XML - Semaine 8

Pierre Nerzic

février-mars 2022

Le cours de cette semaine présente la gestion et la production de données XML par un SGBD.

Pierre Nerzic

XML dans un SGBD

Pierre Nerzic

Présentation

Comment peut-on stocker et récupérer des données XML dans un SGBD tel que PostgreSQL ?

Ce SGBD intègre différents dispositifs permettant de gérer des données XML :

- Un type de données XML pour stocker un arbre d'éléments,
- Des fonctions pour transformer des données en XML et inversement.

Voir la doc.

Pierre Nerzic

Stockage de données XML

Le type `XML` permet de stocker des éléments XML dans une table SQL :

```
CREATE TABLE TestXML (id INTEGER PRIMARY KEY, data XML);
INSERT INTO TestXML VALUES (1, '<test>ok</test>');
SELECT * FROM TestXML;
```

En fait, le type `XML` est quasiment identique à `VARCHAR` ou `TEXT`. Le seul avantage est la vérification du XML :

```
INSERT INTO TestXML VALUES (2, '<test>mauvais</verif>');
```

```
line 1: Opening and ending tag mismatch: test line 1 and ve
line 1: chunk is not well balanced
```

Texte vers XML

Dans les exemples précédents, il y a une conversion implicite d'une chaîne en XML. Il est préférable de faire appel à la fonction `XMLPARSE` :

```
INSERT INTO TestXML VALUES (3,
    XMLPARSE(DOCUMENT '<verif>oui</verif>'));
INSERT INTO TestXML VALUES (4,
    XMLPARSE(CONTENT 'bidule<test>fragment</test>reste'));
```

Elle prend deux paramètres :

- `DOCUMENT` ou `CONTENT` selon qu'on fournit un arbre complet ou un fragment (qui respecte la norme XML),
- une chaîne contenant le code XML à analyser.

Suffixe ::XML ou mot clé XML

On peut aussi suffixer les chaînes par ::XML ou les faire précéder par le mot-clé XML :

```
INSERT INTO TestXML VALUES (5, '<test>bien</test>'::XML);
INSERT INTO TestXML VALUES (6, XML '<test>aussi</test>');
```

C'est un peu moins bien qu'appeler la fonction XMLPARSE parce qu'il n'y a pas de validation du document mais ça convient pour des contenus constants tels que ceux de l'exemple.

XML vers Texte

Pour le travail inverse, il y a la fonction XMLSERIALIZE :

```
SELECT id, XMLSERIALIZE(DOCUMENT data AS TEXT) FROM TestXML;
SELECT id, XMLSERIALIZE(CONTENT data AS TEXT) FROM TestXML;
```

XMLSERIALIZE(DOCUMENT ou CONTENT col AS TEXT|VARCHAR)

Il faut mettre DOCUMENT si la donnée est un document XML entier (une racine et des sous-éléments), sinon il faut mettre CONTENT (plusieurs éléments).

Pour savoir si une donnée est un document ou un fragment XML :

```
SELECT id, data IS DOCUMENT FROM TestXML;
```

Génération de XML à partir de données normales

Les types et fonctions précédentes permettent de stocker du contenu XML dans une colonne de table. PostgreSQL propose plusieurs fonctions permettant de produire des documents XML à partir de colonnes ordinaires.

```
CREATE TABLE Voitures (id INTEGER PRIMARY KEY,
    marque TEXT, prix NUMERIC, couleur TEXT);
INSERT INTO Voitures VALUES (1, 'Renault', 2500.00, 'blanc');
INSERT INTO Voitures VALUES (2, 'Peugeot', 3200.00, 'gris');
```

Comment produire un document XML contenant l'extension de cette table plus facilement que comme ça :

```
SELECT CONCAT('<voiture id=',id,'><marq>',marque,'</marq></voitu
    FROM Voitures;
```

Génération du XML d'un n-uplet

Pour commencer, voici comment afficher un fragment XML pour chaque n-uplet de la table :

```
SELECT XMLELEMENT(NAME "voiture", XMLATTRIBUTES(id AS "id"))
    FROM Voitures;
```

Cela produit deux lignes, notez que ce sont des éléments vides, possédant seulement un attribut :

```
<voiture id="1"/>
<voiture id="2"/>
```

NB: En SQL, les ' servent à délimiter des chaînes, et les " délimitent des noms de colonnes quand ces noms sont mal formés pour SQL. Ici, je mets les noms d'éléments et attributs en évidence.

Fonction XMLELEMENT

La fonction XMLELEMENT gènère pour chaque n-uplet sélectionné un texte XML correspondant aux paramètres fournis :

```
XMLELEMENT( nom, attributs, contenu...)
```

nom il faut mettre `NAME "nom"` pour donner le nom de l'élément à générer

attributs ils peuvent ne pas être présents. S'il y en a, il faut employer la fonction `XMLATTRIBUTES`
`XMLATTRIBUTES( colonne AS "nomattr", ...)`

contenu il peut être absent, ou c'est une suite de `XMLELEMENT` et/ou de chaînes transformées en texte XML.

Contenu d'un XMLELEMENT

Ce qu'on met dans la partie contenu d'un XMLELEMENT peut être :

- des appels à XMLELEMENT qui seront des sous-éléments
- des chaînes de caractères qui seront transformées en texte XML
- des appels à XMLCOMMENT(texte) devenant des commentaires.

```
SELECT XMLELEMENT(NAME "parent",
            XMLELEMENT(NAME "enfant1"),
            XMLCOMMENT('blabla'),
            'texte',
            XMLELEMENT(NAME "enfant2"));
```

affiche

```
<parent><enfant1/><!--blabla-->texte<enfant2/></parent>
```

Il n'y a pas encore de fonctions pour créer des sections CDATA ainsi que des références d'entités.

Génération du XML d'un n-uplet (suite)

Voici par exemple la génération d'un contenu pour chaque voiture :

```
SELECT XMLELEMENT(NAME "voiture",
         XMLATTRIBUTES(id AS "id"),
         XMLELEMENT(NAME "marq", marque),
         XMLELEMENT(NAME "coul", couleur))
FROM Voitures;
```

Cela produit deux réponses, une par n-uplet :

```
<voiture id="1"><marq>Renault</marq><coul>blanc</coul></voiture>
<voiture id="2"><marq>Peugeot</marq><coul>gris</coul></voiture>
```

Regroupement de fragments XML

Dans les exemples précédents, on voit qu'il y a plusieurs réponses, une par n-uplet dans la base. On peut demander à agréger les réponses dans un seul arbre XML : 

```
SELECT XMLELEMENT(NAME "voitures", XMLAGG(
            XMLELEMENT(NAME "voiture",
                XMLATTRIBUTES(id AS "id"),
                XMLELEMENT(NAME "marq", marque),
                XMLELEMENT(NAME "coul", couleur))))
    FROM Voitures;
```

Il n'y a plus qu'une seule réponse :

```
<voitures>
   <voiture id="1"><marq>Renault</marq><coul>blanc</coul></voitu
   <voiture id="2"><marq>Peugeot</marq><coul>gris</coul></voitur
</voitures>
```


Regroupement de fragments XML (suite)

La fonction `XMLAGG` est une fonction d'agrégation (comme `COUNT`, `AVG`, `MAX`, `SUM`...).

```
XMLAGG( contenu )
```

Elle concatène toutes les réponses fournies par son paramètre pour tous les n-uplets sélectionnés. Ce paramètre doit retourner des fragments XML.

Attention `XMLAGG` ne génère pas d'élément pour englober les fragments. Donc il faut faire :

```
XMLELEMENT(NAME "réponses", XMLAGG(contenu))
```

Concaténation d'éléments

Ne pas confondre `XMLAGG` avec `XMLCONCAT`. Cette dernière concatène simplement ses paramètres. C'est implicite dans la fonction `XMLELEMENT`.

`XMLCONCAT( e1, e2, e3...)`

retourne e1 suivi de e2 suivi de e3...

```
SELECT XMLCONCAT(
            XMLELEMENT(NAME "marque", marque),
            XMLELEMENT(NAME "couleur", couleur))
    FROM Voitures;
```

affiche ceci :

```
<marque>Renault</marque><couleur>blanc</couleur>
<marque>Peugeot</marque><couleur>gris</couleur>
```

Un contenu plus facile à écrire

Au lieu d'écrire le contenu à l'aide de plusieurs XMLELEMENT, on peut employer XMLFOREST :

```
SELECT XMLELEMENT(NAME "voitures", XMLAGG(
        XMLELEMENT(NAME "voiture",
            XMLATTRIBUTES(id AS "id"),
            XMLFOREST(marque AS "marq", couleur AS "coul"))))
FROM Voitures;
```

```
XMLFOREST(colonne1 AS "nom1", colonne2 AS "nom2", ...)
```

Elle revient à écrire : XMLCONCAT(XMLELEMENT(NAME "nom1", colonne1), XMLELEMENT(NAME "nom2", colonne2), ...)

Entête du document

Pour finir, il manque un prologue à notre document XML. C'est le rôle de la fonction XMLROOT :

```
SELECT XMLROOT(
    XMLELEMENT(NAME "voitures", XMLAGG(
        XMLELEMENT(NAME "voiture",
            XMLATTRIBUTES(id AS "id"),
            XMLFOREST(marque AS "marq", couleur AS "coul")))),
    VERSION '1.0')
FROM Voitures;
```

```
<?xml version="1.0"?>
<voitures>
   <voiture id="1"><marq>Renault</marq><coul>blanc</coul></voitu
   <voiture id="2"><marq>Peugeot</marq><coul>gris</coul></voitur
</voitures>
```

Racine du document

La fonction `XMLROOT` prend trois paramètres, les deux derniers sont optionnels.

```
XMLROOT( document, version, standalone )
```

document	ça doit être un seul élément XML, fourni par exemple par `XMLELEMENT`
version	mettre `VERSION '1.0'`
standalone	mettre `STANDALONE YES` ou `NO` uniquement s'il y a une DTD, `YES` si la DTD est à l'intérieur du document XML, ou `NO` si la DTD est dans un fichier séparé. Ne pas mettre cette directive s'il n'y a pas de DTD.

Fournir une DTD

Actuellement PostgreSQL ne définit rien pour ajouter une DTD au document. C'est à faire à la main :

```
SELECT XMLROOT(
     XMLCONCAT(
        '<!DOCTYPE voitures SYSTEM "voitures.dtd">',
        XMLELEMENT(NAME "voitures", XMLAGG(...))),
    VERSION '1.0',
    STANDALONE NO)
FROM Voitures;
```

Mais en plus ça ne marche pas à ce jour. Il y a un bug qui empêche l'analyse de la balise ouvrante de la DTD. (il y a la fonction XMLPI pour générer une *processing instruction* mais rien pour une DTD).

PostgreSQL et XPath

PostgreSQL permet d'employer une fonction XPath 1.0 sur une colonne de type XML. Il faut utiliser la fonction PostgreSQL `XPATH`.
Exemple :

```
SELECT XPATH(
    '/voitures/voiture[@id=1]/couleur/text()',
    data) FROM VoituresXML;
```

`XPATH( expression, colonne )`

- expression : expression XPath à évaluer sur le document contenu dans la colonne,
- document : document XML (ça ne doit pas être un fragment, mais un document complet), issu d'une colonne de table ou une chaîne constante comme ici.

retourne le résultat de l'évaluation de l'expression sur le document

Pierre Nerzic

PHP, PostgreSQL et XML

Pierre Nerzic

Présentation

On se situe sur un serveur HTTP, dans le programme PHP qui répond à une requête d'un client. Comment le script PHP peut-il envoyer des données XML ? On a deux possibilités :

- Utiliser l'API XMLWriter PHP vu au cours précédent,
- Faire encoder les données par le SGBD comme vu précédemment, PHP n'a donc presque à faire, tout est dans la requête SQL.

Utilisation de l'API XMLWriter

Dans ce cas, le script PHP doit :

1. Faire une requête SQL qui retourne les données
2. Utiliser un `XMLWriter` pour encoder les résultats

Les transparents qui suivent montrent un exemple sur la table `Voiture`.

Ouverture de la base

Pour commencer, on crée un objet PDO représentant la connexion avec la base de données, puis une requête SQL :

```php
<?php
$host='localhost';
$db = 'basecontenantlatableVoitures';
$user = 'utilisateur';
$password = 'motdepasse';
try {
    $pdo = new PDO("pgsql:host=localhost;port=5432;dbname=$db",
            $user, $password);

    $sql = 'SELECT * FROM Voitures';
    $result = $pdo->query($sql);
```


Pierre Nerzic

Création d'un écrivain XML

Ensuite, on crée un XMLWriter pour construire le document XML :

```
    header("Content-Type: text/xml");
    $writer = new XMLWriter();
    $writer->openURI('php://output');
    $writer->startDocument('1.0');
    $writer->startElement('voitures');
```

On n'écrit l'entête `text/xml` que lorsqu'on est sûr qu'il va y avoir des réponses, sinon ça pourrait poser un problème avec l'affichage des messages d'erreur, voir la clause `catch`.

Création d'un écrivain XML

La suite consiste à écrire les n-uplets à l'aide de l'écrivain : 

```
while ($row = $result->fetch(PDO::FETCH_ASSOC)) {
    $writer->startElement('voiture');
    $writer->writeAttribute('id', $row['id']);
    $writer->writeElement('marque', $row['marque']);
    $writer->writeElement('couleur', $row['couleur']);
    $writer->endElement();
}
```


Pierre Nerzic

Terminaison

Pour finir, on ferme le document XML et on l'émet vers le client :

```
    $writer->endElement();
    $writer->endDocument();
    $writer->flush();

} catch (PDOException $e) {
    echo $e->getMessage();
}
?>
```


Pierre Nerzic

Encodage par le SGBD

Dans cette approche, c'est le SGBD qui fait l'encodage en XML. Le script PHP doit seulement transmettre le résultat au client.

Le début est identique :

```php
<?php
$host='localhost';
$db = 'basecontenantlatableVoitures';
$user = 'utilisateur';
$password = 'motdepasse';
try {
    $pdo = new PDO("pgsql:host=localhost;port=5432;dbname=$db",
            $user, $password);
```

Requête SQL

La requête SQL est nettement plus complexe car c'est elle qui encode en XML :

```
$sql = 'SELECT XMLROOT(
    XMLELEMENT(NAME "voitures", XMLAGG(
        XMLELEMENT(NAME "voiture",
            XMLATTRIBUTES(id AS "id"),
            XMLFOREST(marque AS "marque",
                      couleur AS "couleur")))),
    VERSION \'1.0\',
    STANDALONE YES)
FROM Voitures;';
$result = $pdo->query($sql);
```

Notez les \' pour masquer les ' dans la requête.

Pierre Nerzic

Reste du script PHP

Il ne reste plus que ça, l'affichage du premier résultat de la requête (sachant qu'elle agrège les n-uplets dans un document XML complet) :

```
    header("Content-Type: text/xml");
    echo $result->fetch()[0];

} catch (PDOException $e) {
    echo $e->getMessage();
}
?>
```


Pierre Nerzic

Comparaisons

- XML généré par le script PHP
- XML généré par le SGBD
  - simplifie le code PHP
  - quelques bugs pour l'instant avec les entités, CDATA et DOCTYPE

Autres formats de données internet

Alternatives au XML

XML sert à :

- représenter des informations et permettre des recherches à l'aide de XQuery
- échanger des informations entre un serveur et un client, par exemple avec AJAX

Pour ce dernier point, il existe des alternatives :

- JSON
- YAML

Pierre Nerzic

JSON

JavaScript Object Notation est un format texte qui permet de représenter des objets complexes (biblio).

```
class Article {
    private int id;
    private String nom;
    private float prix;
    private String[] infos;
}
```

```
{
    "id": 1,
    "nom": "ballon de basket",
    "prix": 12.50,
    "infos": ["certifié", "orange", "renforcé"]
}
```


Pierre Nerzic

Schéma de JSON

Comme pour XML, il est possible de valider un document JSON à l'aide d'un schéma. Par exemple le document précédent répond à ce schéma, lui-même en JSON : 

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Article",
    "description": "un article de sport",
    "type": "object",
    "properties": {
        "id": {
            "description": "identifiant de l'article",
            "type": "integer"
        },
```


Suite du schéma

```
        "nom": {
            "type": "string"
        },
        "prix": {
            "type": "number",
            "minimum": 0, "exclusiveMinimum": true
        },
        "infos": {
            "type": "array", "items": { "type": "string" },
            "uniqueItems": true
        }
    },
    "required": ["id", "nom", "prix"]
}
```

Outils de validation

Il y a des outils pour valider un document par un schéma, par exemple en ligne

http://www.jsonschemavalidator.net/

Pour davantage d'informations sur les schémas JSON :
http://json-schema.org/

Sérialisation JSON

Il est très simple de produire un document JSON en PHP :

```php
<?php
class Article {
    public $nom = "ballon de basket";
    public $prix = 12.50;
    public $infos = ["certifié", "orange", "renforcé"];
};
$article = new Article;
header('Content-type:application/json; charset=utf-8');
echo json_encode($article);
?>
```

- NB: la classe est définie d'une manière très désinvolte.
- NB: il y a un problème avec les caractères accentués.

Dé-sérialisation JSON

Inversement, pour récupérer un objet JavaScript à partir de JSON, il suffit de faire ceci :

```
<script>
var xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        var article = JSON.parse(xmlhttp.responseText);
        ...
    }
};
xmlhttp.open("GET", "http://serveur/article.php", true);
xmlhttp.send();
```

C'est beaucoup plus simple qu'analyser du XML.

YAML

YAML est un format de représentation des données similaire à JSON. YAML représente toutes les données à l'aide de liste (énumérations commençant par un -) et de dictionnaires (paires nom : valeur).

Voici un exemple :

```yaml
nom:    "ballon de basket"
prix:   12.50
infos:
    - "certifié"
    - "orange"
    - "renforcé"
```

Comme pour JSON, il y a des outils de sérialisation et de dé-sérialisation.

Fin du cours

Ce qu'il faut retenir de ce cours :

- Formats polyvalents et ouverts pour représenter des informations : XML, JSON ou YAML
- Validation avec un modèle de document : DTD, XSL ou RelaxNG
- Sérialisation/désérialisation d'objets : Java, JS ou autre
- Extraction d'informations à l'aide de requêtes XPath ou XQuery
- Transformation dans un autre format : XQuery ou XSLT

Pierre Nerzic