www.mas-ict.ch

Hes·so
Haute Ecole Spécialisée de Suisse occidentale

Module: XML et les bases de données

Xupdate
(W3C Recommendation 17 March 2011)

Houda Chabbi Drissi

houda.chabbi@hefr.ch

http://www.w3.org/TR/xquery-update-10/
http://www-rocq.inria.fr/~abitebou/Master-SSD/slxqupdate.pdf
http://www.xmlmind.com/_tutorials/XQueryUpdate/

Source:
http://monetdb.cwi.nl/projects/monetdb/XQuery/QuickTour/XQUF/index.html

Greetings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<doc>
   <greet kind="informal">Hi </greet>
   <greet kind="casual">Hello </greet>
   <location kind="global">World</location>
   <location kind="local">Amsterdam</location>
</doc>
```

Source:
http://monetdb.cwi.nl/projects/monetdb/XQuery/QuickTour/XQUF/index.html

Plan

- Introduction
- Extension de Xquery
- XQuery Update Processing Model
- Le mixe entre les expressions Updating et non updating
- Exercices

Introduction

- Comme dans tout langage de requête de bases de donnée, nous avons besoin de modifier/oter les données stockées dans la base.

- **XQuery** est la partie requête (~ select) pas de modification possible de l'instance requêtée.

- **XQuery Update** introduits 5 nouvelles opérations:
  - **insert, delete, replace, rename** → réalise des modifications
  - **Transform** → sans modifications

Plan

- Introduction
- Extension de Xquery:
  - insert, delete, replace, rename, transform
- XQuery Update Processing Model
- Le mixe entre les expressions Updating et non updating
- Exercices

Extension de l'expression de XQuery

XQuery Expr :=Constants | Variable | FunctionCalls |
PathExpr | ComparisonExpr | ArithmeticExpr |
LogicExpr | FLWRExpr | ConditionalExpr |
QuantifiedExpr | TypeSwitchExpr |
InstanceofExpr | CastExpr | UnionExpr |
IntersectExceptExpr | ConstructorExpr |
ValidateExpr | InsertExpr| DeleteExpr
| RenameExpr | ReplaceExpr | TransformExpr

L'expression de XQuery

XQuery Expr :=Constants | Variable | FunctionCalls |
PathExpr | ComparisonExpr | ArithmeticExpr |
LogicExpr | FLWRExpr | ConditionalExpr |
QuantifiedExpr | TypeSwitchExpr |
InstanceofExpr | CastExpr | UnionExpr |
IntersectExceptExpr | ConstructorExpr |
ValidateExpr | InsertExpr| DeleteExpr
| RenameExpr | ReplaceExpr | TransformExpr

Syntaxe du Insert

```
InsertExpr::=insert (node|nodes)
             SourceExpr
             ((as (first|last))? into)
              | after | before
             TargetExpr
```

SourceExpr::= must be a simple expression so any sequence of items (nodes, values)

TargetExpr::= exactly one document or element

Exemple 1

Insérer a nouvel élément <last/> comme dernier fils de <doc>:

```
<?xml version="1.0" encoding="utf-8"?>
<doc>
   <greet kind="informal">Hi </greet>
   <greet kind="casual">Hello </greet>
   <location kind="global">World</location>
   <location kind="local">Amsterdam</location>
   <last/>
</doc>
```

```
insert node
      <last/>
as last into
      doc("greetings.xml")/doc
```

Exemple 2

Soit $target := <CONT/>

Que donne

```
insert nodes
        (attribute A { 2.1 }, <child1/>, "text", 1 to 3)
into
        $target
```

?????

→ Applications des même règles que celles pour les constructeurs

Exemple 2

Soit $target := <CONT/>

Que donne

```
insert nodes
        (attribute A { 2.1 }, <child1/>, "text", 1 to 3)
into
        $target
```

<CONT A="2.1"><child1/>text 1 2 3</CONT>

→ Applications des même règles que celles pour les constructeurs

Exemple 3

Insérer un nouvel élément "greet" entre les 2 autres:

```
<?xml version="1.0" encoding="utf-8"?>
<doc>
   <greet kind="informal">Hi </greet>
   <greet kind="formal">Good day</greet>
   <greet kind="casual">Hello </greet>
   <location kind="global">World</location>
   <location kind="local">Amsterdam</location>
   <last/>
</doc>
```

insert node

```
<greet kind="formal">Good day</greet>
```

before

```
doc("greetings.xml")//greet[2]
```

L'expression de XQuery

XQuery Expr :=Constants | Variable | FunctionCalls |
PathExpr | ComparisonExpr | ArithmeticExpr |
LogicExpr | FLWRExpr | ConditionalExpr |
QuantifiedExpr | TypeSwitchExpr |
InstanceofExpr | CastExpr | UnionExpr |
IntersectExceptExpr | ConstructorExpr |
ValidateExpr | InsertExpr| DeleteExpr
| RenameExpr | ReplaceExpr | TransformExpr

Syntaxe du delete

delete one or several nodes

```
deleteExpr ::= delete (node|nodes)
               TargetExpr
```

TargetExpr::= Any sequence of nodes that must have parents

Delete: Exemple

Enlever l'élément <last/>

```
<?xml version="1.0" encoding="utf-8"?>
<doc>
   <greet kind="informal">Hi </greet>
   <greet kind="formal">Good day</greet>
   <greet kind="casual">Hello </greet>
   <location kind="global">World</location>
   <location kind="local">Amsterdam</location>
   <last/>
</doc>
```

delete node

```
doc("greetings.xml")//last
```

L'expression de XQuery

XQuery Expr :=Constants | Variable | FunctionCalls |
PathExpr | ComparisonExpr | ArithmeticExpr |
LogicExpr | FLWRExpr | ConditionalExpr |
QuantifiedExpr | TypeSwitchExpr |
InstanceofExpr | CastExpr | UnionExpr |
IntersectExceptExpr | ConstructorExpr |
ValidateExpr | InsertExpr| DeleteExpr
| RenameExpr | ReplaceExpr | TransformExpr

Replace

L'identité du nœud cible est préservé. Seule sa valeur ou son contenu (pour un élément ou un document) est remplacé.

Deux variantes:

- remplacer un nœud (et tous ses descendants) par une séquence de nœuds.
- remplacer le contenu (les enfants) d'un nœud avec une séquence de nœuds, ou la valeur d'un nœud avec une valeur de chaîne.

Syntaxe du replace

```
ReplaceExpr::=
        replace (value of)? Node
        TargetExpr
        with ExprSingle
```

TargetExpr::= One node (with ID)

1. If **value of is not specified**, a replace expression replaces one node with a new sequence of zero or more nodes.

2. If **value of is specified**, a replace expression is used to modify the value of a node while **preserving its node identity**.

Exemple 1: remplacement de nœuds

Remplacer le deuxième nœud par

<greet kind="polite">Pleased to meet you</greet> :

```
<?xml version="1.0" encoding="utf-8"?>
<doc>
   <greet kind="informal">Hi </greet>
   <greet kind="polite"> Pleased to meet you </greet>
   <greet kind="casual">Hello </greet>
   <location kind="global">World</location>
   <location kind="local">Amsterdam</location>
   <last/>
</doc>
```

replace node

doc("greetings.xml")//greet[2]

with

<greet kind="polite">Pleased to meet you</greet>

Exemple 2: remplacement de valeur

Remplacer les valeur du deuxième nœuds par

polite →impolite et Pleased to meet you→@!!@*#:

```
<?xml version="1.0" encoding="utf-8"?>
<doc>
   <greet kind="informal">Hi </greet>
   <greet kind="impolite"> Pleased to meet you </greet>
   <greet kind="casual">Hello </greet>
   <location kind="global">World</location>
   <location kind="local">Amsterdam</location>
   <last/>
</doc>
```

replace value of node

doc("greetings.xml")//greet[2]/@kind

with

"impolite"

Exemple 2: remplacement de valeur

Remplacer les valeur du deuxième nœuds par

polite →impolite et Pleased to meet you→@!!@*#:

```
<?xml version="1.0" encoding="utf-8"?>
<doc>
   <greet kind="informal">Hi </greet>
   <greet kind="impolite"> @!!@*# </greet>
   <greet kind="casual">Hello </greet>
   <location kind="global">World</location>
   <location kind="local">Amsterdam</location>
   <last/>
</doc>
```

replace value of node

doc("greetings.xml")//greet[2]

with

" @!!@*# "

L'expression de XQuery

XQuery Expr :=Constants | Variable | FunctionCalls |
PathExpr | ComparisonExpr | ArithmeticExpr |
LogicExpr | FLWRExpr | ConditionalExpr |
QuantifiedExpr | TypeSwitchExpr |
InstanceofExpr | CastExpr | UnionExpr |
IntersectExceptExpr | ConstructorExpr |
ValidateExpr | InsertExpr| DeleteExpr
| RenameExpr | ReplaceExpr | TransformExpr

Syntaxe du rename

Renomme un nœud (applicable aux éléments, attributs et instructions de traitement) sans affecter son contenu ou ses attributs.

```
RenameExpr::= rename node TargetExpr
              as NewNameExpr
```

TargetExpr::= must be one element, attribute, or PI

NewNameExpr::= must be an expression that evaluates to a QName (or castable)

Exemple

Renommer le 2ieme nœud de politesse greet en une insulte insult:

```
<?xml version="1.0" encoding="utf-8"?>
<doc>
   <greet kind="informal">Hi </greet>
   <insult kind="impolite"> @!!@*# </insult>
   <greet kind="casual">Hello </greet>
   <location kind="global">World</location>
   <location kind="local">Amsterdam</location>
   <last/>
</doc>
```

```
rename node
  doc("greetings.xml")//greet[2]
as
  "insult"
```

L'expression de XQuery

XQuery Expr :=Constants | Variable | FunctionCalls |
PathExpr | ComparisonExpr | ArithmeticExpr |
LogicExpr | FLWRExpr | ConditionalExpr |
QuantifiedExpr | TypeSwitchExpr |
InstanceofExpr | CastExpr | UnionExpr |
IntersectExceptExpr | ConstructorExpr |
ValidateExpr | InsertExpr| DeleteExpr
| RenameExpr | ReplaceExpr | TransformExpr

Syntaxe de Transform

What to update

```
TransformExpr::=
    copy $VarName := ExprSingle
         (,$VarName := ExprSingle)*
    modify ExprSingle
    return ExprSingle
```

Update expr.

Expr. returned

Transform retourne une copie modifiée, sans impact sur la base d'origine (c'est une non updating expression).

Exemple 1

```
let $oldx := <a>2<b><x>2</x></b></a>
return
  copy $newx := $oldx
  modify (
        rename node $newx as "nouveaux",
        replace value of node $newx//b
        with $newx * 2)
  return ($oldx, $newx)
```

```
<a>
  <b>2
    <x>2</x>
  </b>
</a>
<nouveaux>2
  <b>44</b>
</nouveaux>
```

Exemple 2

```
copy $target := <CONT id="s1">some text</CONT>
modify (
  rename node $target as "SECTION",
  insert node <TITLE>The title</TITLE>
    as first into $target
)
return element DOC { $target }
```

```
<DOC>
  <SECTION id="s1">
      <TITLE>The title</TITLE>
       some text
  </SECTION>
</DOC>
```

Plan

- Introduction
- Extension de Xquery
- XQuery Update Processing Model
- Le mixe entre les expressions Updating et non updating
- Exercices

XQuery Update Processing Model

- Les mises à jour ne sont pas appliquées immédiatement, ils sont accumulés dans une "liste de mise à jour en attente" Pul ("Pending Update List") = ensemble de primitives de mise à jour.
  - Par exemple, pour un insert, la Pul est obtenue ainsi:
    - ✓ évaluer la cible mise à jour (qui sont les nœuds qui devrait obtenir de nouveaux enfants?)
    - ✓ pour chacun de ces nœuds, ajouter à la Pul, le couple nœud et son ajout.
- À la fin de l'étape d'exécution, les primitives de mise à jour sont vérifiées à la recherche de conflits, et si aucun conflit n' apparaît, ils sont tous appliqués à la fois.
- La base de données est mise à jour de manière atomique.

Conséquences du "Pending Update"

- L'ordre dans lequel les mises à jour sont spécifiées n'est pas important. Le delete a la priorité sur d'autres opérations.

- Les primitives de mise à jour pourraient produire des changements conflictuels et des résultats imprévisibles.
  - Par exemple, deux rename du même nœud sont conflictuels, parce que nous ne savons pas dans quel ordre ils seraient appliqués.
  - Autres opérations ambiguës: deux replace d'un même nœud ou deux replace de contenu du même nœud.

Exercice

```
for $idattr in doc("data.xml")//ITEM/@Id   (: selection :)
return (                (: updates :)
   delete node $idattr,
   insert node <NID>{string($idattr)}</NID>
   as first into $idattr/..
)
```

????is a correct expression????

Solution: correct

```
for $idattr in doc("data.xml")//ITEM/@Id   (: selection :)
return (                (: updates :)
   delete node $idattr,
   insert node <NID>{string($idattr)}</NID>
   as first into $idattr/..
)
```

the order in which updates are specified is not important.

In this example you can delete the attribute Id (pointed by $idattr), and *after* use $idattr/.. (the parent ITEM element) for inserting! Or you could insert first and delete after.

Plan

- Introduction
- Extension de Xquery
- XQuery Update Processing Model
- Le mixe entre les expressions Updating et non updating
- Exercices

Le mixe entre les expressions Updating et non updating

- Le mélange entre des Updating and Non-updating Expressions **est interdit** dans une séquence (l'opérateur virgule).

Update conditionnel

- Les branches d'un **if** ou d'un **typeswitch** doivent être cohérents: **soit les deux de type Updating ou les deux de type Non-updating**.

- Si les deux branches dont de type updating → le If est dit de type Updating

Les fonctions de type Updating

- Si le corps de la fonction est de type Updating alors la fonction doit être déclarée avec le mot clé updating :

```
declare updating function insert-id($elem, $id-value)
    { insert node attribute id { $id-value } into $elem }
```

- Un appel à cette fonction est considéré comme une Updating Expression.

Plan

- Introduction
- Extension de Xquery
- XQuery Update Processing Model
- Le mixe entre les expressions Updating et non updating
- Exercices

Exercice

Document ("Employee.xml")

```
<employees>
  <employee>
        <name>Smith</name>
        <skill>Java</skill>
        <salary>100000</salary>
  </employee>
  <employee>
        <name>Jones</name>
        <skill>C++</skill>
        <salary>60000</salary>
  </employee>
  <employee>
        <name>Roberts</name>
        <skill>Java</skill>
        <salary>150000</salary>
  </employee>
</employees>
```

Retourner une séquence avec tous les éléments employee ayant Java comme skill sans l'information concernant leur salary.

Result

```
<employee>
    <name>Smith</name>
    <skill>Java</skill>
</employee>
<employee>
    <name>Roberts</name>
    <skill>Java</skill>
</employee>
```

Exercice: solution avec updating

Retourner une séquence avec tous les éléments employee ayant Java comme skill sans l'information concernant leur salary.

```
for $e in doc("Employee.xml")//employee[skill = "Java"]
return
  copy $je := $e
  modify delete node $je/salary
  return $je
```

Exercice: solution Xquery pur

```
for $e in doc("Employee.xml")//employee[skill = "Java"]
return
   element{name($e)}
            {for $el in $e/*[name(.) != "salary"]
            return $el}
```

```
for $e in doc("Employee.xml")//employee[skill = "Java"]
return
  copy $je := $e
  modify delete node $je/salary
  return $je
```

Implementation XQuery Update

- BaseX
- eXist
- Saxon
- Oracle Berkeley DB XML (Oracle),

Exercise

Assuming that element <BIDS> could or or not be present

```
<PERSON id="p0234">
  <NAME>Joe</NAME>
</PERSON>
```

```
<PERSON id="p0234">
  <NAME>Joe</NAME>
 <BIDS/>
</PERSON>
```

How to do this with an updating function:

declare updating function insert-bid ($person, $bid)

```
<PERSON id="p0234">
  <NAME>Joe</NAME>
  <BIDS>
    <BID id="b0012">data</BID>
  <BIDS>
</PERSON>
```

Solution: incorrect!

```
<PERSON id="p0234">
  <NAME>Joe</NAME>
</PERSON>
```

```
declare updating function insert-bid ($person,
                                      $bid)
{
  if(empty($person/BIDS))
  then insert node <BIDS/> into $person
  else (),
  insert node $bid as last into $person/BIDS
}
```

Why? Because the BIDS element will be created only at the very end, therefore the instruction
insert … as last into $person/BIDS
will not find any node matching $person/BIDS → an execution error.

Solution: correct!

```
<PERSON id="p0234">
  <NAME>Joe</NAME>
</PERSON>
```

```
declare updating function insert-bid ($person, $bid)
{
  if(empty($person/BIDS))
  then insert node <BIDS>{$bid}</BIDS> into $person
  else  insert node $bid as last into $person/BIDS
}
```

```
<PERSON id="p0234">
  <NAME>Joe</NAME>
  <BIDS>
    <BID id="b0012">data</BID>
  <BIDS>
</PERSON>
```

Exercise: Rename expression

The effects of a rename expression are limited to its target node. Attributes and descendants of the target node are not affected.

Rename all the attributes and descendants bound to variable `$root`*:* change all QNames with the prefix `abc` to have a new prefix `xyz` and a new namespace URI `http://xyz/ns`.

(Hint: some form of explicit iteration must be used)

Solution

```
for $node in $root//abc:*
let $localName := fn:local-name($node),
    $newQName := fn:concat("xyz:", $localName)
return (
   rename node $node
     as fn:QName("http://xyz/ns", $newQName),
     for $attr in $node/@abc:*
     let $attrLocalName := fn:local-name($attr),
        $attrNewQName := fn:concat("xyz:", $attrLocalName)
     return rename node $attr
                    as fn:QName("http://xyz/ns", $attrNewQName) )
```

Transform expression: Exercise

Return all managers, omitting their salaries and replacing them with an attribut xsi:nil

Document

```
<employees>
  <employee mgr="true" dept="Toys">
    <name>Smith</name>
    <salary>100000</salary>
  </employee>
  <employee dept="Toys">
    <name>Jones</name>
    <salary>60000</salary>
  </employee>
  <employee mgr="true" dept="Shoes">
    <name>Roberts</name>
    <salary>150000</salary>
  </employee>
</employees>
```

Desired result

```
<employee mgr="true" dept="Toys">
  <name>Smith</name>
  <salary xsi:nil="true"/>
</employee>
<employee mgr="true" dept="Shoes">
  <name>Roberts</name>
  <salary xsi:nil="true"/>
</employee>
```

Solution

Return all managers, omiting their salaries and replacing them with an attribute xsi:nil

```
for $e in document("employees.xml")//employee
where $e/@manager = true()
return
  copy $emp := $e
  modify
   (replace value of node $emp/salary with "" ,
    insert nodes (attribute xsi:nil{"true"})
    into $emp/salary)
   return $em
```

Exercise

- An address book is synchronized among a central archive and two local copies
  - Entries with the same name element are considered to be the same entry
  - It is assumed that entries may be modified, but not inserted or deleted

Exercise

Address book synchronization rules: between archive a and two copies ci, cj

- **Case1:** ci = a and cj ≠a ⇒

  propagate cj to a and ci

- **Case2:** ci ≠ a, cj ≠ a ⇒

  a. if possible, merge differences and propagate them to a, then to ci, cj

  b. Otherwise, report the problem into the "log.xml" file

```
<fail >
  <arch >{ $a }</ arch ><v1 >{ $v1 }</v1 > <v2 >{ $v2 }</v2 >
</fail >
```

Archive.xml/log.xml

archive.xml: The central archive

```
<archived-agenda>
<last-synch-time>2005-10-05T10:00</last-synch-time>
<entry>
<name>Benjamin</name>
<contact>benjamin@inria.fr</contact>
</entry>
<entry>
<name>Dario</name>
<contact>dario@uni-pisa.it</contact>
</entry>
<entry>
<name>Anthony</name>
<contact>tony@uni-toulon.fr</contact>
</entry>
</archived-agenda>
```

log.xml: The central log, before synchronization

```
<log>
</log>
```

copy1.xml

copy1.xml: The first modified copy of the address book

```
<agenda-version>
<entry>
<name>Benjamin</name>
<contact>benjamin@uni-versailles.fr</contact>
</entry>
<entry>
<name>Dario</name>
<contact>dario@uni-parissud.fr</contact>
</entry>
<entry>
<name>Anthony</name>
<contact>tony@ena.fr</contact>
</entry>
</agenda-version>
```

Diffèrent de archive.xml:
<contact>dario@uni-pisa.it</contact>

Diffèrent de archive.xml:
<contact>tony@uni-toulon.fr</contact>

copy2.xml

copy2.xml: The second modified copy of the address book

```
<agenda-version>
<entry>
<name>Benjamin</name>
<contact>benjamin@uni-versailles.fr</contact>
</entry>
<entry>
<name>Dario</name>
<contact>dario@uni-pisa.it</contact>
</entry>
<entry>
<name>Anthony</name>
<contact>tony@ehess.fr</contact>
</entry>
</agenda-version>
```

Case1

Idem que archive.xml:
<contact>dario@uni-pisa.it</contact>
Different de copie1.xml
<contact>dario@uni-parissud.fr</contact>

Case2.b

Diffèrent de archive.xml:
<contact>tony@uni-toulon.fr</contact>
Différent de copie1.xml
<contact>tony@ena.fr</contact>

Solution

```
for $a in doc (" archive.xml")//entry ,
    $v1 in doc (" copy1.xml")/version/entry ,
    $v2 in doc (" copy2.xml")/version/entry
where $a/ name = $v1/name and $v1/name = $v2/name
return
    if ($a/contact = $v1/contact and $v1/contact =$v2/contact  then ()   Case0
     else if ($v1/contact = $v2/contact )  Case2.a
          then replace value of node $a/contact with $v1/contact
           else if ($a/contact = $v1/contact )
      Case1  then ( replace value of node $a/contact  with $v2/contact ,
                    replace value of node $v1/contact with $v2/contact …
      Case1   else if ($a/contact = $v2/contact )
                  then ( replace value of node $a/contact with $v1/contact ,
                         replace value of node $v2/contact with $v1/contact )
                  else ( insert node
                   Case2.b  <fail><arch>{ $a }</arch><v1 >{ $v1 }</v1>
                                <v2 >{ $v2 }</v2></fail>
                        into document("log.xml")/log ),
     replace value of
        node document("archive.xml")/*/lastSynchTime
     with current-dateTime ())
```

log.xml after synchronisation

```
<log>
    <fail>
      <arch>
          <entry>
              <name>Anthony</name><contact>tony@uni-toulon.fr</contact>
          </entry>
      </arch>
      <v1>
         <entry>
             <name>Anthony</name><contact>tony@ena.fr</contact>
         </entry>
      </v1>
      <v2>
        <entry>
          <name>Anthony</name><contact>tony@ehess.fr</contact>
        </entry>
      </v2>
    </fail>
</log>
```

Exercise

1. Delete all parts in part-tree.xml
2. Delete all parts belonging to a car in part-tree.xml, leaving the car itself

Part-tree.xml

```
<parttree>
  <part partid="0" name="car">
      <part partid="1" name="engine">
          <part partid="3" name="piston"/>
      </part>
      <part partid="2" name="door">
          <part partid="4" name="window"/>
          <part partid="5" name="lock"/>
       </part>
  </part>
  <part partid="10" name="skateboard">
       <part partid="11" name="board"/>
       <part partid="12" name="wheel"/>
   </part>
   <part partid="20" name="canoe"/>
</parttree>
```

Solution

1. Delete all parts in part-tree.xml

```
delete nodes doc("part-tree.xml")//part
```

2. Delete all parts belonging to a car in part-tree.xml, leaving the car itself

```
delete nodes doc("part-tree.xml")//part[@name="car"]//part
```

Exercise – Cont.

1. Delete all **parts** in **partlist.xml**
2. Delete all **parts** belonging to a **car** in **partlist.xml**, leaving the **car** itself

partlist.xml: Flat representation

```
<partlist>
    <part partid="0" name="car"/>
    <part partid="1" partof="0" name="engine"/>
    <part partid="2" partof="0" name="door"/>
    <part partid="3" partof="1" name="piston"/>
    <part partid="4" partof="2" name="window"/>
    <part partid="5" partof="2" name="lock"/>
    <part partid="10" name="skateboard"/>
    <part partid="11" partof="10" name="board"/>
    <part partid="12" partof="10" name="wheel"/>
    <part partid="20" name="canoe"/>
</partlist>
```

Solution

1. Delete all parts in partlist.xml

```
delete nodes doc("partlist.xml")//part
```

2. Delete all parts belonging to a car in partlist.xml, leaving the car itself

```
for $t in doc("part-
 tree.xml")//part[@name="car"]//part,
    $l in doc("part-list.xml")//part
where $t/@partid eq $l/@partid
return
delete nodes $l
```

Solution

```
declare updating function local:delete-subtree($p as
  element(part))
{
    for $child in doc("part-list.xml")//part
    where $p/@partid eq $child/@partof
    return (
            delete nodes $child,
            local:delete-subtree($child)
    )
};
for $p in doc("part-list.xml")//part[@name="car"]
return   local:delete-subtree($p)
```

Exercise – Cont.

- Add a radio to the car in part-tree.xml, using a part number that hasn't been taken.

Solution

```
let $next := max(doc("part-tree.xml")//@partid) + 1
return
    insert nodes <part partid="{$next}" name="radio"/>
    into doc("part-tree.xml")//part[@partid=0 and
                                    @name="car"]
```

Position of new element with respect to its siblings is implementation-dependent. If position is significant, the next query ensures that the element appears last:

```
return
    insert … as last into…
```

Exercise – Cont.

- The head office has adopted a new numbering scheme. In part-tree.xml:
  - add 1000 to all part numbers for cars,
  - 2000 to all part numbers for skateboards,
  - and 3000 to all part numbers for canoes

Solution

```
for $keyword at $i in ("car", "skateboard", "canoe"),
    $parent in doc("part-
 tree.xml")//part[@name=$keyword]
let $descendants := $parent//part
for $p in ($parent, $descendants)
            (: parent node + descendents :)
return
   replace value of node $p/@partid
   with $i*1000+$p/@partid
```

* Exercise

- Que donne:

```
let $var := <total><a1><x/></a1><x><xx/></x></total>
for $x in $var//x
return
        copy $je := $x
        modify delete node $je
        return $je
```

```
let $var := <total><a1><x/></a1><x><xx/></x></total>
return
        copy $je := $var
        modify delete node $je//x
        return $je
```

* Solution

```
let $var := <total><a1><x/></a1><x><xx/></x></total>
for $x in $var//x
return
        copy $je := $x
        modify delete node $je
        return $je
```

```
<total>
  <a1>
    <x/>
  </a1>
  <x>
    <xx/>
  </x>
</total>
```

```
<x/>
<x>
  <xx/>
</x>
```

Let $tlist be the list of nodes returned by the target expression.

If any node in $tlist has no parent it is removed from $tlist (and is thus ignored in the following step).

For each node $tnode in $tlist, the update primitive **upd:delete($tnode)** is appended to the pending update list.

* Solution

```
let $var := <total><a1><x/></a1><x><xx/></x></total>
return
        copy $je := $var
        modify delete node $je//x
        return $je
```

```
<total>
  <a1>
    <x/>
  </a1>
  <x>
    <xx/>
  </x>
</total>
```

```
<total>
  <a1/>
</total>
```

Let $tlist be the list of nodes returned by the target expression.
If any node in $tlist has no parent it is removed from $tlist (and is thus ignored in the following step).
For each node $tnode in $tlist, the update primitive **upd:delete($tnode)** is appended to the pending update list.