Information Systems

XQuery

Temur Kutsia

Research Institute for Symbolic Computation
Johannes Kepler University of Linz, Austria
kutsia@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.
- Just as in XSLT, the XPath expressions play the central role in XQuery.
- The value of an expression is always a sequence, having some sequence type.

Simple Examples

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

```
<?xml version="1.0" encoding="utf-8"?>
  <vehicles>
    <vehicle year="2004" make="Acura" model="3.2TL">
      <mileage>13495</mileage>
      <color>green</color>
      <price>33900</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="2005" make="Acura" model="3.2TL">
      <mileage>07541</mileage>
      <color>white</color>
      <price>33900</price>
      <options>
        <option>spoiler</option>
        <option>ground effects</option>
      </options>
    </vehicle>
    <vehicle year="2004" make="Acura" model="3.2TL">
      <mileage>18753</mileage>
      <color>white</color>
      <price>32900</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="2004" make="Acura" model="3.2TL">
  <mileage>13495</mileage>
  <color>green</color>
  <price>33900</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 34000:

```
xquery version "1.0";
for $v in
  doc("vehicles.xml")//vehicle[color='green' or
                                price<'34000']

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 2005:

  ```
  for $v in //vehicle[@year="2005"]
  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>
```

XQuery Processor

- To execute queries, XQuery processor should be isntalled.
- We use the same tool as for XSLT: The SAXON XSLT and XQuery Processor.
  http://saxon.sourceforge.net/.
- XQuery documents are stored in files with a .XQ file extension.
- In addition to the query code, all XQuery documents are required to start with the following line of code:
  ```
  xquery version "1.0";
  ```
- Command that executes XQuery document query.xq on the data file data.xml
  ```
  bin\Query -s data.xml query.xq > out.html
  ```
- (Full path information has to be included for the files involved.)

FLWOR Expressions

- FLWOR is an acronym for “For, Let, Where, Order by, Return”. Reads as “flower”.

-
  ```
  xquery version "1.0";
  <p>
    for $v in //vehicle
    let $y := $v/price
    where $v/mileage > '10000'
    order by $ye
    return
      <div>{data($v/@model)} - {data($y)}</div>
  </p>
  ```
- `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 at $i in //vehicle
    return if (data($v/options) != "")
    then
      <div>Options for $i:  data($v/options)</div>
    else
      <div>The vehicle $i 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 > '2004'`
  Returns true if any `year` attributes have values greater than '2004'.
- `//vehicle/@year gt '2004'`
  Returns true if there is only one `year` attribute returned by the expression, and its value is greater than '2004'. If more than one `year` is returned, an error occurs.

XQuery Functions

- XQuery includes over 100 built-in functions.
- Users can also define their own functions.
- The URI of the XQuery function namespace is: `http://www.w3.org/2005/02/xpath-functions`
- The default prefix for the function namespace is `fn:` (e.g. `fn:string()`)
- Since `fn:` is the default prefix of the namespace, the function names do not need to be prefixed when called.

XQuery Functions

- A call to a function can appear where an expression may appear.
- In an element:
  `<name>{uppercase(//vehicle/@make)}</name>`
- In the predicate of a path expression:
  `//options[substring(option,1,6)='ground']`
- In a let clause:
  `let $name := (substring($option,1,6))`

XQuery Functions

- User-defined functions can be defined in the query or in a separate library.
- Syntax:

```
declare function
  prefix:function_name($parameter AS datatype)
    AS returnDatatype
  {
         (:  ...function code here...  :)
  };
```

- Use the `declare function` keyword.
- The name of the function must be prefixed.
- The data type of the parameters are mostly the same as the data types defined in XML Schema.
- The body of the function must be surrounded by curly braces.

XQuery Functions

- User-defined Function Declared in the Query:

```
declare function local:minPrice(
  $price as xs:decimal?,
  $discount as xs:decimal?)
AS xs:decimal?
{
  let $disc := ($price * $discount) div 100
  return ($price - $disc)
};
```

- An example of how to call this function:

```
<minPrice>{local:minPrice($book/price,
$book/discount)}</minPrice>
```

Quantified Expressions

- Quantified expressions support the universal (`every`) and the existential (`some`) quantifiers.
- The quantifier keyword s followed by possibly several in-clauses that bind sequences to variables, from which a tuple stream is formed as for the FLWOR expressions.
- The final part of the expression is a test-clause, separated by the `satisfies` keyword from the rest of the expression.
- The test-clause is evaluated for each tuple in the stream.
- The `some` expression is true if at least one of the evaluations of the test expression on the tuples from the stream is true (the empty stream yields false value).
- The `every` expression is true if every evaluation of the test expression on the tuples from the stream evaluates to true (the empty stream yields true).

Quantified Expressions

- This expression evaluates to true:

```
xquery version "1.0";
some $vehicle in
doc("vehicles.xml")//vehicles/vehicle
      satisfies $vehicle//option
```

- This expression evaluates to false:

```
xquery version "1.0";
every $vehicle in
   doc("vehicles.xml")//vehicles/vehicle
      satisfies $vehicle//option
```

- This expression evaluates to true (why?):

```
xquery version "1.0";
every $vehicle in
   doc("vehicles.xml")//vehicles/vehicle1
      satisfies $vehicle//option
```

References

For the details on data model and types see the lecture notes and

XQuery.
http://www.w3.org/TR/xquery/

Quick XQuery tutorials:

Michael Morrison.
Sams Teach Yourself XML in 24 Hours, Third Edition
Sams, 2005.

XQuery tutorial.
http://www.w3schools.com/xquery/