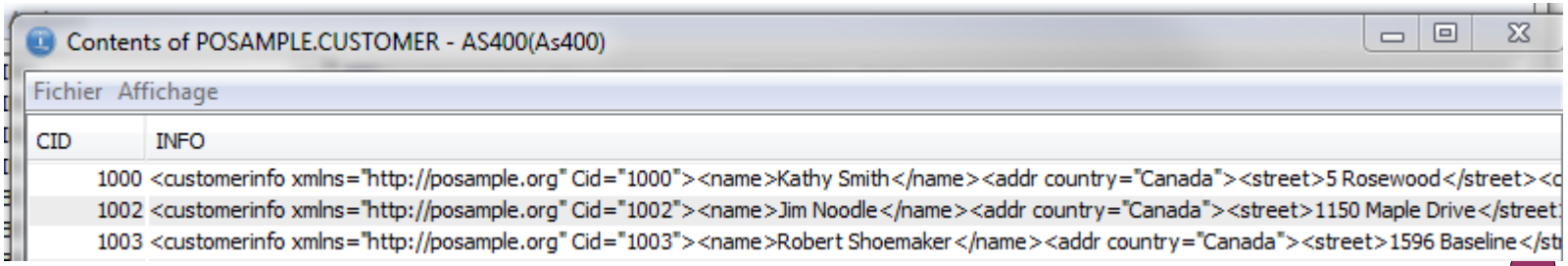
Voyons cela...

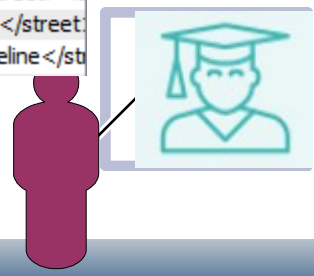


XML vs JSON

- Xml est un type de colonne à part entière pour DB2

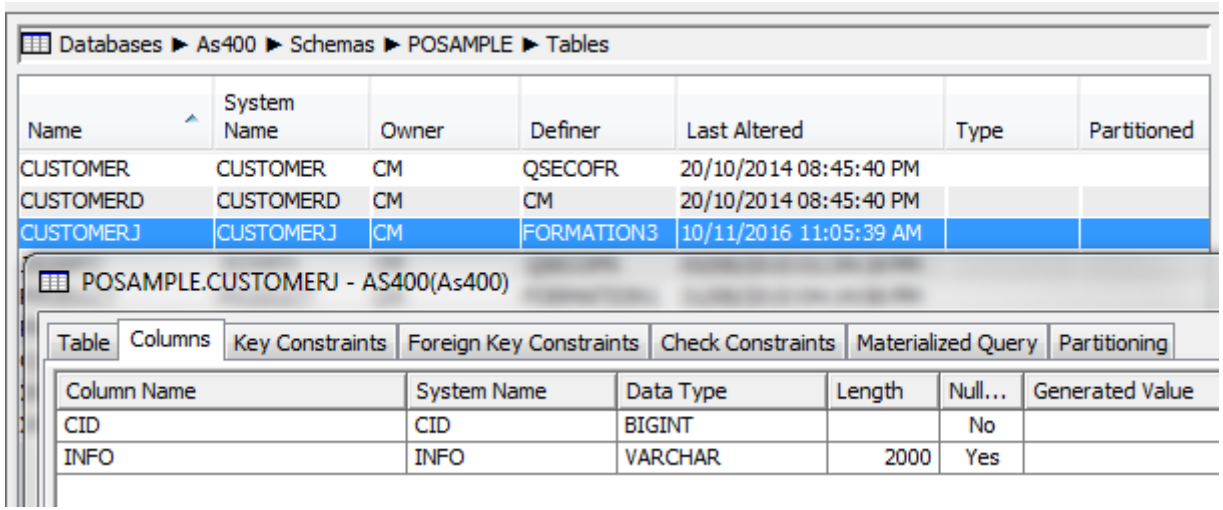




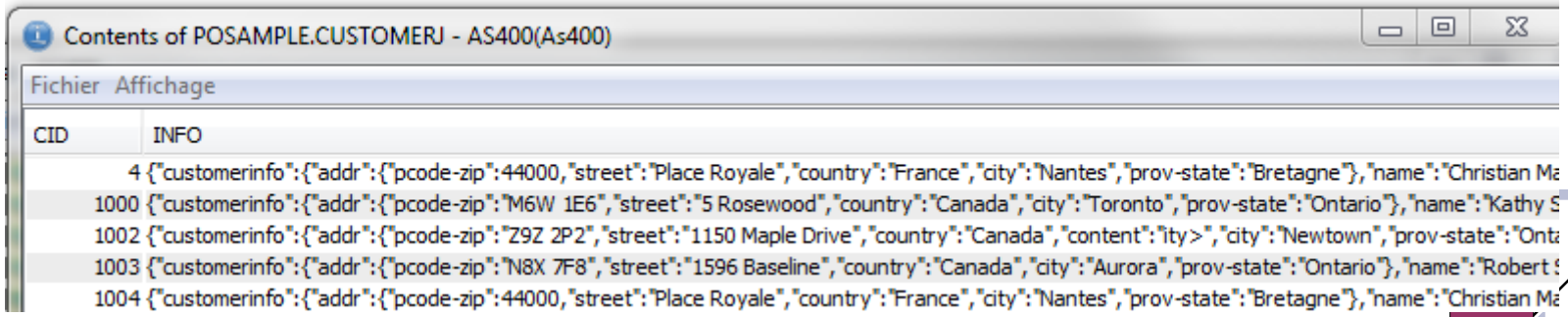


XML vs JSON

- Vous pouvez stocker du JSON dans du VARCHAR



Format JSON ou BSON



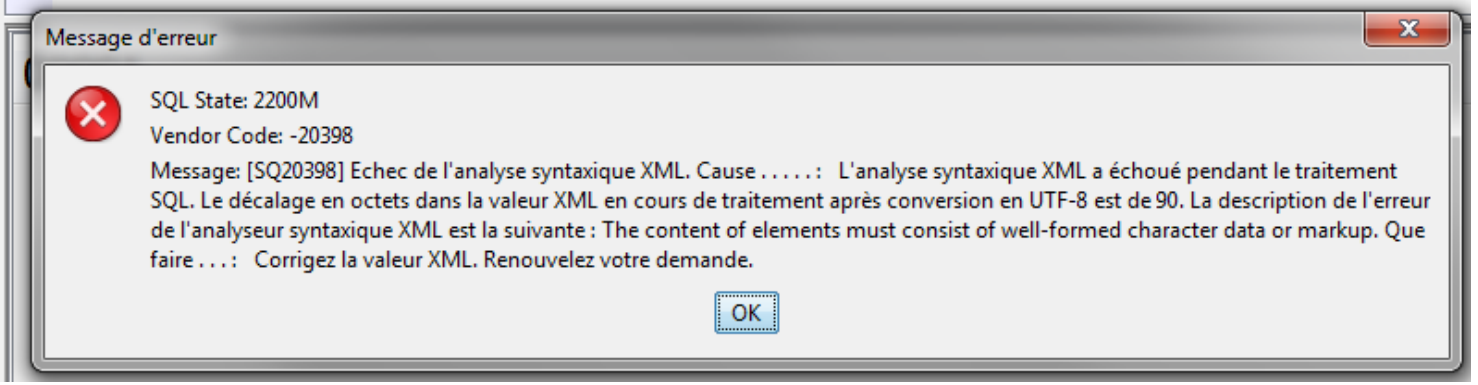
XML vs JSON

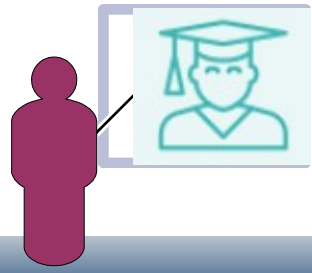
- Xml peut être vérifié par XMLPARSE

```
1 values xmlparse(DOCUMENT
2  '<producteur>
3   <nom>Bruno Cormerais</nom><appellation>muscadet</appellation>
4  </producteur>')
```

```
00001
<producteur><nom>Bruno Cormerais</nom><appellation>muscad...
```

```
1 values xmlparse(DOCUMENT
2  '<producteur>
3   <nom>Bruno Cormerais</nom><appellation>muscadet</appellation>
4  <producteur>')
```





XML vs JSON

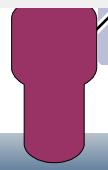
- JSON peut être vérifié par le prédicat IS JSON (TR2/TR6)

```
1 values CASE
2 when '{"producteur":{"appellation":"muscadet","nom":"Bruno Cormerais"}}' IS JSON then 1
3 else 0
4 END
```

00001
1

```
1 values CASE
2 when '<"producteur":{"appellation":"muscadet","nom":"Bruno Cormerais"}>' IS JSON then 1
3 else 0
4 END
```

00001
0



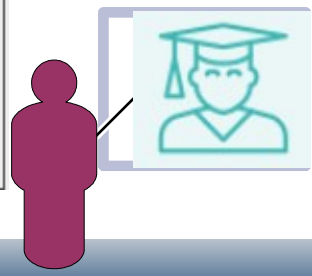
XML vs JSON

- Xml est vérifié lors de l'insertion

```
INSERT INTO POSAMPLE/CUSTOMER
 VALUES( 1004 , GET_XML_FILE('/temp/client04.xml') )
```

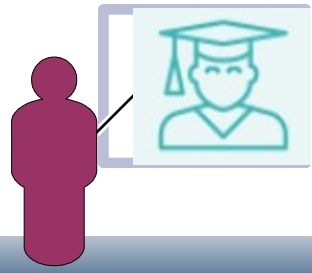
le document doit être bien formé, sinon vous recevrez SQ20398

```
                    Complément d'informations sur message

 ID message . . . . . :   SQ20398


 Message . . . . :   Echec de l'analyse syntaxique XML.

 Cause . . . . . :   L'analyse syntaxique XML a échoué pendant le traitement
   SQL. Le décalage en octets dans la valeur XML en cours de traitement après
   conversion en UTF-8 est de 203. La description de l'erreur de l'analyseur
   syntaxique XML est la suivante : Element type "addr" must be followed by
   either attribute specifications, ">" or "/>".
 Que faire . . . :   Corrigez la valeur XML. Renouvelez ensuite votre demande.
```



XML vs JSON

- Xml est vérifié lors de l'insertion
- Pour JSON vous devez faire un Trigger BEFORE INSERT avec IS JSON
- Vous pouvez associer un schéma (xsd) à une colonne XML
  et utiliser XMLVALIDATE
- Vous pouvez vérifier la présence d'un éléments JSON par la nouvelle fonction (TR2/TR6) JSON_EXISTS



XML vs JSON

- XMLVALIDATE

- Enregistrement du schéma

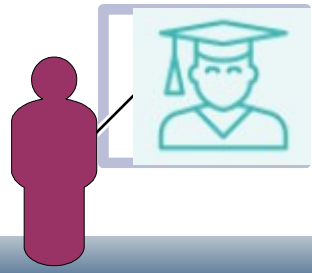
  CALL SYSPROC.XSR_REGISTER('POSAMPLE', 'CUSTOMER',
  'http://posample.org', GET_XML_FILE('schema.xsd'), null);

  CALL SYSPROC.XSR_COMPLETE('POSAMPLE', 'CUSTOMER', null, 0);

Vérifiez en affichant le contenu de XSROBJECTS

```
SELECT XSROBJECTSCHEMA, XSROBJECTNAME FROM QSYS2.XSROBJECTS ;
```

XSROBJECTSCHEMA	XSROBJECTNAME
POSAMPLE	CUSTOMER



XML vs JSON

- XMLVALIDATE

- Essayons d'insérer une donnée ne respectant pas les règles.

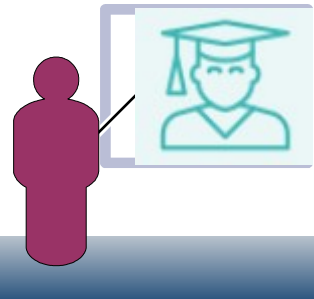
- -> ici nous ne fournissons pas d'élément <addr> alors que ce dernier est déclaré obligatoire (<xs:element name="addr" minOccurs="1")

```
INSERT INTO POSAMPLE.Customer(Cid, Info) VALUES (1004,
XMLVALIDATE (XMLPARSE (DOCUMENT
         '<customerinfo xmlns="http://posample.org" Cid="1003">
           <name>Robert Shoemaker</name>
           <phone type="work">905-555-7258</phone>
           <phone type="home">416-555-2937</phone>
           <phone type="cell">905-555-8743</phone>
           <phone type="cottage">613-555-3278</phone>
         </customerinfo>' PRESERVE WHITESPACE )
  ACCORDING TO XMLSCHEMA ID posample.customer ));

--------------------------------------------------------------------------------
Etat SQL : 2201R
Code fournisseur : -20399
Message : [SQ20399] Echec de l'analyse syntaxique ou de la validation XML.
 Cause . . . . . :   L'analyse syntaxique XML a échoué pendant la validation.
Le décalage en octets dans la valeur XML en cours de traitement après conversion en UTF-8 est de 109.
La description de l'erreur de l'analyseur syntaxique XML est la suivante : cvc-complex-type.2.4.a:
Expecting element with local name "addr" but saw "phone".
 Que faire . . . :   Corrigez l'incident lié au document de l'instance XML.
 Relancez XMLVALIDATE ou XDBDECOMPXML.
```

XML vs JSON

- JSON_EXISTS

```
>>-JSON_EXISTS--(--json-expression--+-------------+-,------>
                                   +-FORMAT JSON-+
                                   '-FORMAT BSON-'

>--sql-json-path-expression--+---------------+------------->
                             '-AS--path-name-'

   .-FALSE ON ERROR--------.
>--+-----------------------+--)---------------------------><
   '-+-TRUE----+--ON ERROR-'
     +-UNKNOWN-+
     '-ERROR---'
```



XML vs JSON

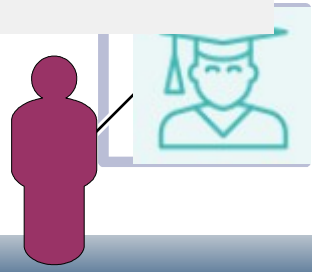
- JSON_EXISTS

```
1 values CASE
2 when JSON_EXISTS('{"producteur":{"appellation":"muscadet","nom":"Bruno Cormerais"}}' ,
3   'lax $.producteur.nom') then 1
4 else 0 END
```

00001
1

```
1 values CASE
2 when JSON_EXISTS('{"producteur":{"appellation":"muscadet","nom":"Bruno Cormerais"}}' ,
3   'lax $.producteur.tel') then 1
4 else 0 END
```

00001
0



XML vs JSON

- Les schémas XSD peuvent être utilisés pour faire de la décomposition

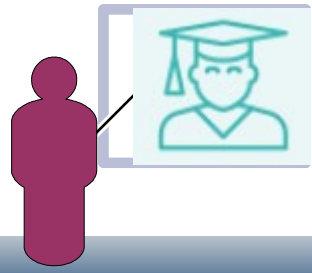
  (passage de XML à DB2)

  Il faut ajouter l'espace de nommage db2-xdb

```
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
    xmlns:db2-xdb="http://www.ibm.com/xmlns/prod/db2/xdb1" >
```

  Indiquer le schéma (bibliothèque) par défaut

```
   <xs:annotation>
      <xs:appinfo>
          <db2-xdb:defaultSQLSchema>POSAMPLE</db2-xdb:defaultSQLSchema>
      </xs:appinfo>
   </xs:annotation>
```



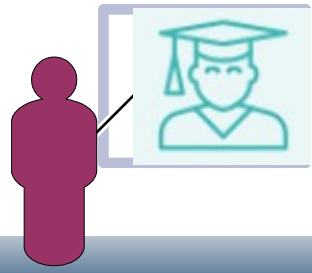
XML vs JSON

- Les schémas XSD peuvent être utilisés pour faire de la décomposition

  (passage de XML à DB2)

  Associer à chaque élément ou attribut concerné un nom de table et de colonne

```
<xs:complexType>
 <xs:sequence>
  <xs:element name="name" type="xs:string" minOccurs="1" db2-xdb:rowSet="CUSTOMERD" db2-xdb:column="NOM" />
  <xs:element name="addr" minOccurs="1" maxOccurs="unbounded">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="street" type="xs:string" minOccurs="1" db2-xdb:rowSet="CUSTOMERD" db2-xdb:column="RUE" />
    <xs:element name="city" type="xs:string" minOccurs="1" db2-xdb:rowSet="CUSTOMERD" db2-xdb:column="VILLE" />
    <xs:element name="prov-state" type="xs:string" minOccurs="1" db2-xdb:rowSet="CUSTOMERD" db2-xdb:column="ETAT" />
    <xs:element name="pcode-zip" type="xs:string" minOccurs="1" />
   </xs:sequence>
   <xs:attribute name="country" type="xs:string" db2-xdb:rowSet="CUSTOMERD" db2-xdb:column="PAYS" />
  </xs:complexType>
  </xs:element>
  <xs:element name="phone" nillable="true" minOccurs="0" maxOccurs="1"
                                                    db2-xdb:rowSet="CUSTOMERD" db2-xdb:column="TEL">
   <xs:complexType>
```



XML vs JSON

- Les schémas XSD peuvent être utilisés pour faire de la décomposition

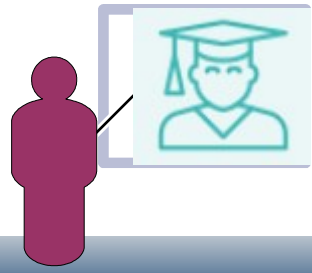
  enregistrer le schéma par XSR_REGISTER (comme vu plus haut)

  puis

  CALL SYSPROC.XSR_COMPLETE('POSAMPLE', 'CUSTOMERD', null, **1**);

  Enfin appeler la procédure XDBDECOMPXML

  ici nous souhaitons traiter la table CUSTOMER dont **chaque ligne** contient du XML, il nous faut donc lire par un curseur et appeler XDBDECOMPXML à chaque ligne lue



XML vs JSON

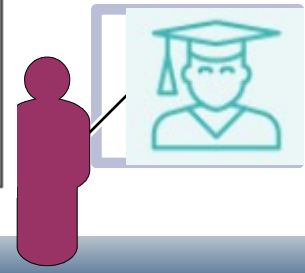
- Les schémas XSD peuvent être utilisés pour faire de la décomposition

```
CREATE PROCEDURE POSAMPLE.DECOMP ( )
       LANGUAGE SQL
       SET OPTION DBGVIEW = *SOURCE
BEGIN
  DECLARE CONTENT BLOB ( 2G ) ;
  DECLARE WEOF INTEGER DEFAULT 0;

  DECLARE not_found CONDITION FOR '02000';

  DECLARE c1 CURSOR FOR
         SELECT XMLSERIALIZE(info as BLOB(2G) ) FROM POSAMPLE.CUSTOMER;

  DECLARE CONTINUE HANDLER FOR not_found SET wEOF = 1;

  OPEN c1;
  FETCH c1 INTO content;
  WHILE wEOF = 0 DO

      CALL XDBDECOMPXML ('POSAMPLE' , 'CUSTOMERD' , content, NULL);
      FETCH c1 INTO content;
  END WHILE;
  CLOSE c1;
END  ;
```

CUSTOMERD | CUSTOMERD | Non | ROOT | ROOT | 12/08/10 10:44:39

Contenu de POSAMPLE.CUSTOMERD - I5test(D6013fc2)

	NOM	RUE	VILLE	ETAT	PAYS	TEL
1	Kathy Smith	5 Rosewood	Toronto	Ontario	Canada	416-555-1358
2	Jim Noodle	1150 Maple Drive	Newtown	Ontario	Canada	905-555-7258
3	Robert Shoemaker	1596 Baseline	Aurora	Ontario	Canada	905-555-2937



XML vs JSON

- Mais nous aurions pu faire le job avec XMLTABLE

XMLTABLE permet de parser un flux XML

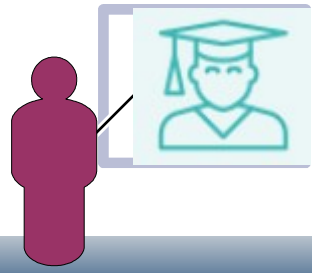
Syntaxe

XMLTABLE ('$alias/élément-à-traiter' passing « flux-XML » as "alias"

COLUMNS
Nom type-sql PATH 'syntaxeXpath',
.../...
…/...
)
AS aliasDeTable

Il y aura autant de lignes résultat qu'il existe de « élément-à-traiter »

avec autant de colonnes que demandé dans la clause COLUMNS



XML vs JSON

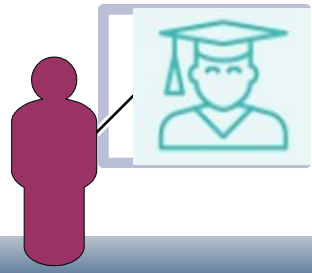
- Mais nous aurions pu faire le job avec XMLTABLE

XMLTABLE permet de parser un flux XML

```
SELECT X.NOM, X.RUE, X.VILLE  FROM Customer,
  XMLTABLE ('$c/customerinfo' passing INFO as "c"
     COLUMNS
     NOM       CHAR(30)    PATH 'name',
     RUE VARCHAR(25) PATH 'addr/street',
     VILLE    VARCHAR(25)   PATH 'addr/city'    )
AS X
```

Affiche

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7
NOM                            RUE                            VILLE
Kathy Smith                    5 Rosewood                     Toronto
Jim Noodle                     1150 Maple Drive               Newtown
Robert Shoemaker               1596 Baseline                  Aurora
********  Fin de données  ********
```



XML vs JSON

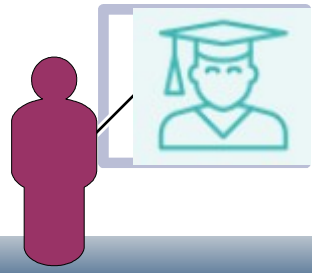
- Mais nous aurions pu faire le job avec XMLTABLE

XMLTABLE avec un fichier de l'IFS

```
SELECT * FROM
    XMLTABLE(
      '$V/vins/vin' PASSING
       XMLPARSE(DOCUMENT GET_XML_FILE('/xml/vins.xml')) AS "V"
 COLUMNS
  nom    CHAR(30) PATH 'nom',
  cepage VARCHAR(50) PATH 'cepage1')
AS XML_IFS
```

•GET_XML_FILE lit un fichier de l'IFS

•XMLPARSE vérifie la validité du flux et fabrique une Colonne XML pour DB2



XML vs JSON

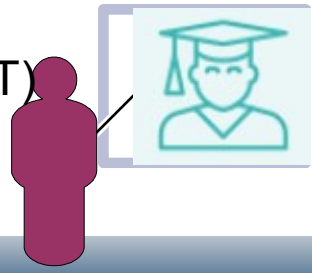
- JSON_TABLE arrive EN TR1(7.3) / TR5 (7.2)

```
JSON_TABLE(
        JSON_SOURCE
        JSON_PATH

       COLUMNS
         nom type PATH 'json_path',
              ...
 ) as AliasDeTable
```

sur une syntaxe « assez » proche de XMLTABLE

•JSON_Source

- Une colonne
- Le résultat de GET_CLOB_FROM_FILE
- Le résultat de HttpGetCLob (comme XMLTABLE → web services REST)



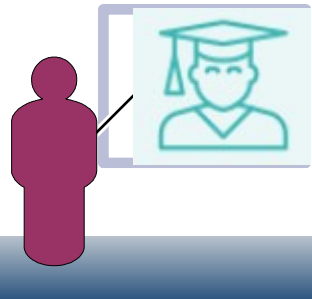
XML vs JSON

- JSON_TABLE arrive EN TR1(7.3) / TR5 (7.2)

```
JSON_TABLE(
        JSON_SOURCE
        JSON_PATH

       COLUMNS
         nom type PATH 'json_path',
              ...
 ) as AliasDeTable
```

•JSON_PATH

$	objet en cours
.	élément dans l'objet en cours
[ ]	élément dans un tableau
un_nom	la valeur de l'élément



XML vs JSON

- JSON_TABLE arrive EN TR1(7.3) / TR5 (7.2)

•COLUMNS

- Comme avec XMLTABLE il s'agit de "fabriquer" des pseudo-colonnes base de données, indiquez :

- un nom de colonne
- un type : CHAR(x) , DEC(x , y), etc...
- PATH

une règle d'utilisation

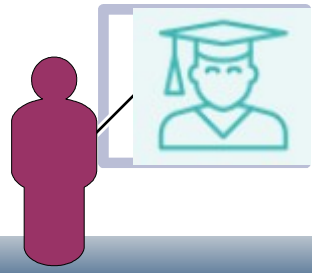
lax, utilisation souple

on peut faire référence à un élément, même quand c'est un tableau, l'itération est alors automatique
on peut faire référence à un tableau, même quand c'est un élément

les cas impossibles génèrent valeur nulle

strict, utilisation stricte, les cas précédents génèrent une erreur

L'élément JSON dont il faut extraire la valeur

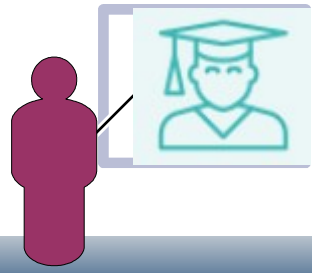


XML vs JSON

- JSON_TABLE arrive EN TR1(7.3) / TR5 (7.2)

•COLUMNS

- Des options :
  - NULL ON EMPTY
    retourne valeur nulle sur un élément manquant
  - ERROR ON EMPTY
    retourne une erreur sur un élément manquant
  - DEFAULT <une-valeur> ON EMPTY
    retourne une valeur par défaut sur un élément manquant
  - ERROR ON ERROR
    retourne une erreur en cas d'erreur (SQL16410)
  - DEFAULT <une-valeur> ON ERROR
    retourne une valeur par défaut suite à une erreur



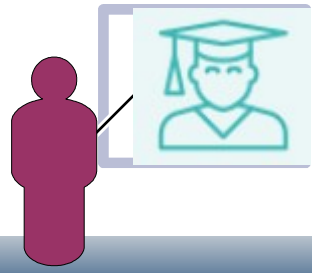
XML vs JSON

- JSON_TABLE arrive EN TR1(7.3) / TR5 (7.2)

•Exemple

```
1  CREATE VARIABLE JSON_VAR VARCHAR(2000);
2  SET JSON_VAR='{
3          "id" : 901,
4          "name" : { "first":"John", "last":"Doe" },
5          "phones" : [{"type":"home", "number":"555-3762"},
6                      {"type":"work", "number":"555-7252"}]
7          }';
8
9   SELECT * FROM JSON_TABLE(JSON_VAR,
10    '$'
11    COLUMNS(
12        id VARCHAR(10) PATH 'lax $.id',
13        first VARCHAR(10) PATH 'lax $.name.first',
14        last VARCHAR(10) PATH 'lax $.name.last' )
15 ) as t;
```

ID	FIRST	LAST
901	John	Doe



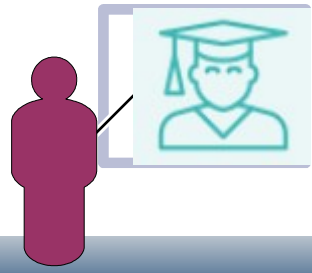
XML vs JSON

- JSON_TABLE arrive EN TR1(7.3) / TR5 (7.2)

•NULL ON EMPTY

```
 9  SELECT * FROM JSON_TABLE(JSON_VAR,
10   '$'
11   COLUMNS(
12       id VARCHAR(10) PATH '$.idd' NULL on EMPTY,
13       first VARCHAR(10) PATH 'lax $.name.first',
14       last VARCHAR(10) PATH 'lax $.name.last',
15       tel CHAR(12) PATH 'lax $.phones[0].number')
16       ERROR on ERROR
```

ID	FIRST	LAST	TEL
-	John	Doe	555-3762



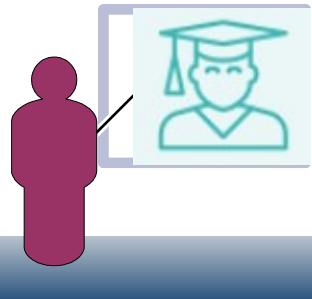
XML vs JSON

- JSON_TABLE arrive EN TR1(7.3) / TR5 (7.2)

•DEFAULT ON EMPTY

```
 9  SELECT * FROM JSON_TABLE(JSON_VAR,
10    '$'
11    COLUMNS(
12        id VARCHAR(10) PATH '$.idd' DEFAULT 'non trouvé'  on EMPTY,
13        first VARCHAR(10) PATH 'lax $.name.first',
14        last VARCHAR(10) PATH 'lax $.name.last',
15        tel CHAR(12) PATH 'lax $.phones[0].number')
16        ERROR on ERROR
```

ID	FIRST	LAST	TEL
non trouvé	John	Doe	555-3762



XML vs JSON

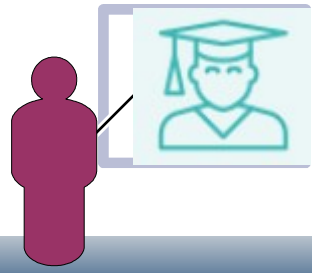
- JSON_TABLE arrive EN TR1(7.3) / TR5 (7.2)

- lax vs strict

  Avec lax, un élément manquant ne provoque pas d'erreur

```
 9  SELECT * FROM JSON_TABLE(JSON_VAR,
10    'lax $.inconnu'
11    COLUMNS(
12        id VARCHAR(10) PATH 'lax $.id',
13        first VARCHAR(10) PATH 'lax $.name.first',
14        last VARCHAR(10) PATH 'lax $.name.last',
15        tel CHAR(12) PATH 'lax $.phones[0].number')
16        ERROR On ERROR
17  ) as t;
18
```

ID	FIRST	LAST	TEL



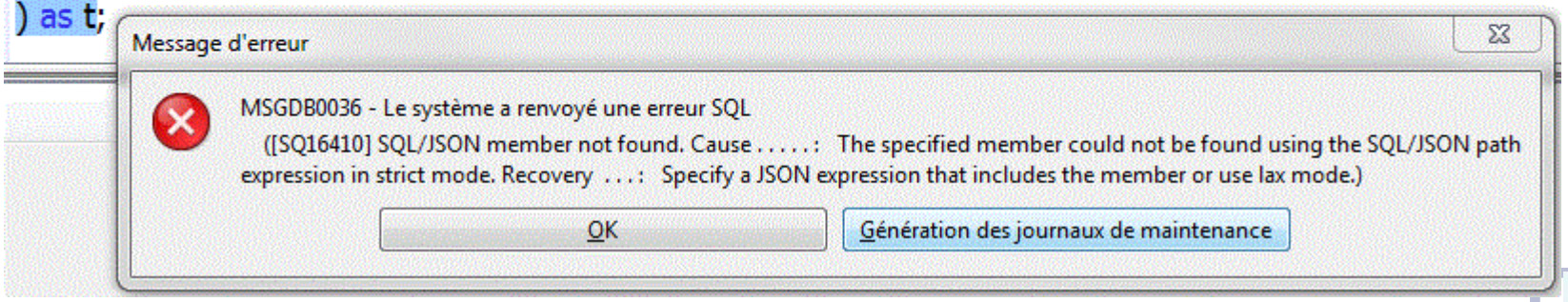
XML vs JSON

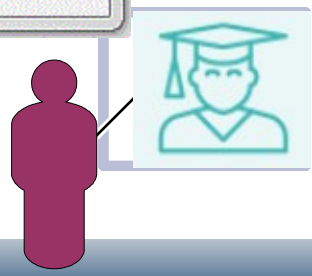
- JSON_TABLE arrive EN TR1(7.3) / TR5 (7.2)

- lax vs strict

  Avec strict, si

```
SELECT * FROM JSON_TABLE(JSON_VAR,
   'strict $.inconnu'
   COLUMNS(
        id VARCHAR(10) PATH 'lax $.id',
        first VARCHAR(10) PATH 'lax $.name.first',
        last VARCHAR(10) PATH 'lax $.name.last',
        tel CHAR(12) PATH 'lax $.phones[0].number')
        ERROR On ERROR
) as t;
```





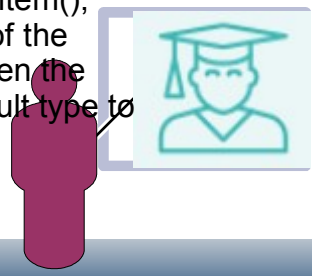
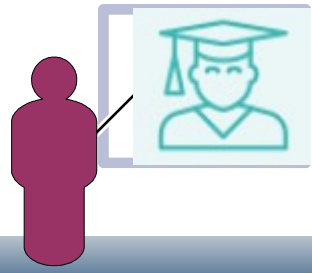
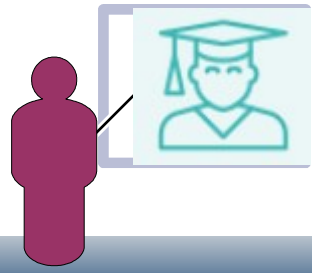
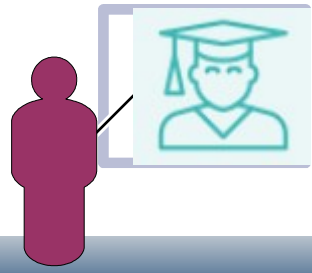
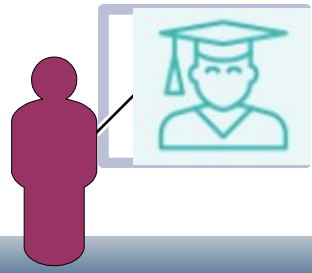
XML vs JSON

- Gestion de tableaux (éléments présents n fois)

- la fonction XMLTABLE ne peut pas mettre une série de valeur dans UNE colonne(par exemple, plusieurs téléphones) , vous recevez **SQ16003**

- ```
  SELECT X.nom ,x.rue, x.ville, x.TEL
   FROM posample/Customer,
  XMLTABLE (
   XMLNAMESPACES(DEFAULT 'http://posample.org'),
    '$c/customerinfo' passing INFO as "c"
   COLUMNS
    NOM    CHAR(30)    PATH 'name',
    RUE    VARCHAR(25) PATH 'addr/street',
    VILLE  VARCHAR(25) PATH 'addr/city',
    TEL    char(10)    PATH 'phone'
  ) AS X
  ```

- An XPath expression has a type that is not valid for the context in which the expression occurs.
  Cause . . . . . : An Expression of data type (item(), **item()+**) cannot be used when the data type **item() is expected** for the context. The following situations might cause this error: * For a constructor function of an XML Schema atomic type, atomization of its argument must not result in more than one atomic value. * The cast of a value of (item(), item()+) to target item() must be a supported cast. * An XPath expression was used in an output column of the XMLTABLE built in table function that could not be casted to the expected type. This typically happens when the expression result contains more than one atomic value and a singleton is expected; or a cast from the result type to the expected data type is not supported. The XPath expression cannot be processed.
  Recovery . . . : Specify a value of the correct type.
  Technical description . . . . . : Error QName=err:XPTY0004.


  ```

XML vs JSON

- Gestion de tableaux (éléments présents n fois)

- vous pouvez alors :

->ne recevoir que la *nième* valeur

Ecrivez

```
SELECT X.nom ,x.rue, x.ville, x.TEL
 FROM posample/Customer,
XMLTABLE (
 XMLNAMESPACES(DEFAULT 'http://posample.org'),
  '$c/customerinfo' passing INFO as "c"
    COLUMNS
     NOM    CHAR(30)       PATH 'name',
     RUE    VARCHAR(25) PATH 'addr/street',
     VILLE  VARCHAR(25) PATH 'addr/city',
     TEL    char(10)           PATH 'phone[1]'
 ) AS X
```



XML vs JSON

- Gestion de tableaux (éléments présents n fois)

- vous pouvez alors :

->retourner la totalité des téléphones sous forme XML

```
SELECT X.nom ,x.rue, x.ville, x.TEL
 FROM posample/Customer,
XMLTABLE (
 XMLNAMESPACES(DEFAULT 'http://posample.org'),
  '$c/customerinfo' passing INFO as "c"
    COLUMNS
     NOM   CHAR(30)        PATH 'name',
     RUE   VARCHAR(25) PATH 'addr/street',
     VILLE  VARCHAR(25) PATH 'addr/city',
     TEL      XML                PATH 'phone'
 ) AS X
```



XML vs JSON

- Gestion de tableaux (éléments présents n fois)

- vous pouvez alors :

-> retourner chaque téléphone en travaillant au niveau ***phone*** dans *customerinfo*

```
SELECT X.nom ,x.rue, x.ville, x.TEL
  FROM posample.Customer,
 XMLTABLE (
  XMLNAMESPACES(DEFAULT 'http://posample.org'),
   '$c/customerinfo/phone' passing INFO as "c"
     COLUMNS
      NOM    CHAR(30)       PATH '../name',
      RUE    VARCHAR(25) PATH '../addr/street',
      VILLE  VARCHAR(25) PATH '../addr/city',
      TEL    char(20)          PATH '.'
  ) AS X
```



XML vs JSON

- Gestion de tableaux (éléments présents n fois)

- La clause FOR ORDINALITY permet d'ajouter un compteur indiquant l'ordre d'apparition de l'élément dans le flux XML.

- ```
  SELECT X.nom ,x.rue, x.ville, x.TEL, X.NUMERO
   FROM posample.Customer,
  XMLTABLE (
   XMLNAMESPACES(DEFAULT 'http://posample.org'),
    '$c/customerinfo/phone' passing INFO as "c"
     COLUMNS
      NOM    CHAR(30)       PATH '../name',
      RUE    VARCHAR(25) PATH '../addr/street',
      VILLE  VARCHAR(25) PATH '../addr/city',
      TEL    char(20)         PATH '.' ,
      NUMERO                 FOR ORDINALITY   ) AS X
  ```

```
Affiche
 ....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....
 NOM                    RUE                  VILLE        TEL          NUMERO
Kathy Smith           5 Rosewood            Toronto   416-555-1358       1
Jim Noodle            1150 Maple Drive      Newtown   905-555-7258       1
Robert Shoemaker      1596 Baseline         Aurora    905-555-2937       1
Helen SUE             1596 Sunset BD        L.A       999-555-1111       1
Helen SUE             1596 Sunset BD        L.A       999-111-4444       2
   ********  Fin de données  ********
```


```

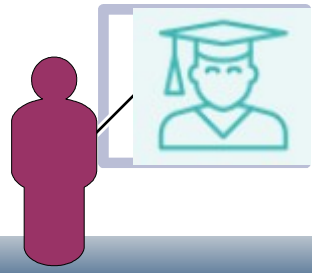
XML vs JSON

- Gestion de tableaux (éléments présents n fois)

- JSON_TABLE, propose une syntaxe approchante :

```
 9  SELECT * FROM JSON_TABLE(JSON_VAR,
10   '$'
11   COLUMNS(
12       id VARCHAR(10) PATH 'lax $.id',
13       first VARCHAR(10) PATH 'lax $.name.first',
14       last VARCHAR(10) PATH 'lax $.name.last',
15       tel CHAR(12) PATH 'lax $.phones[0].number')
16 ) as t;
17
```

ID	FIRST	LAST	TEL
901	John	Doe	555-3762



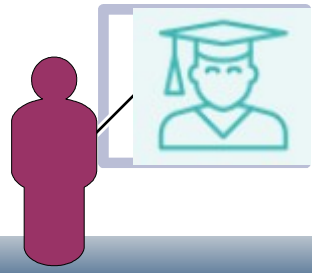
XML vs JSON

- Gestion de tableaux (éléments présents n fois)

- NESTED pour descendre d'un niveau

```
18 SELECT * FROM JSON_TABLE( JSON_VAR,
19    '$'
20    COLUMNS(
21        first VARCHAR(10) PATH 'lax $.name.first',
22        last VARCHAR(10) PATH 'lax $.name.last' ,
23        NESTED '$.phones[*]'COLUMNS (
24            "type" VARCHAR(10),
25            "number" VARCHAR(10)
26        )
27    )
28 ) as t;
29
```

FIRST	LAST	type	number
John	Doe	home	555-3762
John	Doe	work	555-7252



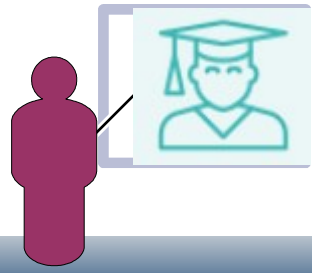
XML vs JSON

- Gestion de tableaux (éléments présents n fois)

- FOR ORDINALITY est également présent

```
39 SELECT * FROM JSON_TABLE( JSON_VAR,
40    '$'
41    COLUMNS(
42        first VARCHAR(10) PATH 'lax $.name.first',
43        last VARCHAR(10) PATH 'lax $.name.last' ,
44        NESTED '$.phones[*]'COLUMNS (
45            numero for ORDINALITY,
46            "type" VARCHAR(10),
47            "number" VARCHAR(10)
48        )
49    )
```

FIRST	LAST	NUMERO	type	number
John	Doe	1	home	555-3762
John	Doe	2	work	555-7252



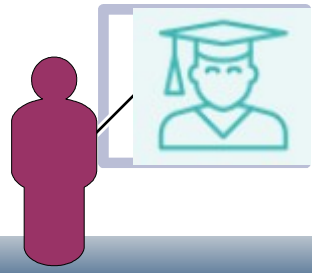
XML vs JSON

- Gestion de tableaux (éléments présents n fois)

- Enfin comme avec XMLTABLE, vous pouvez récupérer la totalité des téléphones, dans un flux

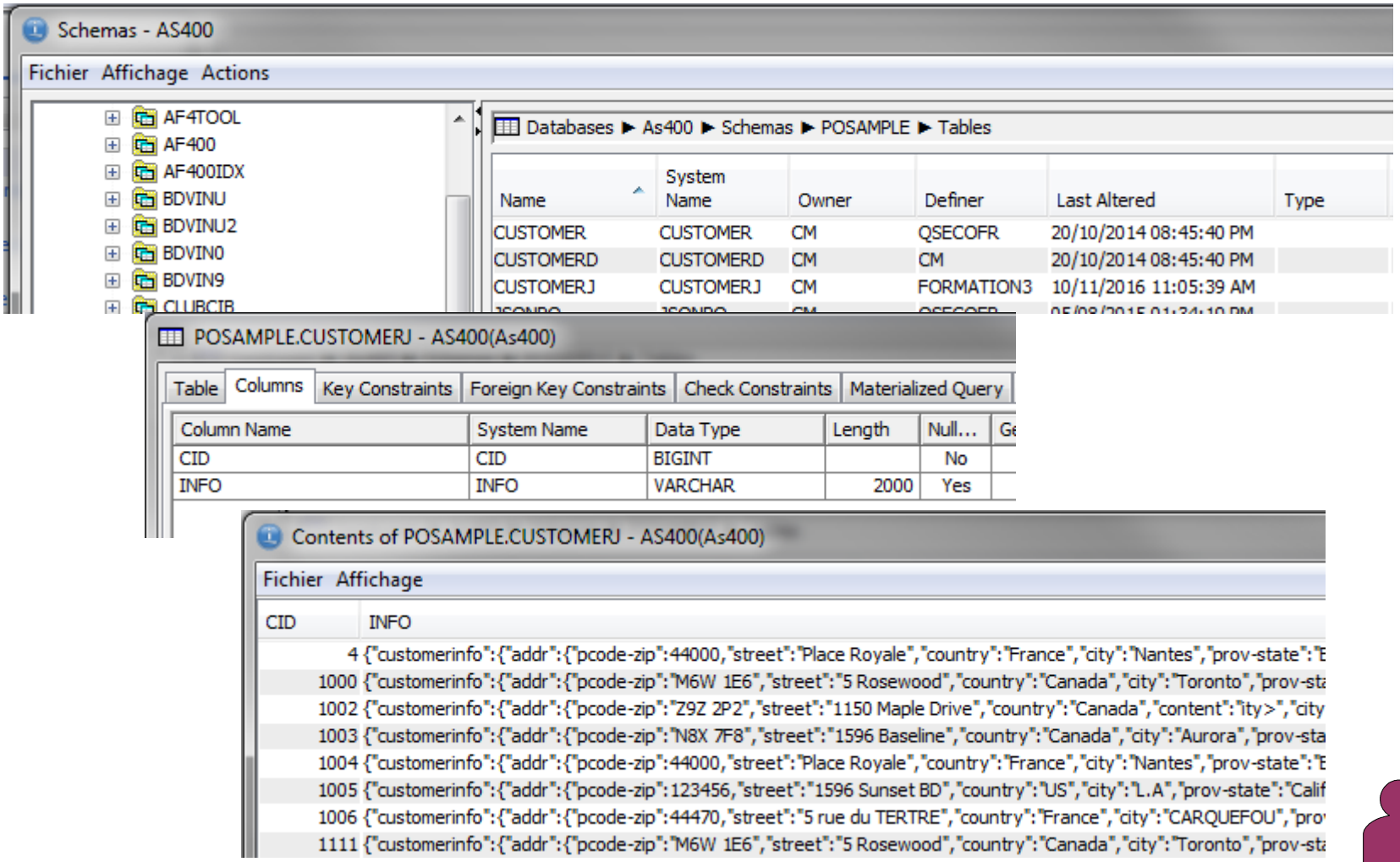
```
30 SELECT * FROM JSON_TABLE( JSON_VAR,
31    '$'
32    COLUMNS(
33        first VARCHAR(10) PATH 'lax $.name.first',
34        last VARCHAR(10) PATH 'lax $.name.last' ,
35        tel CLOB FORMAT JSON PATH '$.phones'
36    )
37 ) as t;
38
```

FIRST	LAST	TEL
John	Doe	[{"type":"home","number":"555-3762"},{"type":"work","number":"555-7252"}]



XML vs JSON

- Bien sur vous pouvez travailler avec des champs VARCHAR d'une table



XML vs JSON

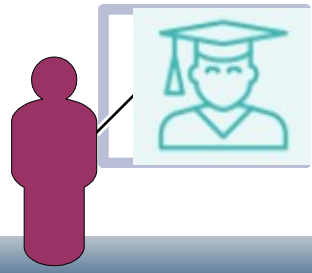
- Gestion de tableaux (éléments présents n fois)

- Remarquez les deux téléphones

"phone":[{"content":"999-555-1111","type":"work"},{"content":"999-111-4444","type":"home"}],"Cid":1005}}

```
51 SELECT CID, ZIPCODE, RUE
52    FROM poSAMPLE.CUSTOMERJ,
53       JSON_TABLE( INFO,
54       'lax $.customerinfo'
55          COLUMNS (
56                zipcode CHAR(8) PATH '$.addr.pcode-zip',
57                rue char(35) PATH '$.addr.street',
58             NESTED '$.phone[*]'COLUMNS (
                    "content" VARCHAR(10) ,
                    "type" VARCHAR(10)
61         ) )
62        ) as client;
```

CID	ZIPCODE	RUE
4	44000	Place Royale
1000	M6W 1E6	5 Rosewood
1002	Z9Z 2P2	1150 Maple Drive
1003	N8X 7F8	1596 Baseline
1004	44000	Place Royale
1005	123456	1596 Sunset BD
1005	123456	1596 Sunset BD
1006	44470	5 rue du TERTRE
1111	M6W 1E6	5 Rosewood

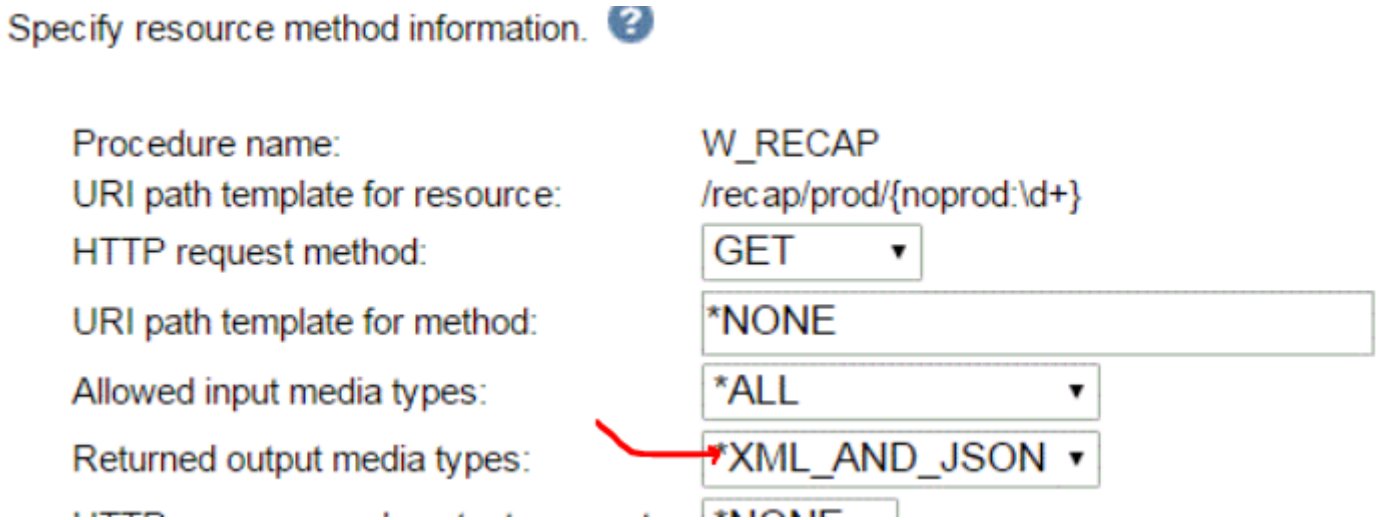


XML vs JSON

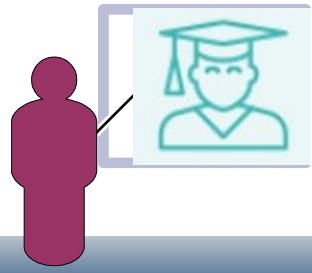
- Gestion de tableaux (éléments présents n fois)

- Quand vous générez un web service REST (avec Liberty)

  Vous aurez le choix entre XML et JSON



Ici, le client indiquera ces préférences par le directive *Accept*

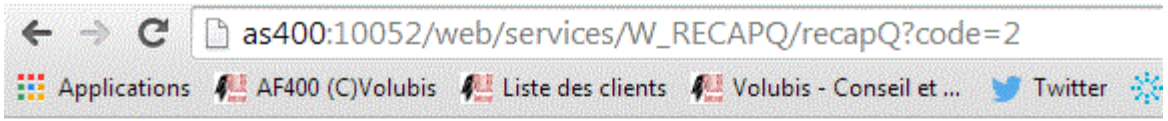


XML vs JSON

- Gestion de tableaux (éléments présents n fois)

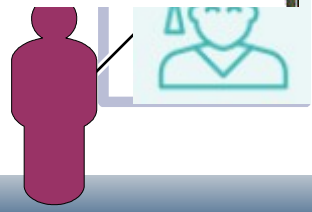
- Quand vous générez un web service REST (avec Liberty)

Vous aurez le choix entre XML et JSON



This XML file does not appear to have any sty

```
▼<w_RECAPResult>
  ▼<INFOCENTRE>
      <CEPAGE>Cabernet Sauvignon</CEPAGE>
      <NBCEPAGE>3</NBCEPAGE>
      <ENCAVE>Non</ENCAVE>
      <NBVIN>3</NBVIN>
      <APPEL00001>Moulis</APPEL00001>
      <PR_CODE>2</PR_CODE>
      <PR_TEL>05 56 58 01 23</PR_TEL>
      <PR_NOM>Château Maucaillou</PR_NOM>
    </INFOCENTRE>
  </w_RECAPResult>
```

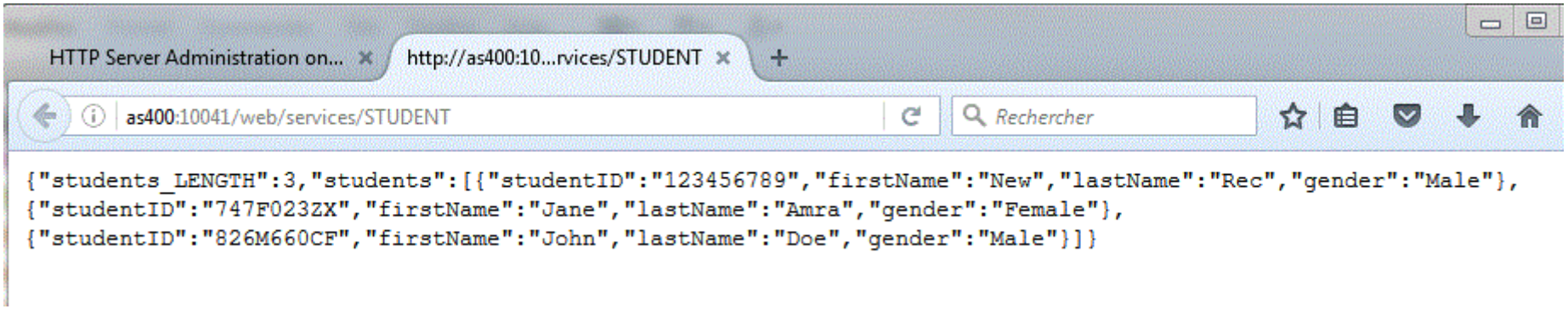


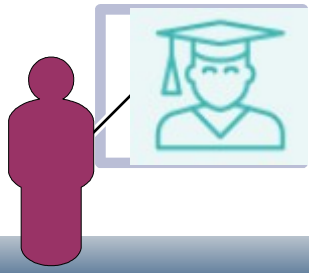
XML vs JSON

- Bien sur vous pouvez travailler avec le résultat d'un web service REST

- Ici l'exemple donné sur DeveloperWorks

  http://www.ibm.com/developerworks/ibmi/library/i-rest-web-services-server3/index.html

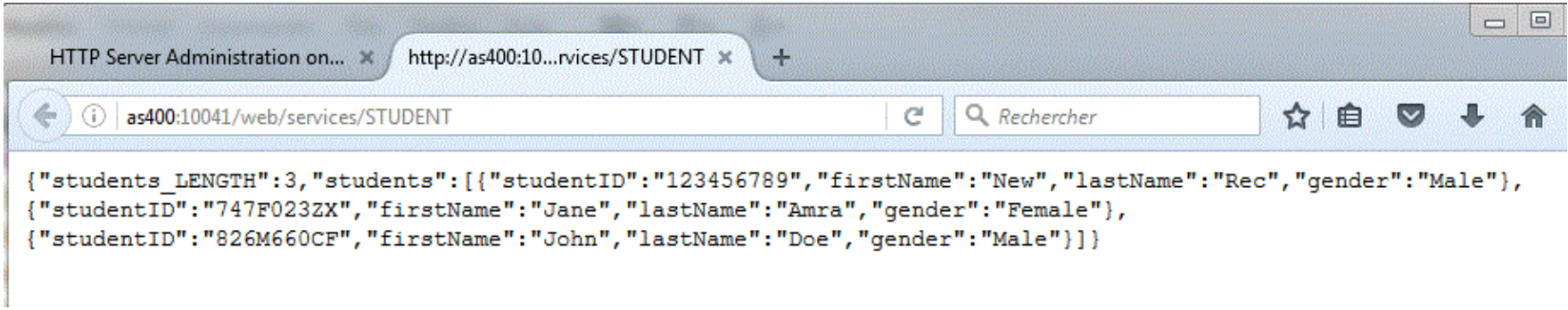




XML vs JSON

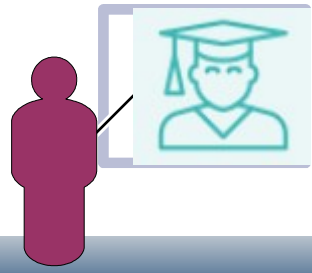
- Bien sur vous pouvez travailler avec le résultat d'un web service REST

- Ici l'exemple donné sur DeveloperWorks




```
64 select * from JSON_TABLE(
65   SYSTOOLS.HTTPGETCLOB('http://as400:10041/web/services/STUDENT',null),
66    '$.students[*]'
67   COLUMNS( ID CHAR(10) PATH '$.studentID',
68                PRENOM VARCHAR(25) PATH '$.firstName',
69                NOM VARCHAR(25) PATH '$.lastName'
70         )) AS X;
71
```

ID	PRENOM	NOM
123456789	New	Rec
747F023ZX	Jane	Amra
826M660CF	John	Doe



XML vs JSON

- Bien sur vous pouvez travailler avec le résultat d'un web service REST

- Avec des données OpenData de la ville de Nantes (voir http://data.nantes.fr)

```
72 --Jardins familiaux de la ville de Nantes
73 select * from JSON_TABLE(
74   SYSTOOLS.HTTPGETCLOB(
75   'http://data.nantes.fr/api/publication/24440040400129_NM_VDN_00083/JARDINS_FAMILIAUX_VDN_STBL/content/?format=json'
76   ,null),    '$.data[*]'
77   COLUMNS( ASSO CHAR(35) PATH '$.ASSOCIATION',
78                ADRESSE VARCHAR(30) PATH '$.ADRESSE',
79                PARCELLES VARCHAR(25) PATH '$.NBRE_PARCELLES'
80         )) AS X;
81
82
```

ASSO	ADRESSE	PARCELLES
Asso " Les jardins de la Roche"	CHEMIN DE LA ROCHE	17
Asso " Les jardins de la Cressonniè	RUE DES RENARDS	10
Asso. "Des petits jardins du Lait d	RUE EMILE PEHANT	1
Confédération Syndicale des Famille	RUE SUZANNE LENGLEN	1
Asso.Jardins Familiaux Nantais	BD PIERRE DE COUBERTIN	14
Asso "les Jardins des Collines"	CHEMIN DES COLLINES	31
Asso. "Jardins de la Marrière "	RUE DU CROISSANT	90
Asso."la crapaudine"	AVENUE DES GOBELETS	92
ECOS	RUE D'HENDAYE	1

XML vs JSON

- Bien sur vous pouvez travailler avec le résultat d'un web service REST

- Évidemment, sur tout cela peut être utilisé en RPG

```
**free
dcl-s json sqltype(clob:32767);
dcl-s wid varchar(10);
dcl-s wnom varchar(10);
dcl-s wtel char(12);
JSON_data='{                                                    +
            "id" : 901,                                         +
            "name" : { "first":"John", "last":"Doe" },          +
            "phones" : [{"type":"home", "number":"555-3762"},   +
                        {"type":"work", "number":"555-7252"}    +
                       ]                                        +
          }';

json_len = %len(%trimr(json_data));
EXEC SQL
  select id, last, tel into :wid, :wnom, :wtel
    from JSON_TABLE(:json, '$'
     COLUMNS( id VARCHAR(10) PATH '$.id',
              first VARCHAR(10) PATH '$.name.first',
              last VARCHAR(10) PATH '$.name.last',
              tel CHAR(12) PATH 'lax $.phones[0].number')
    ) AS X;

    dsply (wid +'-' + wnom);

*INLR=*ON;
```

```
3 > call lir_json
     DSPLY  901-Doe
```