Information Systems

XQuery

Nikolaj Popov

Research Institute for Symbolic Computation
Johannes Kepler University of Linz, Austria
popov@risc.uni-linz.ac.at

What Is XQuery

- The purpose of XQuery is to provide a language for extracting data from XML documents.
- Queries can operate on more than one documents at once. Subsets of nodes can be selected using XPath expressions.
- The query language is functional (but it also includes universal and existential quantifiers), supports simple and complex data types defined in XML Schema.

Simple Examples

vehicles.xml. A Sample XML Document Containing Vehicle Data:

```
<?xml version="1.0" encoding="utf-8"?>
  <vehicles>
    <vehicle year="2007" make="Opel" model="Astra">
      <mileage>13495</mileage>
      <color>green</color>
      <price>13900</price>
      <options>
        <option>navigation system</option>
        <option>heated seats</option>
      </options>
    </vehicle>
...
```

Simple Examples

vehicles.xml. A Sample XML Document Containing Vehicle Data (cont):

```
    <vehicle year="2008" make="Opel" model="Astra">
     <mileage>07541</mileage>
     <color>white</color>
     <price>13900</price>
     <options>
       <option>spoiler</option>
       <option>Traction Control</option>
     </options>
   </vehicle>
   <vehicle year="2007" make="Opel" model="Astra">
     <mileage>18753</mileage>
     <color>white</color>
     <price>12900</price>
     <options />
   </vehicle>

</vehicles>
```

Simple Examples

- The query that retrieves all of the color elements from the vehicles:

```
xquery version "1.0";
for $c in doc("vehicles.xml")//color
return $c
```

- Output:

```
<?xml version="1.0" encoding="UTF-8"?>
<color>green</color>
<color>white</color>
<color>white</color>
```

- Explanation:
  - `doc("vehicles.xml")` is used to open vehicles.xml file.
  - `doc("vehicles.xml")//color` selects all color elements in the document.
  - `$c` is a variable.

Simple Examples

Using filters:

- Return any vehicle elements that contain a `color` element with a value of `green`:

```
xquery version "1.0";
for $v in doc("vehicles.xml")//vehicle[color='green']
return $v
```

- Output:

```
<?xml version="1.0" encoding="UTF-8"?>
<vehicle year="2007" make="Opel" model="Astra">
<mileage>13495</mileage>
<color>green</color>
<price>13900</price>
<options>
  <option>navigation system</option>
  <option>heated seats</option>
</options>
</vehicle>
```

Simple Examples

Using filters:

- Find all of the vehicles with a color of green or a price less than 14000:

```
xquery version "1.0";
for $v in
  doc("vehicles.xml")//vehicle[color='green' or
                               price<'14000']

return $v
```

Simple Examples

Using filters:

- Find `options` of all the white cars:
  `//vehicle[color='white']/options`

Using wildcards:

- Find the `option` elements that are one layer below `vehicle`:

```
 for $o in vehicles/vehicle/*/option
return $o
```

Simple Examples

Referencing attributes:

- Find all the `year` attributes of `vehicle` elements:
  `//vehicle/@year`
- Return all of the vehicle elements that have year attributes as well:
  `//vehicle[@year]`
- Return all the vehicle elements that have a year attribute with the value 2008:

```
for $v in //vehicle[@year="2008"]
return $v
```

Simple Examples

Processing results:

- Query results can be packaged within other surrounding XML code to create transformed data.
- To incorporating query results into surrounding code, query data is put within curly braces (`{}`).
- Access the content within a node by calling the XQuery `data()` function and supplying it with the node in question.
- Query:

```
for $c in //color
return <p>Vehicle color:  {data($c)}</p>
```

- Output:

```
<?xml version="1.0" encoding="UTF-8"?>
<p>Vehicle color:  green</p>
<p>Vehicle color:  white</p>
<p>Vehicle color:  white</p>
```

FLWOR Expressions

- FLWOR is an acronym for “For, Let, Where, Order by, Return”. Reads as “flower”.
- ```
  xquery version "1.0";

    for $v in //vehicle
    let $y := $v/price
    where $v/mileage > '10000'
    order by $y
    return
      {data($v/@model)} - {data($y)}

  ```
- `for` - (optional) binds a variable to each item returned by the `in` expression
- `let` - (optional)
- `where` - (optional) specifies criteria
- `order by` - (optional) specifies the sort-order of the result
- `return` - specifies what to return in the result
  ```

FLWOR Expressions

- The `for` clause binds a variable to each item returned by the `in` expression.
- The `for` clause results in iteration.
- There can be multiple `for` clauses in the same FLWOR expression.
- To loop a specific number of times in a for clause, you may use the `to` keyword:

```
for $x in (1 to 3)
return <test>$x</test>
```

- Returns

```
<test>1</test>
<test>2</test>
<test>3</test>
```

FLWOR Expressions

- It is allowed with more than one in expression in the `for` clause.
- Use comma to separate each in expression:

```
for $x in (10,20), $y in (100,200)
return <test>x=$x and y=$y</test>
```

- Result:

```
<test>x=10 and y=100</test>
<test>x=10 and y=200</test>
<test>x=20 and y=100</test>
<test>x=20 and y=200</test>
```

FLWOR Expressions

- The `let` clause allows variable assignments and avoids repeating the same expression many times.
- The `let` clause does not result in iteration.

```
let $x := (1 to 5)
return <test>$x</test>
```

- Result:

```
<test>1 2 3 4 5</test>
```

FLWOR Expressions

- The `where` clause is used to specify one or more criteria for the result.
- The `order by` clause is used to specify the sort order of the result.
- The `return` clause specifies what is to be returned.

Basic Syntax Rules

- XQuery is case-sensitive
- XQuery elements, attributes, and variables must be valid XML names
- An XQuery string value can be in single or double quotes
- An XQuery variable is defined with a `$` followed by a name, e.g. `$vehicle`
- XQuery comments are delimited by `(:` and `:)`, e.g.

```
(:  XQuery Comment :)
```

Conditional Expressions

```
xquery version "1.0";
<p>
  {
    for $v in //vehicle
    return if (data($v/options) != "")
    then
      <div>Options for $v:  data($v/options)</div>
    else
      <div>The vehicle $v has no options</div>
  }
</p>
```

XQuery Comparisons

- Two ways of comparing values.
  1. General comparisons: `=, !=, <, <=, >, >=`
  2. Value comparisons: `eq, ne, lt, le, gt, ge`
- Difference between the two comparison methods:
- `//vehicle/@year > '2007'`
  Returns true if any `year` attributes have values greater than '2007'.
- `//vehicle/@year gt '2007'`
  Returns true if there is only one `year` attribute returned by the expression, and its value is greater than '2007'. If more than one `year` is returned, an error occurs.

Summary

- XQuery provides a language for extracting data from XML documents.
- Queries can operate on more than one documents at once and may become inputs for other queries.