Storing and Queryng XML Data, XQuery

Tomáš Pitner, Luděk Bártek, Adam Rambousek, Lukáš Grolig

XQuery

- is a **query language**
  - for **searching** and **extraction** of XML nodes (elements, attributes) from a document
  - for an output XML document **construction**.
- Created by a different W3C group (community) than XSLT
- Purpose might be the same but XQuery tends to be used for
- more structured data while XSLT rather for documents
- (which are more narrative, less structured).
- **SQL-like**

Characteristics

- The XQuery is the ***most common XML query language*** at present time (and it seems to be in future as well).
- Based on ***XPath 2.0 data model*** (XQuery 1.0) or common XML Schema and XPath 3.0 data model, operators and functions in case of XQuery 3.0.
- Supported by main database engines producers (IBM, MS, Oracle, etc)

What is XQuery?

- Combines XPath 2.0,
- FLWOR construct(s) - main backbone of the language,
- Element constructors,
- User-defined functions,
- other directives.

XQuery features

- All XQuery expressions operate on ***sequences***, and evaluate to sequences, i.e.
  - list of ***nodes***
  - list of ***atomic values.***
- - Extending XPath, adding complex queries.
- - Extension for ***updates***.

XQuery Specification

- Developed by W3C Consortium
  - XML Query Group in collaboration with the XSL Working Group
- XQuery 1.0 became a W3C Recommendation on 14 December 2010
- XQuery 3.0 became a W3C Recommendation on April 8, 2014
- The XQuery timeline

Processing of Queries

- Native XML databases
  - BaseX
  - eXist
- Used for "real" querying within collections of XML document.
- XML-enabled databases.
- Some XSLT/XQuery processors, such as Saxon
  - usually for querying just one document.

Should we use DOM, XSLT, or XQuery?

- Tasks where **extraction (selection) part is more complicated** than the construction part
  - use **XQuery**
- In other cases, i.e. more **complex output** is required
  - if more "narrative", less structured input - use **XSLT**
  - if more **complex operations** are required - use a more **general API** such as **DOM**

Source code example

```
<?xml version="1.0" encoding="UTF-8"?>
<addressbook>
 <person category="friends">
      <firstname>Petr</firstname>
      <lastname>Novak</lastname>
      <date-of-birth>1969-05-14</date-of-birth>
      <email>novak@myfriends.com</email>
      <characteristics lang="en">Very good friend</characteristics>
    </person>
   <person category="friends">
      <firstname>Jaroslav</firstname>
      <lastname>Nováček</lastname>
      <date-of-birth>1968-06-14</date-of-birth>
      <email>novacek@myfriends.com</email>
      <characteristics lang="en">Another good friend</characteristics>
    </person>
```

Source code example

```
<person category="staff">
      <firstname>Jan</firstname>
      <lastname>Horak</lastname>
      <date-of-birth>1970-02-0</date-of-birth>
      <email>horak@mycompany.com</email>
      <characteristics lang="en">Just colleague</characteristics>
    </person>
    <person category="friends">
      <firstname>Erich</firstname>
      <lastname>Polak</lastname>
      <date-of-birth>1980-02-28</date-of-birth>
      <email>erich@myfriends.com</email>
      <characteristics lang="en">Good friend</characteristics>
    </person>
  </addressbook>
```

Example - Simple Query (XPath)

- Task: "extract all surnames in the addressbook".
- Query is more-or-less just an XPath expression, like ***"selects all lastname elements"***:

```
doc('myaddresses.xml')/addressbook/person/lastname
```

Result

The query to above mentioned document will output:

<lastname>Novák</lastname> <lastname>Nováček</lastname>
<lastname>Horák</lastname> <lastname>Polák</lastname>

Complex XQuery (FLWOR)

- ***FLWOR*** is an acronym of an XQuery structure. It roughly corresponds to the SQL query structure:
  - **(F)or** - Initial query part that specifies query cycle including control variable. Results of XPath expression behind the keyword " in" are assigned to the variable.
  - **(L)et** - You can assign values of next variable that can be used later in this section.
  - **(W)here** - specifies selection condition ie. which nodes (values) selected by for section will be used.The condition can utilize the variables defined in the "let" section.
  - **(O)rder** - Defines how the nodes should be ordered.
  - **Return** - Defines what is returned, constructed from extracted nodes (values).

FLWOR -- example

- Condition used to select requested nodes can be specified either in an XPath expression in `for` clause or in the `where` clause.
- "Return Mr. Polak's birth-date." :

```
for $person in doc('myaddresses.xml')/addressbook/person
where $person/lastname='Polák'
return $person/date-of-birth
```

- XQuery returns ->

```
<?xml version=" 1.0" encodings"UTF-8"?>
<date-of-birth>l980-02-28</date-of-birth>
```

Output formatting

- Return clause may contain XML tags, creating new output XML document
- Enclosed expression to evaluate in **{ }**

```
for $person in doc('myaddresses.xml')/addressbook/person
where $person/lastname='Polák'
return <datum>{$person/date-of-birth/text()}</datum>
```

- XQuery returns:

```
<datum>1980-02-28</datum>
```

Nested queries

- Queries may be nested in each other, use ***{ }***:

```
for $person in doc('myaddresses.xml')/addressbook/person
where $person/lastname='Polák'
return
  <osoba>
    <chars>
      {for $char in $person/characteristics return <char>{$char/text()}</char>}
    </chars>
  </osoba>
```

- XQuery returns:

```
<osoba>
 <chars>
  <char>Good friend</char>
 </chars>
</osoba>
```

XQuery 3

- group by
- less strict order of elements
- function items (lambda functions)
- ***let $f := function($x, $y) { $x + $y } return $f(17, 25)***
- ***try/catch***
- ***switch***
- ...

XQuery Update

- XQuery Update Facility 1.0, Recommendation 17 March 2011
- Extension to provide ***update features***
- Insert nodes
- Delete nodes
- Replace value/node

XQuery Update example

```
update insert <email>new@mail.com</email> into
for $person in doc('myaddresses.xml')/addressbook/person
where $person/lastname='Polák'

update delete doc('myaddresses.xml')/addressbook/person/lastname
```

XQuery (XPath) frequently used numeric functions

- ***count($seq as item()*)***
  - Counts the items in a sequence.
- ***sum($seq as item()*)***
  - Returns the sum of the items in a sequence.
- ***avg($seq as item()*)***
  - Returns the average of the items in a sequence.
- ***min($seq as item()*)***
  - Returns the minimum valued item in a sequence.
- ***max($seq as item()*)***
  - Returns the maximum valued item in a sequence.

XQuery (XPath) frequently used element function

- ***distinct-values($seq as item()*)***
  - Returns select distinct items from a sequence.
- ***subsequence($seq as item()*, $startingLoc as xs:double, $length as xs:double)***
  - Returns a subset of provided sequence.
- ***insert-before($seq as item()*, $position as xs:integer, $inserts as item()*)***
  - Inserts an item in a sequence.
- ***remove($seq as item()*, $position as xs:integer)***
  - Removes an item from a sequence.

XQuery (XPath) frequently used element functions

- ***reverse($seq as item()*)***
  - Returns the reversed sequence.
- ***index-of($seq as anyAtomicType()*, $target as anyAtomicType())***
  - Returns indexes as integers to indicate availability of an item within a sequence.
- ***last()***
  - Returns the last element of a sequence when used in predicate expression.
- ***position()***
  - Used in FLOWR expressions to get the position of an item in a sequence.

XQuery (XPath) frequently used string functions

- ***string-length($string as xs:string) as xs:integer***
  - Returns the length of the string.
- ***concat($input as xs:anyAtomicType?) as xs:string***
  - Returns the concatenated string as output.
- ***string-join($sequence as xs:string*, $delimiter as xs:string) as xs:string***
  - Returns the combination of items in a sequence separated by a delimiter.

Regular expressions in XQuery

- ***matches($input, $regex)***
  - Returns true if the input matches with the provided regular expression.
- ***replace($input, $regex, $string)***
  - Replaces the matched input string with given string.
- ***tokenize($input, $regex)***
  - Returns a sequence of items matching the regular expression.

User defined functions

- allow to create a named function (possibly with parameters) to be used in query script
- can be assigned to a "local" or other namespace

Example - *function*

```
declare namespace company='http://www.mycompany.com';
declare function company:first_person() as node() {
    <person id="1">
        <firstname>Tomáš</firstname>
        <surname>Novák</surname>
    </person>
};
```

Quantifiers

- Enable to construct conditions testing whether some condition is satisfied "for all" or "for some" items
  - ***some***
  - ***all***

Example

- returns those element from the first list that are also present in the second list
- syntax: some ***item*** in ***list*** satisfies ***condition:***

```
declare function local:in_both_lists($list1 as node()*, $list2 as
node()*) as node()* {
  for $item1 in $list1/item
  let $item-text := $item1/text()
  return
    if (some $item2 in $list2/item satisfies $item2/text() =
$item1/text())
       then $item1
       else ()
};
```

Example of *if/then/else*

```
<result>
{
   if(not(doc("books.xml"))) then (
      <error>
         <message>books.xml does not exist</message>
      </error>
   )
   else (
      for $x in doc("books.xml")/books/book
      where $x/price>30
      return $x/title
   )
}
</result>
```

Resource on XQuery

- http://en.wikibooks.org/wiki/XQuery