SDXML VT2026 Models and languages for semi-structured data and XML

Query languages for XML XQuery

nikos dimitrakas
nikos@dsv.su.se
08-161295

Corresponding reading
Chapter 10, 11, 13.3, A.3.1, C.2 of the course book
Parts of chapter 30 of Database Systems (Connolly, Begg) 6th edition (chapter 31 in 5th edition)
Articles about XML Query Languages
Compendium about XQuery

XQuery 1.0

- **Standard**
- **Model**
- **Query language**
  - based on (inspired by) SQL, XQL, XML-QL, Lorel, YATL, etc.
  - declarative (not procedural)
  - includes XPath 2.0
  - XQueryX - XQuery in XML syntax
  - FLWOR (for let where order by return)
    - » Corresponds to SQL SELECT
  - transform statements for the rest of the DML statements (from 2011)
    - » separate specification
- **Next version XQuery 3.0**
  - together with XPath 3.0 and XSLT 3.0

Sample data

```
<Movies>
    <Movie Title="Driven" Year="2001">
        <Actor Name="Burt Reynolds" YearOfBirth="1936" Country="USA"/>
        <Actor Name="Silvester Stallone" YearOfBirth="1946" Country="USA"/>
        <Actor Name="Kip Pardue" YearOfBirth="1976" Country="Canada"/>
        <Director Name="Silvester Stallone" YearOfBirth="1946" Country="USA"/>
        <ProductionCompany>Tri-Star</ProductionCompany>
    </Movie>
    <Movie Title="Antz" Year="1998">
        <Actor Name="Woody Allen" YearOfBirth="1935" Country="USA"/>
        <Actor Name="Silvester Stallone" YearOfBirth="1946" Country="USA"/>
        <Actor Name="Sharon Stone" YearOfBirth="1958" Country="USA"/>
        <Director Name="Eric Darnell" YearOfBirth="1961" Country="Ireland"/>
        <ProductionCompany>Universal</ProductionCompany>
    </Movie>
    <Movie Title="Picking Up the Pieces" Year="2000">
        <Actor Name="Woody Allen" YearOfBirth="1935" Country="USA"/>
        <Actor Name="Sharon Stone" YearOfBirth="1958" Country="USA"/>
        <Actor Name="Alfonso Arau" YearOfBirth="1948" Country="USA"/>
        <Director Name="Eric Darnell" YearOfBirth="1961" Country="Ireland"/>
        <ProductionCompany>Tri-Star</ProductionCompany>
    </Movie>
    …
</Movies>
```

XQuery - FLWOR

- **for**
  - **Loops through sequences (nodes or values)**
- **let**
  - **Assignments**
- **where**
  - **Conditions**
- **order by**
  - **Sorting (affects the result)**
  - **ascending (default) or descending**
- **return**
  - **Construction of the result**

XQuery

- FLWOR statements may be nested.
- No clause is compulsory.
- for and let may appear multiple times in random order (before the where clause).
- XPath expressions may be used in all clauses.
- The result can be well-formed XML, but it does not have to.
  - the result can be anything that is supported by the XQuery model, meaning a sequence of nodes and/or values
- The function doc() can be used to define a source/context (an XML document). Otherwise, the execution environment can be used to configure the source/context.

XQuery

- Variables start with $:
  - for $a in //Movie/Actor
  - let $n := $a/@Name
- Sequences:
  - for $x in (1, 2, 3)
    - » same as for $x in 1 to 3
  - let $y := (1, 2, 3)
- Evaluation of expressions:
  - Expression to be evaluated inside { }:
  - <result>{$x*3}</result>

XQuery - Computed Constructors

- **element**
  - element name value:

  let $a := "a", $b := 2

  return <x>{element {$a} {$b}}</x>

    » gives <x><a>2</a></x>
- **attribute**
  - attribute name value:

  let $a := "a", $b := 2

  return <x>{attribute {$a} {$b}}</x>

    » gives <x a="2"/>
- **comment**
  - comment value
  - comment {"Ahoy"}
    » gives <!--Ahoy-->

XQuery - Computed Constructors

- **processing-instruction**
  - processing-instruction name value
  - processing-instruction Say {"Hello"}
    » gives <?Say Hello?>
- **text**
  - text value
  - text {"Howdy"}
    » creates a text node
    » same result as "Howdy"
- **document**
  - document content
  - document {comment {"The best movies"}, element Movies {…}}

XQuery - Computed Constructors

- Avoid unnecessary creations
- What do the alternative return clauses do?

```
let $m := //Movie[1]
return <Movie Title="{$m/@Title}"/>
return <Movie>{$m/@Title}</Movie>
return <Movie>{attribute Title {$m/@Title}}</Movie>
return element Movie {$m/@Title}
return element Movie {attribute Title {$m/@Title}}
```

XQuery - Conditional expressions

- if-then-else

```
for $a in (1 to 5)
return if ($a mod 2 = 0)
         then <even>{$a}</even>
         else <odd>{$a}</odd>
```

```
<odd>1</odd>
<even>2</even>
<odd>3</odd>
<even>4</even>
<odd>5</odd>
```

XQuery - Quantified expressions

- **some**

```
for $a in //Movie
where some $b in $a/Actor/@Country satisfies string($b) = "Austria"
return  $a
```

- **every**

```
for $a in //Movie
where every $b in $a/Actor/@Country satisfies string($b) = "USA"
return  $a
```

XQuery - Nested statements

- **The result of a statement becomes a source for another statement:**

```
for $x in distinct-values (for $a in (1 to 6), $b in (1 to 6)
                                    return <sum>{$a + $b}</sum>)
return <unique>{$x}</unique>
```

- **The result of a statement is assigned to a variable:**

```
for $x in (1 to 6)
let $content := for $a in (1 to 6)
                    return element Plus {attribute value {$a}, $x+$a}
return element Number {attribute value {$x}, $content}
```

XQuery - Namespaces

- Cannot be accessed through an axis
- Namespace aliases are not available
- To work with namespaces and aliases, they need to be declared in the XQuery prolog:
  - declare namespace alias = "URI";
  - declare default element namespace "URI"

```
declare namespace aaa = "URI-1";
declare default element namespace "URI-2";
element Result {for $x in (1 to 5)
                    return element aaa:Number {attribute value {$x}}}
```

```
<Result xmlns="URI-2">
  <aaa:Number xmlns:aaa="URI-1" value="1"/>
  <aaa:Number xmlns:aaa="URI-1" value="2"/>
  <aaa:Number xmlns:aaa="URI-1" value="3"/>
  <aaa:Number xmlns:aaa="URI-1" value="4"/>
  <aaa:Number xmlns:aaa="URI-1" value="5"/>
</Result>
```

XQuery prolog

- Declare
  - functions
  - variables
  - namespaces
  - collation
  - sorting
  - etc.

XQuery 3.0 and 3.1

- News
  - group by clause
  - the different clauses may appear in a more flexible order
  - switch expression
  - count clause
  - try/catch expression
  - dynamic and inline functions
  - computed constructor for namespace
  - many new functions, among others for mathematical operations
  - XQuery node test functions formally part of XPath
  - and much more

XQuery 3 - group by

- Groups the iterations so that
  - the return clause is executed once per group
  - the iteration variables become sequences containing all the values of the relevant iterations

```
for $x in 1 to 5
let $even := ($x mod 2) = 0
group by $even
return element Numbers {attribute even {$even}, $x}
```

Result:

```
<Numbers even="false">1 3 5</Numbers>
<Numbers even="true">2 4</Numbers>
```

XQuery 3 - group by

- can group on one or more variables
- the grouping variables can be declared earlier or in the group by clause
  - Note! If a variable is reused, its content is overwritten

```
for $x in (1,1,2,3,2,1,2,3,2)
group by $x
return element Number {
        attribute times {count($x)}, $x}

Result:

<Number times="1">1</Number>
<Number times="1">2</Number>
<Number times="1">3</Number>
```

```
for $x in (1,1,2,3,2,1,2,3,2)
group by $y := $x
return element Number {
        attribute times {count($x)}, $y}

Result:

<Number times="3">1</Number>
<Number times="4">2</Number>
<Number times="2">3</Number
```

XQuery 3 - Mixed clauses

```
for $x in (1,1,2,3,2,1,2,3,2,4,5,3)
where $x < 5
group by $v := $x
where count($x) > 1
let $s := sum($x)
return <Number value="{$v}" sum="{$s}" />

Result:

<Number value="1" sum="3"/>
<Number value="2" sum="8"/>
<Number value="3" sum="9"/>
```

Start with a let or a for and finish with a return. The rest is free!

XQuery 3 - switch

- Instead of nested if-then-else
  - many case
  - one default

```
for $x in 0 to 3
return element {switch ($x)
                    case 0 return "Zero"
                    case 1 return "One"
                    default return "Many"} {$x}
```

Result:

```
<Zero>0</Zero>
<One>1</One>
<Many>2</Many>
<Many>3</Many>
```

XQuery 3 - count (clause)

- Counts the iterations that reach it (the count clause)

```
for $x in 4 to 10
count $loop
let $nested := let $a := $x count $i return $i
where $x mod 2 = 0
count $correct
return <Number x="{$x}" loop="{$loop}" correct="{$correct}" nested="{$nested}"/>
```

Result:

```
<Number x="4" loop="1" correct="1" nested="1"/>
<Number x="6" loop="3" correct="2" nested="1"/>
<Number x="8" loop="5" correct="3" nested="1"/>
<Number x="10" loop="7" correct="4" nested="1"/>
```

XQuery 3 - dynamic functions

- Functions may be handled as values and assigned to variables
- All functions can be referred to with their name and cardinality
  - Example: The function count that takes one parameter is count#1

```
let $f := (upper-case#1, lower-case#1)
for $x in 1 to 4
return element Result {$f[$x mod 2 + 1]("STockHOLm")}
```

Result:

```
<Result>stockholm</Result>
<Result>STOCKHOLM</Result>
<Result>stockholm</Result>
<Result>STOCKHOLM</Result>
```

XQuery 3 - inline functions

- Functions may be defined inline (without naming them)

```
let $f := function($a) {sum(let $as := string($a)
                            for $p in 1 to string-length($as)
                            return xs:integer(substring($as, $p, 1))) }
for $x in (1999, 5005005, 123456789)
return element Result {attribute input {$x}, $f($x)}
```

Result:

```
<Result input="1999">28</Result>
<Result input="5005005">15</Result>
<Result input="123456789">45</Result>
```

XQuery 3 - higher-order functions

- **for-each (sequence, function)**
  - **Calls a function for all the items in a sequence**
  - **Return a sequence with the results**

for-each(1 to 4, function($a) {$a*$a})	**Returns (1, 4, 9, 16)**
sum(for-each(1 to 4, function($a) {$a*$a}))	Returns 30
for-each(-4 to 3, abs#1)	**Returns (4, 3, 2, 1, 0, 1, 2, 3)**
sum(for-each(-4 to 3, abs#1))	Returns 16

- **filter (sequence, function)**
  - **Returns only the sequence items that make the function return true**

```
filter(("Maria", "Lisa", "Rebecka", "Ylva", "Evelina"),
        function($s) {contains($s, "e")})
```

Returns the sequence ("Rebecka", "Evelina")

XQuery 3 - Computed Constructors

- **namespace**
  - **namespace prefix URI**
  - **namespace sdxml {"http://ns.dsv.su.se/SDXML"}**
  - **namespace {""} {"http://ns.dsv.su.se/SDXML"}**
  - **Namespaces created in this way may not be used directly**
  - **Use a declaration in the XQuery prolog instead, or hardcode them as xmlns "attributes"**

XQuery Update Facility

- According to XQUF 1.0:
- transform statement
  - copy clause
  - modify clause
    - » delete expression
    - » insert expression
    - » rename expression
    - » replace expression
  - return clause
- Supports nested FLWOR
  - that return update expressions

- keywords for insert
  - » node or nodes
  - » before
  - » after
  - » as first into
  - » as last into
  - » into
- keywords for delete
  - node or nodes
- keywords for replace
  - » (value of) node … with …
- » keywords for rename
  - » node … as …

XQuery - transform

- copy
  - create a copy of an XML document
- modify
  - one or more expressions that update/change the copy
- return
  - usually the copy, but result construction is possible

XQuery - transform - insert

```
copy $x := <root><a>456</a><a>789</a></root>
modify insert node <a>123</a> as first into $x
return $x
```

```
<root>
        <a>123</a>
        <a>456</a>
        <a>789</a>
</root>
```

XQuery - transform - insert

```
copy $x := <root><a>456</a><a>789</a></root>
modify insert node <a>123</a> before $x/a[text() = 789]
return $x
```

```
<root>
        <a>456</a>
        <a>123</a>
        <a>789</a>
</root>
```

XQuery - transform - insert

```
copy $x := <root><a>456</a><a>789</a></root>
modify insert node attribute c {5} into $x/a[text() = 789]
return $x
```

```
<root>
	<a>456</a>
	<a c="5">789</a>
</root>
```

XQuery - transform - delete

```
copy $x := <root><a>456</a><a>789</a></root>
modify delete node $x/a[text() = 789]
return $x
```

```
<root>
	<a>456</a>
</root>
```

XQuery - transform - delete

```
copy $x := <root><a>456</a><a>789</a></root>
modify delete node $x/a
return $x

<root/>
```

XQuery - transform - replace

```
copy $x := <root><a>456</a><a>789</a></root>
modify replace node $x/a[text() = 789] with <b>123</b>
return $x

<root>
	<a>456</a>
	<b>123</b>
</root>
```

XQuery - transform - replace

```
copy $x := <root><a>456</a><a>789</a></root>
modify replace value of node $x/a[text() = 789] with 123
return $x
```

```
<root>
	<a>456</a>
	<a>123</a>
</root>
```

XQuery - transform - replace

```
copy $x := <root><a b="ccc">456</a><a>789</a></root>
modify replace node $x/a[1]/@b with attribute f {"ddd"}
return $x
```

```
<root>
	<a f="ddd">456</a>
	<a>789</a>
</root>
```

XQuery - transform - rename

```
copy $x := <root><a>456</a><a>789</a></root>
modify rename node $x/a[2] as "b"
return $x
```

```
<root>
        <a>456</a>
        <b>789</b>
</root>
```

XQuery - transform - more, loop

```
copy $x := <root><a>456</a><a>789</a></root>
modify for $a in $x/a
            return rename node $a as "b"
return $x
```

```
<root>
        <b>456</b>
        <b>789</b>
</root>
```

XQuery - transform - more, list

```
copy $x := <root><a>456</a><a>789</a></root>
modify (rename node $x/a[1] as "b",
            rename node $x/a[2] as "c")
return $x
```

```
<root>
        <b>456</b>
        <c>789</c>
</root>
```

XQuery - transform - more, list

```
copy $x := <root><a>456</a><a>789</a></root>
modify (rename node $x/a[1] as "b",
             insert node attribute f {"200"} into $x/a[1],
             replace value of node $x/a[1] with "123")
return $x
```

```
<root>
        <b f="200">123</b>
        <a>789</a>
</root>
```

Note! The element b does not exists until after the execution of all the expressions (which is done at the end).

XQuery Update Facility

- According to XQUF 3.0:
- Copy-modify statement
  - As before:
  - copy clause
  - modify clause
  - return clause
- Expressions in return
  - insert
  - delete
  - replace
  - rename
- Transform-with statement
  - … transform with {expressions}
  - Possible expressions:
  - insert
  - delete
  - replace
  - rename
- "Updating expressions"
  - Modify the current node

XQuery - transform with

```
<root><a>456</a><a>789</a></root>
transform with {insert node <a>123</a> as first into .}
```

Same effect as

```
copy $x := <root><a>456</a><a>789</a></root>
modify insert node <a>123</a> as first into $x
return $x
```

```
<root>
	<a>123</a>
	<a>456</a>
	<a>789</a>
</root>
```

XQuery - transform with

```
let $x := <root><a>456</a><a>789</a></root>
return $x transform with {
        insert node <a>123</a> as first into .,
        rename node ./a[2] as "b"
}
```

```
<root>
        <a>123</a>
        <a>456</a>
        <b>789</b>
</root>
```

Note! The second a is based on the original/unmodified structure, since no changes have been applied yet.

XQuery - Updating expressions

```
rename node //Movie[1] as "Picture"
```

The node is modified and nothing is returned.

```
for $m in //Movie
return rename node $m as "Picture"
```

All Movie elements are changed to Picture and nothing is returned.

What is returned, is actually one or more "pending updates". Nothing visible.

XQuery Update Facility - Support

- Supported in BaseX
  - According to XQuery Update Facility 3.0
  - Copy-modify
  - Transform-with
  - Updating expressions
  - Does not change the files, only the in-memory copy.
- Supported in DB2 and Oracle (from version 12)
  - According to XQuery Update Facility 1.0
    - » Copy-modify
  - Some syntactical differences
- Supported in some other products

What to do next

- Quiz about XQuery (Quiz 6)
- XQuery (compendium Querying XML Data with XQuery)
  - Contains more examples
  - Contains information about BaseX and XQuisitor
- Lesson exercises (Lessons 2 & 3)
- Seminar exercises (Assignment 1)
  - Wait until after the lessons