Updating XML with XQuery

Web Data Management and Distribution

Serge Abiteboul   Ioana Manolescu   Philippe Rigaux
Marie-Christine Rousset   Pierre Senellart

Web Data Management and Distribution
http://webdam.inria.fr/textbook

September 29, 2011

Why XQuery Update

XQuery is a *read-only* language: it can *return (compute)* an instance of the XQuery Data Model, but it cannot *modify* an existing instance.

SQL parallel:

                select... from... where...

without

          insert into table... update table...

Applications require reading *and* updating XML data.

XQuery Update Facility: a working draft, not yet a specification

                http://www.w3.org/TR/xquery-update-10/

Requirements for the XQuery Update language

Expressive power:

- Insert
- Delete
- Update
- Copy with new identity

Extension of XQuery itself:

- Simplifies understanding and learning the language
- Difficulty to introduce side effects...

Well-defined semantics

Conciseness

Amenable to efficient implementation...

XQuery Update concepts

All XQuery expressions can be classified into:

- Updating expressions
- Non-updating expressions

XQuery Update introduces five new kind of expressions:

- insert, delete, replace, rename: updating expressions
- transform: non-updating expression

XQuery Update specifies:

- how all XQuery expressions are classified into updating and non-updating
- places where each type of expression can appear
- syntax and semantics of each new expression

XQuery Update processing model

The evaluation of an expression produces:

- an instance of the XQuery Data Model and
- a pending update list: set of update primitives, i.e. node stat changes that have to be applied.

In the current specification, one of the two has to be empty. This may change in the future.

(The evaluation of a simple XQuery produces an instance of the XQuery Data Model.)

Each update primitive has a target node.

Update primitives are checked for conflicts, and if no conflict appears, they are applied.

Insert expressions

Insert is an updating expression.

General form:

insert (constructor|({expr})) (as (first|last))? into
                              (after|before) expr

The first expression is called the *source*, and the second the *target*.

The source and target expressions must not be updating.

```
insert <year>2005</year>
after doc("bib.xml")/books/book[1]/published
```

```
insert $article/author
as last into doc("bib.xml")/books/book[3]
```

Insert expressions

The pending update list is obtained as follows:

- evaluate the update target (which are the nodes that should get new children)
- for each such node, add to the pul the corresponding add-child operation

```
insert {$new-police-report} as last
into doc("insurance.xml")//policies/policy[id=$pid]
/driver[licence=$licence]/accident[date=$dateacc]
/police-reports
```

- locate the proper police-reports element
- for each element in $new-police-report, add an add-last-child operation to the pul

Delete expressions

Delete is an updating expressions. Its produces a non-empty pending update list.

General form:

delete expr

```
delete doc("bib.xml")/books/book[1]/author[last()]

delete /email/message[fn:currentDate()-date >
xdt:dayTimeDuration(P365D)]
```

Replace expressions

Replace is an updating expression. It produces a non-empty pending update list.

General form:

replace expr with expression

```
replace doc("bib.xml")/books/book[1]/publisher
with doc("bib.xml")/books/book[2]/publisher

replace value of doc("bib.xml")/books/book[1]/price
with doc("bib.xml")/books/book[1]/price*1.1
```

Rename expression

Rename is an updating expression.

General form:

rename expr to expr

```
rename doc("bib.xml")/books/book[1]/author[1]
to main-author

rename doc("bib.xml")/books/book[1]/author[1]
to $newname
```

Transform expressions (1)

Transform is a non-updating expression.

General form:

copy $varName := expr (, $varName := expr)*
modify expr return expr

Example: return all managers, omiting their salaries and replacing them with an attribute xsi:nil.

Remark

It can be done with XQuery. But it's painful!

Transform returns a *modified copy*, without impacting the original database (it is a non-updating expression).

Transform expressions (2)

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>
```

It can be done with XQuery. But it is difficult! Exercise...

Transform expressions (3)

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

```
for $e in doc("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
```

Programming with XQuery Update

Address book synchronization:

- One archive version and two copies
- $c_1 = a$ and $c_2 \neq a \Rightarrow$ propagate c_2 to a and c_1
- $c_1 \neq a$, $c_2 \neq a \Rightarrow$
    - If possible, merge differences and propagate them to a, then to c_1, c_2
    - Otherwise, raise an error.

Agenda entries are of the form:

```
<entry>
    <name>Benjamin</name>
    <contact>benjamin@inria.fr</contact>
</entry>
<entry>
    <name>Anthony</name>
    <contact>tony@uni-toulon.fr</contact>
</entry>
```

Programming with XQuery Update

```
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 ()
  else
    if ($v1/contact=$v2/contact) then
      replace value of node $a/contact with $v1/contact
    else
      if ($a/contact=$v1/contact)
      then (
            replace value of $a/contact
                   with $v2/contact,
            replace value of $v1/contact
                   with $v2/contact ...
```

Programming with XQueryUpdate

```
...
   if ($a/contact = $v1/contact)
   then ...
   else
     if ($a/contact = $v2/contact)
     then ( replace value of $a/contact
                   with $v1/contact,
            replace value of $v2/contact
                   with $v1/contact )
     else ( insert node
              <fail> <arch>{$a}</arch>
                     <v1>{$v1}</v1> <v2>{$v2}</v2>
              </fail>
            into doc("log.xml")/log ),
replace value of node doc("archive.xml")
       /*/last-synch-time with current-dateTime()
```

XQuery - SQL comparison

Function	Relational	XML
Query (read-only)	SQL *select*	XQuery
Update	SQL *update*	XQuery Update
Full-text	SQL MMS	XQuery Full-Text
Scripting	PL/SQL	XQuery Scripting Extension

XQuery update is not a programming language. Missing:

- Control over the *scope of snapshots*, i.e. when do my updates become visible to another query? XQuery Update: after the current query has finished executing.

- Control over atomicity, i.e. which expressions must be executed atomically?

- The possibility to both return a result and have side effects. XQuery Update: one or the other is empty.

- Error handling.

An XQuery scripting language: XQuery-P

D.Chamberlin, M.Carey, D.Florescu, D.Kossman: "Programming with XQuery", XIME-P 2006

1. Define a sequential execution mode: the statements must be evaluated in order, and each statement sees the side effect of the previous one

2. Define blocks, which are units of code to be executed sequentially. New variables can be defined inside a block. The returned result is that of the last expression.

3. Introduce assignments to bind variables to new values.

```
for $item in /catalog/item[price < 100]
return
{replace value of $item/price with $item/price * 1.1;
 $item}
```

An XQuery scripting language: XQuery-P

Forces to define evaluation order on an XQuery expression:

- for, let, where, order by executed in the order of their appearance; then, return
- if evaluated first, then evaluate then or else
- ,: evaluate from left to right, apply all the updates after each item
- function call: evaluate the arguments before the body

Specifying evaluation order is a big departure from traditional query language style. (Which of `select`, `from` and `where` is evaluated first?)

Programming with XQuery-P

```
declare updating function local:transfer
 ($from-acctno as xs:string, $to-acctno as xs:string,
  $amount as xs:decimal) as xs:integer
 { declare $from-acct as element(account)
     := /bank/account[acctno eq $from-acctno],
          $to-acct as element(account)
     := /bank/account[acctno eq $to-acctno];
   if ($from-acct/balance > $amount)
   then atomic {
     do replace value of $from-acct/balance
        with $from-acct/balance - $amount;
     do replace value of $to-acct/balance
        with $to-acct/balance + $amount;
     0 } (: end of atomic region :)
   else -1
};
```

Implementations

XQuery Update:

- eXist
- MonetDB

XQuery-P and similar proposals: preliminary prototypes