

Utiliser XML dans MySQL 5.1 et 6.0



par Jon Stephens Joris Crozier (Traducteur)

Date de publication : 04/08/2008

Dernière mise à jour :

Cet article est la traduction de **Using XML in MySQL 5.1 and 6.0** (disponible [ici](#)) et a pour but de vous montrer l'utilisation de MySQL 5.1 et 6.0 avec les fichiers XML.

Introduction.....	3
I - Utiliser XML vers et depuis MySQL.....	3
I-1 - Exporter les données.....	3
I-2 - Importer XML et ses données.....	9
En utilisant LOAD_FILE().....	9
En important les données XML de mysqldump dans la colonne de la table MySQL avec une procédure stockée.....	10
En utilisant Load XML.....	12
La clause LOAD DATA utilisée dans LOAD XML.....	13
ROWS IDENTIFIED BY '<tagname>'.....	13
II - Les fonctionnalités XPath.....	14
II-1 - La fonction ExtractValue().....	14
II-2 - La fonction UpdateXML().....	15
II-3 - Le support de XPath dans MySQL 5.1 et 6.0.....	16
Les fonctions et les opérateurs supportés.....	16
Les fonctions et les opérateurs non supportés.....	18
Traitement d'erreur.....	20
Les variables utilisateur dans les expressions XPath.....	21
III - Considérations de Sécurité.....	23
III-1 - Charger des données depuis un fichier.....	23
III-2 - Les privilèges XML et MySQL.....	24
III-3 - Injection XPath.....	25

Introduction

Dans cet article, nous discutons des fonctionnalités XML disponibles dans MySQL, avec l'accent sur les nouvelles fonctionnalités des versions 5.1 et 6.0.

Nous partons du principe que vous avez déjà des connaissances sur XML et que vous savez ce que les termes "valide " et " bien formé " veulent dire.

Nous supposons aussi que vous ayez quelques connaissances sur XPath.

Nous allons couvrir ces différents sujets :

- Les méthodes pour exporter des données MySQL Dans le format XML , incluant l'utilisation de **lib_MySQLudf_xql** , une librairie tierce qui peut être utilisée pour cette tâche
- L'utilisation des fonctions (nouvelles dans MySQL 5.1) **ExtractValue()** et **UpdateXML()** pour travailler avec XML et XPath
- Le stockage de données depuis XML dans une base MySQL en utilisant la déclaration **LOAD XML** (implémentée dans MySQL 6.0)
- Quelques elements de sécurités à garder à l'esprit quand on utilise ces techniques

I - Utiliser XML vers et depuis MySQL

Dans cette section, nous discutons de la manière d'exporter les données en XML depuis MySQL, et de comment stocker les données obtenues dans un fichier XML vers une base MySQL.

I-1 - Exporter les données

Dans cette section, nous commençons avec quelques données déjà stockées dans une table MySQL , et démontrons différentes façons de les sortir dans le format XML.

En utilisant l'option **-xml**. Les 2 programmes clients **MySQL** et **mysqldump** supportent une option au démarrage qui leur demande de produire une sortir XML.

Voici un bref exemple en utilisant le client **MySQL** :

Commande MySQL

```
shell> mysql -uroot -e "SHOW VARIABLES LIKE '%version%" --xml
```

Sortie XML générée

```
<?xml version="1.0"?>

<resultset statement="SHOW VARIABLES LIKE '%version%"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
    <field name="Variable_name">protocol_version</field>
    <field name="Value">10</field>
  </row>

  <row>
    <field name="Variable_name">version</field>
    <field name="Value">5.1.22-beta-debug</field>
  </row>

  <row>
    <field name="Variable_name">version_comment</field>
    <field name="Value">Source distribution</field>
  </row>

  <row>
    <field name="Variable_name">version_compile_machine</field>
```

Sortie XML générée

```
<field name="Value">x86_64</field>
</row>

<row>
  <field name="Variable_name">version_compile_os</field>
  <field name="Value">suse-linux-gnu</field>
</row>
</resultset>
```

Le contenu de l'élément **<field name="Value">**, correspond à la valeur trouvée dans la colonne **Value** affichée quand la même commande est exécutée dans le client **MySQL** sans l'option **--xml**, comme montré ici :

Commande MySQL

```
shell> mysql -uroot -e "SHOW VARIABLES LIKE '%version%'"
```

Sortie générée

Variable_name	Value
protocol_version	10
version	5.1.22-beta-debug
version_comment	Source distribution
version_compile_machine	x86_64
version_compile_os	suse-linux-gnu

Naturellement, les valeurs elles-mêmes dépendent de la version de MySQL que vous utilisez et de la machine sur laquelle MySQL est installé, donc si vous utilisez la commande, vos résultats peuvent être différents de ceux montrés ici.

Table d'exemple pour l'export Xml.


Le reste de cette section utilise la table créée et peuplée par les commandes sql ci-dessous.

Commandes MySQL

```
CREATE SCHEMA xmltest;

CREATE TABLE xmltest.cities (
  name CHAR(35) NOT NULL DEFAULT '',
  country CHAR(52) NOT NULL DEFAULT '',
  population int(11) NOT NULL DEFAULT '0'
);

INSERT INTO cities VALUES ('Mumbai (Bombay)', 'India', 10500000);
INSERT INTO cities VALUES ('Seoul', 'South Korea', 9981619);
INSERT INTO cities VALUES ('São Paulo', 'Brazil', 9968485);
INSERT INTO cities VALUES ('Shanghai', 'China', 9696300);
INSERT INTO cities VALUES ('Jakarta', 'Indonesia', 9604900);
INSERT INTO cities VALUES ('Karachi', 'Pakistan', 9269265);
INSERT INTO cities VALUES ('Istanbul', 'Turkey', 8787958);
INSERT INTO cities VALUES ('Ciudad de México', 'Mexico', 8591309);
INSERT INTO cities VALUES ('Moscow', 'Russian Federation', 8389200);
INSERT INTO cities VALUES ('New York', 'United States', 8008278);
```


 Cette table a été créée à l'origine en utilisant la commande SQL suivante sur la vénérable table exemple **world**.

Commandes MySQL

```
CREATE TABLE xmltest.cities
```

Commandes MySQL

```
SELECT i.Name AS name,
o.Name AS country,
i.Population AS population
FROM City i JOIN Country o ON i.CountryCode=o.Code
ORDER BY i.Population DESC LIMIT 10;
```

Vous pouvez obtenir une copie de la table **world** ici  <http://dev.mysql.com/doc/>

Depuis MySQL 5.1.12 les formats **<field>** et **<row>** produits par le client **MySQL** correspondent à ceux produits par **mysqldump**.

Cependant, l'élément racine de la sortie **mysql --xml** est **<resultset>**, dont l'attribut de la commande contient la commande SQL passée à **MySQL** comme montré ici :

Commande MySQL

```
shell> mysql -uroot --xml -e 'SELECT * FROM xmltest.cities ORDER BY name'
```

Sortie XML générée

```
<?xml version="1.0"?>
<resultset statement="SELECT * FROM xmltest.cities ORDER BY name"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
    <field name="name">Ciudad de Méico</field>
    <field name="country">Mexico</field>
    <field name="population">8591309</field>
  </row>
  <row>
    <field name="name">Istanbul</field>
    <field name="country">Turkey</field>
    <field name="population">8787958</field>
  </row>
  <row>
    <field name="name">Jakarta</field>
    <field name="country">Indonesia</field>
    <field name="population">9604900</field>
  </row>
  <row>
    <field name="name">Karachi</field>
    <field name="country">Pakistan</field>
    <field name="population">9269265</field>
  </row>
  <row>
    <field name="name">Moscow</field>
    <field name="country">Russian Federation</field>
    <field name="population">8389200</field>
  </row>
  <row>
    <field name="name">Mumbai (Bombay)</field>
    <field name="country">India</field>
    <field name="population">10500000</field>
  </row>
  <row>
    <field name="name">New York</field>
    <field name="country">United States</field>
    <field name="population">8008278</field>
  </row>
  <row>
    <field name="name">São Paulo</field>
```

Sortie XML générée

```

        <field name="country">Brazil</field>
        <field name="population">9968485</field>
    </row>

    <row>
        <field name="name">Seoul</field>
        <field name="country">South Korea</field>
        <field name="population">9981619</field>
    </row>

    <row>
        <field name="name">Shanghai</field>
        <field name="country">China</field>
        <field name="population">9696300</field>
    </row>
</resultset>
    
```

La sortie de **mysqldump-xml** est structurée quelque peu différemment, comme montré ici :

Commande MySQL

```
shell> mysqldump --xml xmltest cities
```

Sortie XML générée


```

<?xml version="1.0"?>
<mysqldump xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<database name="xmltest">
  <table_structure name="cities">
    <field Field="name" Type="char(35)" Null="NO" Key="" Default=""
    Extra="" />
    <field Field="country" Type="char(52)" Null="NO" Key="" Default=""
    Extra="" />
    <field Field="population" Type="int(11)" Null="NO" Key="" Default="0"
    Extra="" />
    <options Name="cities" Engine="MyISAM" Version="10" Row_format="Fixed"
    Rows="10" Avg_row_length="92" Data_length="920"
    Max_data_length="25895697857380351" Index_length="1024"
    Data_free="0" Create_time="2007-08-24 14:19:42"
    Update_time="2007-08-24 14:19:42" Collation="latin1_swedish_ci"
    Create_options="" Comment="" />
  </table_structure>
  <table_data name="cities">
    <row>
      <field name="name">Mumbai (Bombay)</field>
      <field name="country">India</field>
      <field name="population">10500000</field>
    </row>
    <row>
      <field name="name">Seoul</field>
      <field name="country">South Korea</field>
      <field name="population">9981619</field>
    </row>
    <row>
      <field name="name">São Paulo</field>
      <field name="country">Brazil</field>
      <field name="population">9968485</field>
    </row>
    <row>
      <field name="name">Shanghai</field>
      <field name="country">China</field>
      <field name="population">9696300</field>
    </row>
    <row>
      <field name="name">Jakarta</field>
      <field name="country">Indonesia</field>
      <field name="population">9604900</field>
    </row>
  </table_data>
</database>
    
```

Sortie XML générée

```

<field name="name">Karachi</field>
<field name="country">Pakistan</field>
<field name="population">9269265</field>
</row>
<row>
<field name="name">Istanbul</field>
<field name="country">Turkey</field>
<field name="population">8787958</field>
</row>
<row>
<field name="name">Ciudad de México</field>
<field name="country">Mexico</field>
<field name="population">8591309</field>
</row>
<row>
<field name="name">Moscow</field>
<field name="country">Russian Federation</field>
<field name="population">8389200</field>
</row>
<row>
<field name="name">New York</field>
<field name="country">United States</field>
<field name="population">8008278</field>
</row>
</table_data>
</database>
</mysqldump>
    
```

 *Le formatage de certaines sorties XML a été légèrement changé pour s'adapter à l'espace disponible d'une page à imprimer.*

mysqldump-xml emploie les éléments suivants :

- L'élément racine du document XML représentant le dump est **<mysqldump>**
- Les définitions de table et les données appartenant à chaque base représentée dans le dump sont enveloppées ensemble dans l'élément **<database>** dont la valeur de l'attribut **name** est le nom de cette base de données
- Chaque définition de table est enveloppée dans un élément **<table_structure>**
- Les données de chaque table sont enveloppées dans un élément **<table_data>**, et comprennent les éléments **<field>** et **<row>**

Pour sauvegarder les sorties de **MySQL** et **mysqldump** dans un fichier, utilisez simplement l'opérateur **>** avec le nom de fichier désiré, comme montré ici :

Commande MySQL

```

shell> mysql -uroot --xml xmltest -e 'SELECT name FROM cities LIMIT 2' > /tmp/2cities.xml
shell> more /tmp/2cities.xml
    
```


Affichage du contenu du fichier /tmp/2cities.xml

```

<?xml version="1.0"?>
<resultset statement="SELECT name,country FROM cities LIMIT 2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
    <field name="name">Mumbai (Bombay)</field>
    <field name="country">India</field>
  </row>
  <row>
    <field name="name">Seoul</field>
    <field name="country">South Korea</field>
  </row>
    
```

Affichage du contenu du fichier /tmp/2cities.xml

```
</resultset>
```

 La déclaration XML d'un espace de nom est incluse dans la sortie XML de **MySQL** et **mysqldump** depuis MySQL 5.1.18

Il peut y avoir des fois où votre application requiert un format XML différent de celui obtenu normalement par **MySQL** et **mysqldump**.

Supposons, par exemple, que votre application ait besoin d'un format comme celui-ci :

```
<cities>
  <city>name1</city>
  <city>name2</city>
  <-- etc. -->
</cities>
```

Une manière d'obtenir cette sortie est d'utiliser la concaténation avec les fonctions **CONCAT()** et **GROUP_CONCAT()**, comme ceci :

Commande MySQL

```
mysql> SELECT CONCAT('\n<cities>\n',
->   GROUP_CONCAT(' <city>', name, '</city>\n' SEPARATOR ''),
->   '</cities>') AS xmldoc
-> FROM cities\G
```

Sortie XML générée

```
***** 1. row *****
xmldoc:
<cities>
  <city>Mumbai (Bombay)</city>
  <city>Seoul</city>
  <city>São Paulo</city>
  <city>Shanghai</city>
  <city>Jakarta</city>
  <city>Karachi</city>
  <city>Istanbul</city>
  <city>Ciudad de Méico</city>
  <city>Moscow</city>
  <city>New York</city>
</cities>
1 row in set (0.01 sec)
```

Un autre exemple dans ce format :

```
<cities>
  <city name="name1" population="population1"/>
  <city name="name2" population="population2"/>
  <-- etc. -->
</cities>
```

Ceci peut être produit par ces commandes :

Commande MySQL

```
mysql> SELECT CONCAT(
->   '\n<cities>',
->   GROUP_CONCAT(
->   '\n\t<city name="', name, '" population="', population, '"/>'
-> SEPARATOR ' '
->   ),
```

Commande MySQL


```
->          '\n</cities>'
->          ) AS xmldoc
-> FROM cities\G
```

Sortie XML générée

```
***** 1. row *****
xmldoc:
<cities>
  <city name="Mumbai (Bombay)" population="10500000"/>
  <city name="Seoul" population="9981619"/>
  <city name="São Paulo" population="9968485"/>
  <city name="Shanghai" population="9696300"/>
  <city name="Jakarta" population="9604900"/>
  <city name="Karachi" population="9269265"/>
  <city name="Istanbul" population="8787958"/>
  <city name="Ciudad de México" population="8591309"/>
  <city name="Moscow" population="8389200"/>
  <city name="New York" population="8008278"/>
</cities>
1 row in set (0.01 sec)
```

Vous pouvez voir que la production de XML simple par de tels moyens peut rapidement devenir compliquée. Tant que vous pouvez envelopper les commandes **SELECT** dans des procédures stockées, le fait est que les routines stockées de MySQL ne peuvent pas (du moins jusqu'à présent) prendre un nombre variable d'arguments. Heureusement, il existe une librairie tierce qui peut être utilisée avec MySQL pour rendre cette tâche plus facile.

Utilisation de la librairie lib_mysqludf_xql

La librairie lib_mysqludf_xql est un lot d' **UDF** écrites et placées sous licence GPL par Arnold Daniels. Les sources sont disponibles ici : www.mysqludf.org

Commande MySQL

```
mysql> SELECT xql_element('city', name) FROM cities;
```

I-2 - Importer XML et ses données

Dans cette section, nous discutons de quelques techniques pour importer des données XML dans une table MySQL

En utilisant LOAD_FILE()

La façon la plus simple de stocker du XML dans une table MySQL est d'utiliser la fonction **LOAD_FILE()** pour pouvoir importer un document XML en entier, le rendre disponible en tant que chaîne de caractères, et d'insérer cette chaîne dans une colonne de la table.

En utilisant le fichier **2cities.xml** créé plus tôt, il est possible de faire quelque chose comme ceci :

Commande MySQL

```
mysql> USE xmltest;
mysql> CREATE TABLE xmldocs (
->   id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
->   doc_content BLOB NOT NULL,
->   comment VARCHAR(100) NOT NULL DEFAULT ''
-> );
Query OK, 0 rows affected (0.04 sec)

mysql> INSERT INTO xmldocs VALUES
->   (NULL, LOAD_FILE('/tmp/2cities.xml'), '2 cities file');
Query OK, 1 row affected (0.00 sec)
```

Commande MySQL

```
mysql> SELECT * FROM xmldocs\G
```

Sortie XML générée

```

    id: 1
doc_content: <?xml version="1.0"?>

<resultset statement="SELECT name,country FROM cities LIMIT 2
" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
    <field name="name">Mumbai (Bombay)</field>
    <field name="country">India</field>
  </row>

  <row>
    <field name="name">Seoul</field>
    <field name="country">South Korea</field>
  </row>
</resultset>

    comment: 2 cities file
1 row in set (0.00 sec)
```

En important les données XML de mysqldump dans la colonne de la table MySQL avec une procédure stockée.

Charger un fichier XML entier dans une seule ligne de la table MySQL résout le problème d'insertion du XML dans MySQL, où il peut être parsé en utilisant les fonctions **XPATH** de MySQL 5.1 (voir la section XPATH de l'article)

Quoi qu'il en soit, avoir à faire ça à chaque fois que vous voulez accéder aux données n'est pas terrible. Heureusement le développeur **Alexander Barkov** a écrit une procédure stockée nommée **xmldump_load** qui extrait les données depuis les éléments et les attributs XML trouvés dans un fichier créé via **mysqldump -xml** et insert ces données dans les colonnes d'une table MySQL. La source de cette procédure est montrée ici :

xmldump_load

```

DELIMITER |

DROP PROCEDURE IF EXISTS xmldump_load |

CREATE PROCEDURE xmldump_load(file_name VARCHAR(128),
    database_name VARCHAR(128),
    table_name VARCHAR(128))
BEGIN
    DECLARE xml TEXT;

    DECLARE nrows INT;
    DECLARE rownum INT DEFAULT 1;

    DECLARE ncols INT;
    DECLARE colnum INT DEFAULT 1;

    DECLARE ins_list TEXT DEFAULT '';
    DECLARE val_list TEXT DEFAULT '';

    DECLARE tmp VARCHAR(255);

    -- load the XML file's contents into a string
    SET xml = LOAD_FILE(file_name);

    -- get the number of <row>s in this table
    SET nrows = ExtractValue(xml,
        'count(/mysqldump/database[@name=$database_name]/table_data[@name=$table_name]/row)');

```

xmldump_load

```

-- get the number of <field>s (columns) in this table
SET ncols = ExtractValue(xml,
    'count(/mysqldump/database[@name=$database_name]/table_data[@name=$table_name]/row[1]/field)');

-- for each <row>
WHILE rownum <= nrows DO

    -- for each <field> (column)
    WHILE colnum <= ncols DO
        SET tmp = ExtractValue(xml,
            '/mysqldump/database[@name=$database_name]/table_data[@name=$table_name]/row[$rownum]/
            field[$colnum]/@name');
        SET ins_list = CONCAT(ins_list, tmp, IF(colnum<ncols, ',', ''));
        SET tmp = ExtractValue(xml,
            '/mysqldump/database[@name=$database_name]/table_data[@name=$table_name]/row[$rownum]/
            field[$colnum]');
        SET val_list = CONCAT(val_list, '','', tmp, '','', IF(colnum<ncols, ',', ''));
        SET colnum = colnum + 1;
    END WHILE;

    SET @ins_text = CONCAT('INSERT INTO t1 (', ins_list, ') VALUES (', val_list, ')');

    SET ins_list = '';
    SET val_list = '';

    PREPARE stmt FROM @ins_text;
    EXECUTE stmt;

    SET rownum = rownum + 1;
    SET colnum = 1;
END WHILE;
END |

DELIMITER ;
    
```

Cette procédure stockée emploie les variables utilisateurs XPATH et ne fonctionne donc que sous MySQL 5.1.20 ou supérieur. Vous pouvez trouver une copie de **create-xmldump-load.sql** à la fin de cet article.

 *L'appelant de cette procédure stockée doit avoir les privilèges **FILE** de MySQL*

Vous pouvez tester **xmldump_load** en utilisant le script **test.sh** suivant :

test.sh

```

# Demo for xmldump_load()

DB="test"

# Change "root" and "mypass" in the following 2 lines to
# a user and password appropriate to your installation

MYSQL="mysql -uroot -pmypass --socket=/tmp/mysql.sock"
MYSQLDUMP="mysqldump -uroot -pmypass --socket=/tmp/mysql.sock"

# Creates a test table with two columns and fills it with some data

$MYSQL $DB <> END
SELECT VERSION();
DROP TABLE IF EXISTS t1;
CREATE TABLE t1 (a INT, b VARCHAR(128));
INSERT INTO t1 VALUES (1, '11111');
INSERT INTO t1 VALUES (2, '22222');
INSERT INTO t1 VALUES (3, '33333');
INSERT INTO t1 VALUES (4, '44444');
INSERT INTO t1 VALUES (5, '55555');
INSERT INTO t1 VALUES (6, '66666');
INSERT INTO t1 VALUES (7, '77777');
    
```

test.sh

```

INSERT INTO t1 VALUES (8,'88888');
INSERT INTO t1 VALUES (9,'99999');
END

# Dumps data into an XML file

$MYSQLDUMP --xml $DB t1 > /tmp/t1.xml

# Empties the table

$MYSQL --execute="DELETE FROM t1" $DB

# Creates the procedure, calls it, and
# makes sure we've restored all records

$MYSQL $DB << END
\. create-xmldump-load.sql
CALL xmldump_load('/tmp/t1.xml', 'test', 't1');
SELECT * FROM t1;
END

# Performs cleanup

# Comment out the remaining lines if you wish to
# preserve the stored procedure, table, and XML
# file following the test run

$MYSQL $DB << END
DROP PROCEDURE xmldump_load;
DROP TABLE t1;
END

rm /tmp/t1.xml
    
```

En utilisant Load XML

Une contribution d'**Erik Wetterberg** d'une nouvelle commande SQL a été acceptée pour MySQL 6.0, et sera disponible à partir de la version 6.0.3. **LOAD XML** simplifie réellement la tâche d'importation de données depuis un fichier XML vers une table MySQL, sans avoir à utiliser la procédure stockée montrée précédemment.

La syntaxe de cette commande est montrée ici :

LOAD XML

```

LOAD XML [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'filename'
[REPLACE | IGNORE]
INTO TABLE [db_name.]tbl_name
[CHARACTER SET charset_name]
[ROWS IDENTIFIED BY '<tagname>']
[IGNORE number [LINES | ROWS]]
[(column_or_user_var,...)]
[SET col_name = expr,...]
    
```

Cette commande lit les données d'un fichier XML dans une table, et agit comme complément de **MySQL** et **mysqldump** dans le mode de sortie XML (c a d en utilisant l'option **--xml**) .

Le nom de fichier **filename** doit être donné en tant que chaîne de caractères . L'élément **tagname** de l'option **ROWS IDENTIFIED BY** doit aussi être donné en tant que chaîne de caractères et doit être entourée des signes supérieur et inférieur (< et >)

La clause LOAD DATA utilisée dans LOAD XML

Si vous avez utilisé la commande MySQL **LOAD DATA** décrite précédemment, vous devriez trouver les clauses suivantes familières, puisque elles travaillent essentiellement de la même façon que se soit **LOAD XML** que **LOAD DATA** :

- **LOW_PRIORITY** ou **CONCURRENT**
- **LOCAL**
- **REPLACE** ou **IGNORE**
- **CHARACTER SET**
- La clause **IGNORE number LINES** est analogue aux clauses **IGNORE ... LINES** de la commande **LOAD DATA**. **LOAD XML** accepte aussi **IGNORE number ROWS** Dans les deux cas, la clause force les **number** premières lignes à être ignorée et donc non importées.
- **(column_or_user_var,...)**
- **SET**

Reportez vous à la syntaxe de  **LOAD DATA INFILE** dans le manuel de MySQL 5.1 pour plus d'informations sur ces clauses.

ROWS IDENTIFIED BY '<tagname>'

LOAD XML supporte 3 formats XML différents :

- Les attributs sont interprétés en tant que noms de colonnes, et leurs valeurs en tant que valeurs des colonnes

```
<row column1="value1" column2="value2" .../>
```

- Les noms des balises sont interprétés comme noms de colonnes, et le contenu de ces balises en tant que valeurs des colonnes

```
<row>
  <column1>value1</column1>
  <column2>value2</column2>
</row>
```

- Les noms de colonnes d'une table dérivent de l'attribut name de la balise <fields>, et les valeurs des colonnes sont prises dans le contenu de ces balises.

```
<row>
  <field name='column1'>value1</field>
  <field name='column2'>value2</field>
</row>
```

Ce dernier est celui utilisé par l'outil MySQL **mysqldump** .

Les routines d'import utilisées par **LOAD XML** détectent automatiquement le format utilisé pour chaque ligne et l'interprète correctement, faisant correspondre les noms basés sur les balises ou les attributs à ceux des colonnes. Vous pouvez facilement vérifier cela par vous-même en créant un fichier XML qui utilise un mix des formats cités précédemment avec la commande **LOAD XML** pour l'importer dans une table.

 Vous devez avoir les privilèges **FILE** de MySQL pour utiliser **LOAD XML**.

II - Les fonctionnalités XPath

Dans cette section, nous voyons les fonctionnalités XPath ajoutées dans MySQL 5.1.5.

ExtractValue() vous autorise à utiliser une expression XPath sur un fragment de XML dans le but de retourner le contenu d'un ou plusieurs éléments. **UpdateXML()** rend cela possible en remplaçant un fragment de XML existant par un nouveau, en utilisant XPath pour préciser le fragment à remplacer.

II-1 - La fonction ExtractValue()

L'exemple dans 'En utilisant **LOAD FILE()**' précédent démontre comment récupérer le contenu d'un fichier XML et le réinsérer dans une base, mais le problème d'obtention des données reste réel.

Une manière d'y arriver est d'utiliser la fonction **ExtractValue()**. La syntaxe de cette fonction est montrée ici :

Syntaxe de ExtractValue

```
ExtractValue(xml_fragment, xpath_expression)
```

ExtractValue() prends 2 arguments. Le premier est le fragment de XML à tester, le deuxième est l'expression XPath à trouver.

Exemple : Regardons comment nous pouvons obtenir le nom de la ville dans ce document (que nous avons créé précédemment : 2cities.xml)

Nous procédons en 2 étapes : d'abord nous prenons le XML depuis la table **xmldocs** et on la place dans une variable utilisateur.

Commande MySQL

```
mysql> SELECT doc_content FROM xmldocs LIMIT 1 INTO @xml;
Query OK, 1 row affected (0.00 sec)

mysql> SELECT @xml\G
```

Sortie XML générée

```
***** 1. row *****
@xml: <?xml version="1.0"?>

<resultset statement="SELECT name,country FROM cities LIMIT 2
" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
    <field name="name">Mumbai (Bombay)</field>
    <field name="country">India</field>
  </row>

  <row>
    <field name="name">Seoul</field>
    <field name="country">South Korea</field>
  </row>
</resultset>

1 row in set (0.00 sec)
```

Maintenant nous utilisons **ExtractValue()** avec @xml en tant que premier argument. Pour le deuxième argument, nous employons une expression XPath qui veut dire "le contenu du premier élément **<field >** inclus dans le deuxième élément **<row >** trouvé n'importe où dans le document ".

ExtractValue() prends 2 arguments. Le premier est le fragment de XML à tester, le deuxième est l'expression XPath à trouver.

Exemple : Regardons comment nous pouvons obtenir le nom de la ville dans ce document (que nous avons créé précédemment : 2cities.xml)

Nous procédons en 2 étapes : d'abord nous prenons le XML depuis la table **xmldocs** et on la place dans une variable utilisateur.

Commande MySQL

```
mysql> SELECT ExtractValue(@xml, '//row[2]/field[1]');
```

Sortie XML générée

```
+-----+
| ExtractValue(@xml, '//row[2]/field[1]') |
+-----+
| Seoul          |
+-----+
1 row in set (0.00 sec)
```

Un autre moyen de faire cela serait d'utiliser une expression XPath qui veut dire : " le contenu du second **<field >** dont l'attribut **name** est 'name' ".

Commande MySQL

```
mysql> SELECT ExtractValue(@xml, '//field[@name="name"][2]');
```

Sortie XML générée

```
+-----+
| ExtractValue(@xml, '//field[@name="name"][2]') |
+-----+
| Seoul          |
+-----+
1 row in set (0.00 sec)
```

Comme vous pouvez le voir, le résultat (**Seoul**) est le même que tout à l'heure, ce qui est exactement ce que nous voulions.

II-2 - La fonction UpdateXML()

Cette fonction vous permet de remplacer une portion d'un fragment XML (identifié par un localisateur XPath) avec différentes balises XML.

Par exemple, prenez ce fragment XML **<book><chapter/></book>** .

Maintenant supposez que vous voulez le transformer en **<book><part><chapter/></part></book>** cet exemple vous montre comment vous pouvez le faire avec **UpdateXML()** en sauvegardant le resultat dans une variable utilisateur **@new_xml**.

Commande MySQL

```
mysql> SELECT @new_xml:=UpdateXML('<book><chapter/></book>',
->    '//chapter',
->    '<part><chapter/></part>')
-> AS uxml;
```

Sortie XML générée

```
+-----+
| uxml          |
+-----+
| <book><part><chapter/></part></book> |
+-----+
```

Sortie XML générée

```
+-----+
| 1 row in set (0.00 sec)
```

Commande MySQL

```
mysql> SELECT @new_xml;
```

Sortie XML générée

```
+-----+
| @new_xml |
+-----+
| <book><part><chapter/></part></book> |
+-----+
1 row in set (0.00 sec)
```

La syntaxe pour cette fonction est montrée ici :

Syntaxe de UpdateXML

```
UpdateXML(xml, locator, replacement)
```

Xml est la balise XML à tester pour la correspondance.

locator est l'expression XPath utilisée pour obtenir la correspondance.

replacement est la balise à utiliser pour remplacer le XML qui correspond à **locator**.

Contrairement à **ExtractValue()**, **UpdateXML()** récupère les éléments et les contenus et ensuite renvoi le tout sous forme de chaîne de caractères. S'il n'y a aucune correspondance pour **locator**, le XML original est retourné.

II-3 - Le support de XPath dans MySQL 5.1 et 6.0

Dans cette section nous discutons des fonctionnalités XPath fournies avec les fonctions **ExtractValue()** et **UpdateXML()** dans MySQL 5.1 et 6.0.

Les fonctions et les opérateurs supportés

Les expressions XPath basiques (connues comme repères) sont supportées, ceci inclut les repères utilisant les opérateurs suivants :

- L'opérateur **/ (slash)**, cet opérateur agit de la même manière que sont homologues dans les chemins de fichiers systèmes UNIX, quand le premier **/** représente la racine du document et l'identifiant suivant est le nom de l'élément XML.

Par exemple, **/book** correspond au premier élément **book**.

De multiples slashes peuvent être utilisées pour retracer une branche.

Par exemple **/book/chapter** correspond à **<book><chapter/></book>** où **book** est le premier élément et **chapter** est un fils de cet élément.

Un double slash (**//**) en entête que le pattern correspond n'importe où dans le document.

Par exemple le repère **//section** correspond à l'élément **section** partout où il se trouve, quelque soit sa relation avec les autres éléments du document.
- L'opérateur **| union**. Cet opérateur peut être utilisé quand vous voulez combiner les repères quand vous voulez faire correspondre n'importe lesquels d'entre eux.

Par exemple **//section//paragraph** correspond à n'importe quel élément **section** ou **paragraph**, n'importe où dans le document.
- L'opérateur *** wildcard** : cet opérateur correspond à n'importe quel élément.

Par exemple, **//chapter/*** correspond à n'importe quel élément enfant de **chapter**, n'importe où dans le document et ***/section** correspond à n'importe quel élément **section** enfant de l'élément le plus haut.

- Correspondance d'attribut : vous pouvez trouver un élément par la valeur de un ou plusieurs de ses attributs en utilisant la syntaxe **element [@ attribute = value]**.
Par exemple **//section[@id='xpath-locators']** correspond à un élément **section** n'importe où dans le document qui possède un attribut **id** avec la valeur **xpath-locators**.

NOTE

Vous pouvez trouver de multiples valeurs d'attributs en les combinant simplement.

Par exemple, le repère **//paragraph[@role='intro'][@title='xpath support']** correspond à n'importe quel élément **paragraph** (n'importe où dans le document XML à tester) qui a un attribut **role** dont la valeur est **intro** et un attribut **title** dont la valeur est **xPath-support**.

Pour trouver des éléments pour lesquels aux mêmes attributs correspondent une des multiples valeurs, vous pouvez utiliser plusieurs repères joints par l'opérateur union |.

Par exemple, pour trouver tout les éléments **paragraph**, n'importe où dans le document XML à tester dont l'attribut **title** a l'une des valeurs **Example** ou **Syntax**, vous utiliserez l'expression **//paragraph[@title="Example"]//paragraph[@title="Syntax"]** Vous pouvez aussi utiliser l'opérateur logique **or** pour cette expression : **//paragraph[@title="Example" or @title="Syntax"]**

La différence entre **or** et **|** est que **or** joint les conditions alors que **|** joint les jeux de résultat.

- Les espaces de noms ne sont pas explicitement supportés. Quoiqu'il en soit les noms d'éléments contenant : sont autorisés, donc vous pouvez travailler avec des balises XML qui utilisent la notation d'espaces de noms. Par exemple **//person:biography** correspond au tag **<person:biography/>** n'importe où dans le document.
- Tous les opérateurs standards de comparaison XPath (or, and, =, !=, <=, <, >=, and >) sont supportés , Par exemple considérez ce qui suit :

```
mysql> SET @xml = '<foo bar="2">123</foo><foo bar="6">456</foo>';
```

Sortie XML générée

```
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT ExtractValue(@xml, '//foo[@bar="2"]');
```

Sortie XML générée

```
+-----+
| extractvalue(@xml, '//foo[@bar="2"]') |
+-----+
| 123          |
+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT ExtractValue(@xml, '//foo[@bar>"2"]');
```

Sortie XML générée

```
+-----+
| extractvalue(@xml, '//foo[@bar>"2"]') |
+-----+
| 456          |
+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT ExtractValue(@xml, '//foo[@bar>="2"]');
```

Sortie XML générée

```
+-----+
| extractvalue(@xml, '//foo[@bar>="2"]') |
+-----+
| 123 456          |
+-----+
```

Sortie XML générée

```
+-----+
| 1 row in set (0.01 sec)
```

Notez que quand plusieurs correspondances sont trouvées, **ExtractValue()** les retourne dans un seul espace délimité.

- Vous pouvez aussi utiliser l'indexation dans des repères pour identifier les éléments particuliers d'intérêt, comme on l'a vu précédemment pour **//row[2]/field[1]**.

ExtractValue(), comme son nom l'indique, obtient une valeur, il ne retourne aucun élément XML. Voici un exemple qui devrait rendre cela plus clair :

Commande MySQL

```
SET @xml = '<a>3<b><c>5<d/></c></b></a>';
```

Sortie XML générée

```
Query OK, 0 rows affected (0.00 sec)
```

Commande MySQL

```
mysql> SELECT ExtractValue(@xml, '//a'), ExtractValue(@xml, '//b'), ExtractValue(@xml, '//c');
```

Sortie XML générée

```
+-----+-----+-----+
| ExtractValue(@xml, '//a') | ExtractValue(@xml, '//b') | ExtractValue(@xml, '//c') |
+-----+-----+-----+
| 3 | | 5 |
+-----+-----+-----+
1 row in set (0.01 sec)
```

ExtractValue(@xml, '//a') retourne seulement le contenu de n'importe quel élément **<a>**, il ne retourne rien des éléments ****, **<c>**, ou **<d>**, ou rien de leur contenus. **ExtractValue(@xml, '//c')** retourne seulement le contenu de n'importe quel élément **<c>**, car le seule élément **** ne contient pas de texte mais seulement d'autres éléments XML. **ExtractValue(@xml, '//b')** retourne une chaîne vide.

Les fonctions et les opérateurs non supportés

Beaucoup d'opérateurs et de fonctions XPath sont supportés.

Quoi qu'il en soit le support de Xpath dans MySQL est toujours en développement et donc certains aspects ne sont pas encore implémentés, ceci inclut les limitations suivantes.

- Les expressions de repères relatifs sont résolues dans le contexte du n#ud de racine, par exemple considérez la requête suivante et son résultat :

Commande MySQL

```
mysql> SELECT ExtractValue(
-> '<book>
-> <paragraph title="Example">P1</paragraph>
-> <paragraph title="Syntax">P2</paragraph>
-> </book>',
-> 'book/paragraph'
```

Commande MySQL

```
-> ) AS paras;
```

Sortie XML générée

```
+-----+
| paras |
+-----+
| P1 P2 |
+-----+
1 row in set (0.03 sec)
```

Dans ce cas le repère **book/paragraph** est résolu en **/book/paragraph**.

Les repères relatifs sont aussi supportés dans les prédicats. Dans l'exemple suivant **/@title="Example"** est résolu en **/book/paragraph/@title="Example"**:

Commande MySQL

```
mysql> SELECT ExtractValue(
->     '<book>
->     <paragraph title="Example"><body>P1</body></paragraph>
->     <paragraph title="Syntax"><body>P2</body></paragraph>
->     </book>',
->     'book/paragraph/body[../@title="Example"]')
-> AS para;
```

Sortie XML générée

```
+-----+
| para |
+-----+
| P1 |
+-----+
1 row in set (0.00 sec)
```

- L'opérateur **::** n'est pas supporté en combinaison avec les n#uds du type **axis ::comment()**, **axis ::text()**, **axis ::processing-instructions()**, et **axis ::node()**.
Quoi qu'il en soit les tests nommés (comme **axis :: name** et **axis ::***) sont supportés , comme montré dans l'exemple.

Commande MySQL

```
mysql> SELECT ExtractValue('<a><b>x</b><c>y</c></a>', '/a/child::b');
```

Sortie XML générée

```
+-----+
| ExtractValue('<a><b>x</b><c>y</c></a>', '/a/child::b') |
+-----+
| x |
+-----+
1 row in set (0.02 sec)
```

Commande MySQL

```
mysql> SELECT ExtractValue('<a><b>x</b><c>y</c></a>', '/a/child::*');
```

Sortie XML générée

```
+-----+
| ExtractValue('<a><b>x</b><c>y</c></a>', '/a/child::*') |
+-----+
| x y |
+-----+
```

Sortie XML générée

```
1 row in set (0.01 sec)
```

- La navigation "Up-and-down" n'est pas supportées dans les cas où le chemin mènerait "au-dessus" de l'élément de racine. C'est pourquoi vous ne pouvez pas utiliser les expressions qui correspondent sur les descendants d'un ancêtre d'un élément donné où un ou plus des ancêtres de l'élément actuel sont aussi un ancêtre de l'élément de racine (cfBug #16321).
- Les fonctions XPath suivantes ne sont pas supportées : **id()**, **lang()**, **local-name()**, **name()** (un patch a récemment été soumis pour implémenter la fonction, il devrait être effectif dans la version 6.0) **namespace-uri()**, **normalize-space()**, **starts-with()**, **string()**, **substring-after()**, **substring-before()**, et **translate()**.
- Les axes suivants ne sont pas supportés : **following-sibling**, **following**, **preceding-sibling**, et **preceding**.

Traitement d'erreur

Pour **ExtractValue()** et **UpdateXML()**, le repère XPath utilisé doit être valide et le XML doit être bien formé. Si le repère est invalide, une erreur est générée :

Commande MySQL

```
mysql> SELECT ExtractValue('<foo bar="2">123</foo>', '//foo[@bar>="2"]');
```

Sortie XML générée

```
+-----+
| ExtractValue('<foo bar="2">123</foo>', '//foo[@bar>="2"]') |
+-----+
| 123          |
+-----+
1 row in set (0.00 sec)
```

Commande MySQL

```
mysql> SELECT ExtractValue('<foo bar="2">123</foo>', '//foo@bar>="2"]');
```

Sortie XML générée

```
ERROR 1105 (HY000): XPATH syntax error: '@bar>="2"]'
```

Si le XML est mal formé, la valeur NULL est retournée, et un avertissement s'affiche.

Commande MySQL

```
mysql> SELECT ExtractValue('<foo bar="2"123</foo>', '//foo[@bar>="2"]');
```

Sortie XML générée

```
+-----+
| ExtractValue('<foo bar="2"123</foo>', '//foo[@bar>="2"]'); |
+-----+
| NULL          |
+-----+
1 row in set, 1 warning (0.00 sec)
```

Commande MySQL

```
mysql> SHOW WARNINGS;
```

Sortie XML générée

Sortie XML générée

```
+-----+-----+
+-----+-----+
| Level | Code | Message |
+-----+-----+
| Warning | 1522 | Incorrect XML value: 'parse error at line 1 pos 13: unknown token unexpected ('>'
wanted)' |
+-----+-----+
+-----+-----+
1 row in set (0.00 sec)
```

Quoiqu'il en soit, le XML de remplacement utilisé comme 3eme argument de la fonction **UpdateXML()** n'est pas vérifié pour la bonne-formation du document.

NOTE

L'espace ne permet pas une discussion complète de la syntaxe XPath et de son utilisation. Pour de plus amples informations rendez vous sur : [XML Path Language \(XPath\) 1.0 standard](#). La page [Zvon.org XPath Tutorial](#) fournit aussi des ressources utiles pour ceux qui débutent dans XPath ou qui veulent se rafraichir les bases de XPath.

Les variables utilisateur dans les expressions XPath

Depuis MySQL 5.1.20, vous pouvez employer les variables utilisateurs dans les repères XPath sous n'importe quelle forme :

- Légèrement vérifiée** : une variable qui utilise la syntaxe **\$@ variable_name** ne voit pas son type vérifié ni si elle contient une valeur et dans ce cas aucune erreur n'est renvoyée par le serveur.
 En d'autres termes vous êtes responsables des vérifications adéquates et des fautes de frappes lors de l'appel des variables.
 Par exemple si vous utilisez **\$@myvairable** au lieu de **\$@myvariable** et que **\$@myvairable** n'a pas de valeur alors MySQL pense que **\$@myvairable** n'a aucune valeur de type approprié .

Commande MySQL

```
mysql> SET @xml = '<a><b>X</b><b>Y</b></a>';
```

Sortie XML générée

```
Query OK, 0 rows affected (0.00 sec)
```

Commande MySQL

```
mysql> SET @i =1; @j = 2;
```

Sortie XML générée

```
Query OK, 0 rows affected (0.00 sec)
```

Commande MySQL

```
mysql> SELECT @i, ExtractValue(@xml, '//b[$@i]');
```

Sortie XML générée

```
+-----+-----+
| @i | ExtractValue(@xml, '//b[$@i]') |
+-----+-----+
| 1 | X |
+-----+-----+
```


Sortie XML générée

```

| xml          | i | ExtractValue(xml, '//a[$i]') |
+-----+-----+-----+
| <a>X</a><a>Y</a><a>Z</a> | 1 | X      |
+-----+-----+-----+
1 row in set (0.00 sec)

+-----+-----+-----+
| xml          | i | ExtractValue(xml, '//a[$i]') |
+-----+-----+-----+
| <a>X</a><a>Y</a><a>Z</a> | 2 | Y      |
+-----+-----+-----+
1 row in set (0.01 sec)

+-----+-----+-----+
| xml          | i | ExtractValue(xml, '//a[$i]') |
+-----+-----+-----+
| <a>X</a><a>Y</a><a>Z</a> | 3 | Z      |
+-----+-----+-----+
1 row in set (0.01 sec)
    
```

Les expressions contenant les variables définies par l'utilisateur n'importe quelle sorte doivent (à part la notation) se conformer aux règles des expressions XPath contenant des variables comme décrit dans les spécifications XPath.

III - Considérations de Sécurité

Avec n'importe quelle fonction pour les bases, vous devez être conscient des implications de sécurité, de travailler sur les Xml et les expressions Xpath n'est pas différent à cet égard. Il y a de nombreux points à considérer, incluant la possibilité de lire depuis un fichier en utilisant **LOAD XML INFILE**.

Le fait que le système de privilège MySQL ne s'applique pas au contenu du fichier XML, et les possibilités pour les saisies des utilisateurs subversif d'avoir des conséquences fortuites.

III-1 - Charger des données depuis un fichier

Comme avec la commande LOAD DATA le transfert du fichier XML depuis le poste client jusqu'au serveur est initié par le serveur MySQL. En théorie un serveur patché devrait être chargé de dire au programme client de charger un fichier choisit par le serveur plutôt que par l'utilisateur dans la commande **LOAD XML**.

Un tel serveur pourrait avoir accès à n'importe quel fichier sur le client à qui le client donne l'accès en lecture.

Dans un environnement web, les clients se connectent habituellement depuis le serveur web. Un utilisateur qui peut exécuter n'importe quelle commande sur le serveur MySQL peut utiliser **LOAD XML LOCAL**, peut lire n'importe quel document pour lequel le serveur web a les accès en lecture.

Dans cet environnement, le client en ce qui concerne le serveur MySQL est en réalité le serveur Web, pas le programme distant étant exécuté par l'utilisateur qui se connecte au serveur du Web.

Vous pouvez désactiver **LOAD XML** sur le serveur en le démarrant avec **--local-infile=0** ou **--local-infile=OFF**. Le résultat est montré dans cet exemple.

Commande MySQL

```

shell> mysqld_safe --local-infile=OFF &

shell> mysql -uroot xtest
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 6.0.4-alpha-debug Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
    
```

Commande MySQL

```
mysql> LOAD XML LOCAL INFILE '/home/jon/person.xml'
-> INTO TABLE person
-> ROWS IDENTIFIED BY '<person>';
ERROR 1148 (42000): The used command is not allowed with this MySQL version
```

Cette option peut aussi être utilisée quand vous démarrez le client MySQL pour désactiver **LOAD XML** le temps de la session utilisateur.

Pour prévenir un client contre le chargement d'un fichier XML sur le serveur, ne donnez pas les droits **FILE** à l'utilisateur MySQL correspondant ou révoquez les droits si l'utilisateur les possède déjà.

Commande MySQL

```
shell> mysql -uroot -p
Password: *****
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 6.0.4-alpha-debug Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.


mysql> REVOKE FILE ON *.* FROM jon@localhost;
Query OK, 0 rows affected (0.00 sec)

mysql> exit
Bye
shell> mysql -ujon -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 6.0.4-alpha-debug Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> USE xmltest;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

mysql> LOAD XML INFILE '/home/jon/person.xml'
-> INTO TABLE person
-> ROWS IDENTIFIED BY '>person>';
ERROR 1045 (28000): Access denied for user 'jon'@'localhost' (using password: YES)
```

 **Révoquer les droits FILE (ou ne pas les accorder dans un premier temps) protège seulement l'utilisateur de l'exécution de la commande **LOAD XML INFILE** ou de la fonction **LOAD_FILE()**.**

*Cela ne protège pas l'utilisateur d'exécuter la commande **LOAD XML LOCAL INFILE**. Pour désactiver cet état vous devez démarrer le serveur ou le client avec l'option **--local-infile=OFF** comme vu précédemment.*

En d'autres termes, le droit FILE affecte seulement si le client peut lire des fichiers sur le serveur, il n'a aucun comportement sur le client si il peut lire des fichiers sur le système en local.

III-2 - Les privilèges XML et MySQL

La granularité trouvée dans le système de privilèges MySQL en ce qui concerne la plupart des objets de base de données ne s'étend pas au document XML. Un client MySQL qui a les accès à un document XML peut accéder à tout le document ou à n'importe quelle partie de celui-ci et il n'y a aucun moyen de restreindre l'accès à certaines parties du XML ou certains éléments.

Gardez à l'esprit que MySQL n'a pas de privilèges préventifs, ce qui veut dire, que vous ne pouvez restreindre l'accès de type d'objets particulier dans une base de données.

À la place, si vous voulez empêcher un client d'accéder à un XML trouvé dans une table, vous devez changer les droits pour que cet utilisateur n'est plus accès du tout à cette table, ou au moins les colonnes de cette table qui contient les données XML. XPath ne fait pas d'exceptions à cet égard.

III-3 - Injection XPath

Une des menaces de sécurité les plus grandes aux applications est l'injection de code, le code malveillant introduit dans le système pour gagner l'accès non autorisé aux privilèges et aux données.

Elles sont connues pour exister dans plusieurs langages de programmation et de scripting.

Ce qu'ils ont tous en commun est l'exploitation des suppositions faites par des développeurs sur le type et le contenu des saisies des utilisateurs.

XPath ne fait pas d'exceptions à cet égard.

Supposons que votre application donne des autorisations en trouvant la combinaison login et password dans le fichier XML **users.xml** dont le contenu est montré ici :

Fichier users.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<users>
  <user id="00327">
    <login>douglas42</login>
    <password>24ph0d</password>
  </user>
  <user id="13579">
    <login>cherrygarcia</login>
    <password>1c3cr34m</password>
  </user>
  <user id="02403">
    <login>jimbo</login>
    <password>p4nc4k35</password>
  </user>
  <user id="42354">
    <login>kitten</login>
    <password>m3330w</password>
  </user>
  <user id="28570">
    <login>lucyvanpelt</login>
    <password>f0076411</password>
  </user>
</users>
```

Supposons que chaque combinaison login / password est unique, votre application peut utiliser une expression XPath comme celle-ci pour valider l'utilisateur, commencer une session et associer la session à l'ID unique de l'utilisateur

```
//user[login/text()='cherrygarcia' and password/text()='1c3cr34m']/attribute::id
```

Ceci est l'équivalent d'une commande SQL comme celle-ci :

```
SELECT id FROM users WHERE login='cherrygarcia' AND password='1c3cr34m';
```

Une application PHP employant XPath et le fichier users.xml devrait utiliser le processus de login via un formulaire comme ceci :

```
<?php
```

```

$file      = "users.xml";

$login     = $POST["login"];
$password  = $POST["password"];

$xmlpath = "//user[login/text()=$login and password/text()=$password]/attribute::id";

if( file_exists($file) )
{
    $xml = simplexml_load_file($file);

    if($result = $xml->xpath($xmlpath))
        echo "You are now logged in as user $result[0].";
    else
        echo "Invalid login name or password.";
}
else
    exit("Failed to open $file.");

?>
    
```

L'entrée n'est pas du tout vérifiée, ce qui veut dire qu'un utilisateur malveillant peut court-circuiter le test en entrant ' or 1=1 pour le login et le password pour que l'expression XPath soit évaluée comme ceci :

```

//user[login/text()=' ' or 1=1 and password/text()=' ' or 1=1]/attribute::id
    
```

L'expression entre crochets est évaluée à true et est donc effectivement la même que celle-ci qui cherche l'attribut **id** de tout les éléments **user** du Xml.

```

//user/attribute::id
    
```

Une manière de contrer simplement cette attaque est de "quoting" les noms des variables pour les interpoler dans la définition de **\$xmlpath**

```

$xmlpath = "//user[login/text()=' $login ' and password/text()=' $password ']/attribute::id";
    
```

Si cette technique semble familière, c'est parce que c'est celle généralement conseillée pour se prémunir contre les injections SQL.

En générale les pratiques pour éviter les injections XPath sont les mêmes que pour les injections SQL :

- Ne jamais accepter les données non testées d'un utilisateur dans votre application.
- Vérifier le type de données utilisateurs, rejeter ou convertir les données de type erroné.
- Tester les valeurs numériques pour les valeurs hors fourchette, tronquer, arrondir ou rejeter les valeurs. Tester les chaînes pour les caractères illégaux et supprimez-les ou rejetez les chaînes les contenant.
- Ne renvoyez pas de messages d'erreur spécifiques, qui pourraient indiquer à un utilisateur des indices sur le façon de compromettre votre système. Logger les dans un fichier ou une table à la place.

Juste comme les injections SQL peuvent être utilisées pour obtenir des informations sur le schéma des tables, les injections Xpath peuvent être utilisées pour traverser les fichiers XML et découvrir leurs structures. L'espace ne me permet pas d'aller plus loin mais vous pouvez approfondir vos connaissances sur le sujet avec le papier d'**Amit Klein**

Blind XPath Injection

C'est aussi important de vérifier les sorties envoyées au client. Par exemple regardons notre exemple précédent, cette fois, au lieu d'utiliser les fonctionnalités XPath de PHP, on utilise la fonction MySQL **ExtractValue()**

```

Commande MySQL
    
```

Commande MySQL

```
mysql> SELECT ExtractValue(
->     LOAD_FILE('users.xml'),
->     '//user[login/text()="" or 1=1 and password/text()="" or 1=1]/attribute::id'
-> ) AS id;
```

Sortie XML générée

```
+-----+
| id |
+-----+
| 00327 13579 02403 42354 28570 |
+-----+
1 row in set (0.01 sec)
```

Parce que **ExtractValue()** retourne des correspondances multiples dans un seul espace délimité , cette injection fournit à l'utilisateur dans une seule ligne tout les ID valides contenus dans **users.xml** .
Vous devriez donc tester la sortie avant de l'envoyer à l'utilisateur, par exemple :

Commande MySQL

```
mysql> SELECT @id = ExtractValue(
->     LOAD_FILE('users.xml'),
->     '//user[login/text()="" or 1=1 and password/text()="" or 1=1]/attribute::id'
-> );
```

Sortie XML générée

```
Query OK, 0 rows affected (0.00 sec)
```

Commande MySQL

```
mysql> SELECT IF(
->     INSTR(@id, ' ') = 0,
->     @id,
->     'Unable to retrieve user ID')
-> AS singleID;
```

Sortie XML générée

```
+-----+
| singleID |
+-----+
| Unable to retrieve user ID |
+-----+
1 row in set (0.00 sec)
```

En générale la ligne de conduite pour retourner des données à l'utilisateur de façon sécurisée sont les mêmes que lors de l'acceptation de données de la part de celui-ci.
On peut les résumer :

- Toujours tester les données en entrée pour le type et la valeur.
- Ne jamais autoriser les utilisateurs à voir des messages qui pourraient compromettre la sécurité de votre système.

Suivre ces principes pour construire une application XML peut aider pour assurer la sécurité de celle-ci .