Database Management Systems

XML / XPath / XQuery

EMRULLAH GAZİOĞLU, NOVEMBER 2015, ISTANBUL TECHNICAL UNIVERSITY.

What is XPath?

- XPath is a syntax for defining parts of an XML document.
- XPath uses path expressions to navigate in XML documents.
- XPath contains a library of standard functions.

XPath Syntax

- XPath uses path expressions to select nodes or node-sets in an XML document. The node is selected by following a path or steps.

Example XML file:

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
    <book>
        <title lang="en">Harry Potter</title>
        <price>29.99</price>
    </book>
    <book>
        <title lang="en">Learning XML</title>
        <price>39.95</price>
    </book>
</bookstore>
```

XPath Syntax

Expressions

- The most useful path expressions are listed below:

Expression	**Description**
nodename	Selects all nodes with the name "*nodename*"
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

XPath Syntax

Predicates

- Predicates are used to find a specific node or a node that contains a specific value.
- Predicates are always embedded in square brackets.
- In the table below we have listed some path expressions with predicates and the result of the expressions:

Path Expression	**Result**
/bookstore/book[1]	Selects the first book element that is the child of the bookstore element.
/bookstore/book[last()]	Selects the last book element that is the child of the bookstore element
/bookstore/book[last()-1]	Selects the last but one book element that is the child of the bookstore element
/bookstore/book[position()<3]	Selects the first two book elements that are children of the bookstore element
//title[@lang]	Selects all the title elements that have an attribute named lang
//title[@lang='en']	Selects all the title elements that have a "lang" attribute with a value of "en"
/bookstore/book[price>35.00]	Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00
/bookstore/book[price>35.00]/title	Selects all the title elements of the book elements of the bookstore element that have a price element with a value greater than 35.00

XPath Syntax

Selecting Unknown Nodes

- XPath wildcards can be used to select unknown XML nodes.

Wildcard	**Description**
*	Matches any element node
@*	Matches any attribute node
node()	Matches any node of any kind

XPath Syntax

Selecting Several Nodes

- By using the | operator in an XPath expression you can select several paths.

Path Expression	**Result**
//book/title | //book/price	Selects all the title AND price elements of all book elements
//title | //price	Selects all the title AND price elements in the document
/bookstore/book/title | //price	Selects all the title elements of the book element of the bookstore element AND all the price elements in the document

- Example:

```
let $doc := doc('books.xml')/bookstore/book
return $doc//title | $doc//price
```

XPath Operators

- An XPath expression returns either a node-set, a string, a Boolean, or a number.

Operator	Description	Example
|	Computes two node-sets	//book | //cd
+	Addition	6 + 4
-	Subtraction	6 - 4
*	Multiplication	6 * 4
div	Division	8 div 4
=	Equal	price=9.80
!=	Not equal	price!=9.80
<	Less than	price<9.80
<=	Less than or equal to	price<=9.80
>	Greater than	price>9.80
>=	Greater than or equal to	price>=9.80
or	or	price=9.80 or price=9.70
and	and	price>9.00 and price<9.90
mod	Modulus (division remainder)	5 mod 2

What is XQuery?

- XQuery is to XML what SQL is to database tables.
- XQuery is designed to query XML data - not just XML files, but anything that can appear as XML, including databases.
- XQuery is ***the*** language for querying XML data
- XQuery for XML is like SQL for databases
- XQuery is built on XPath expressions
- XQuery is supported by all major databases
- XQuery is a W3C Recommendation

Example:

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title
```

XQuery

- How to open an XML file to query?
  - The doc() function is used to open the "books.xml" file:

```
doc("books.xml")
```

- Path Expressions: XQuery uses path expressions to navigate through elements in an XML document.
  - The following path expression is used to select all the title elements in the "books.xml" file:

```
doc("books.xml")/bookstore/book/title
```

- Predicates: XQuery uses predicates to limit the extracted data from XML documents.

```
doc("books.xml")/bookstore/book[price<30]
```

XQuery - FLWOR

- Acronym for FOR, LET, WHERE, ORDER BY, RETURN

- Example:

```
doc("books.xml")/bookstore/book[price>30]/title
```

- The following FLWOR expression does the same thing exactly:

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
return $x/title
```

XQuery – FLWOR + HTML

- Example:

```
for $x in doc("books.xml")/bookstore/book/title
order by $x
return $x
```

- The following FLWOR expression does the same thing exactly:

```
<ul>
{
  for $x in doc("books.xml")/bookstore/book/title
  order by $x
  return<li>{$x}</li>
}
</ul>
```

XQuery – FLWOR + HTML

- data() function - Example:

```
<ul>
{
  for $x in doc("books.xml")/bookstore/book/title
  order by $x
  return <li>{data($x)}</li>
}
</ul>
```

XQuery – Syntax

- IF-THEN-ELSE:

```
for $x in doc("books.xml")/bookstore/book
return if ($x/@category="CHILDREN") then
<child>{data($x/title)}</child>
else<adult>{data($x/title)}</adult>
```

XQuery – Comparisons

In XQuery there are two ways of comparing values.

1. General comparisons: =, !=, <, <=, >, >=
2. Value comparisons: eq, ne, lt, le, gt, ge

XQuery – Comparisons

- Example:

```
for $item in
doc("books.xml")//bookstore/book
where xs:decimal($item/price) > 30.00
return $item/title
```

- The following expression does the same thing:

```
for $item in
doc("books.xml")//bookstore/book
where xs:decimal($item/price) gt 30.00
return $item/title
```

XQuery – Add HTML elements and Text

- Now, we want to add some HTML elements to the result. We will put the result in an HTML list - together with some text:

```
<html><body>
<h1>Bookstore</h1>
<ul>
{
    for $x in doc("books.xml")/bookstore/book
    order by $x/title
    return
      <li>
          {data($x/title)}
          Category: {data($x/@category)}
      </li>
}</ul>
</body></html>
```

XQuery – Add HTML elements and Text

- Next, we want to use the category attribute as a class attribute in the HTML list:

```
<html><body>
<h1>Bookstore</h1>
<ul>
{
  for $x in doc("books.xml")/bookstore/book
  order by $x/title
  return
  <li
      class="{data($x/@category)}">{data($x/title)}
  </li>
}
</ul>
</body></html>
```

XQuery – Selecting and Filtering

- The FOR clause
- To loop a specific number of times in a for clause, you may use the **to** keyword:

```
for $x in (1 to 5)
return <test>{$x}</test>
```

- The **at** keyword can be used to count the iteration:

```
for $x at $i in
doc("books.xml")/bookstore/book/title
return <book>{$i}. {data($x)}</book>
```

- It is also 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>
```

XQuery – Selecting and Filtering

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

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

- The WHERE clause
- The where clause is used to specify one or more criteria for the result:

```
where $x/price>30 and $x/price<100
```

XQuery – Selecting and Filtering

- **The ORDER BY clause**
- The order by clause is used to specify the sort order of the result. Here we want to order the result by category and title:

```
for $x in doc("books.xml")/bookstore/book
order by $x/@category, $x/title
return $x/title
```

- **The RETURN clause**
- The return clause specifies what is to be returned.

```
for $x in doc("books.xml")/bookstore/book
return $x/title
```

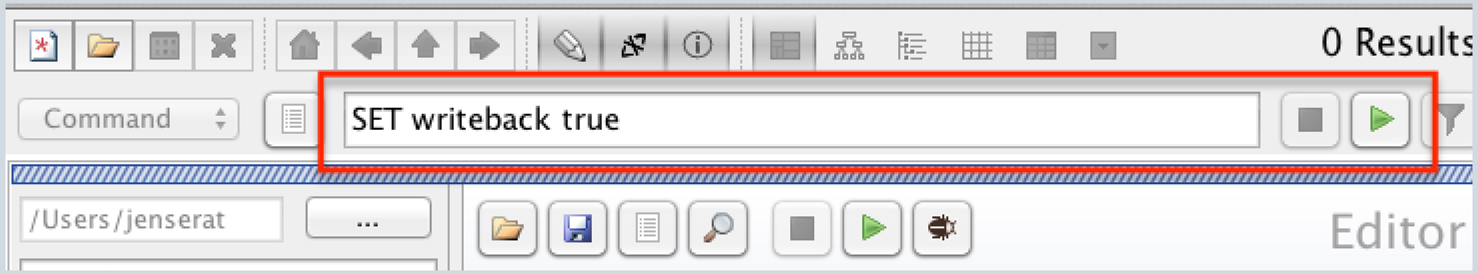
XQuery – Update

In BaseX, all updates are performed on database nodes or in main memory.

By default, update operations do not affect the original input file (the info string "*Updates are not written back*" appears in the query info to indicate this).

To activate WRITEBACK option in the Basex GUI, type the following command in the command input:

```
SET writeback TRUE
```



```
SET writeback true
```

XQuery – Update

Xquery Update Commands:

- Insert
    - Insert into
    - Insert before / after
    - Insert attribute
    - Insert into first / last
- Delete
- Replace
- Rename
- Replace value
- *Transform*

XQuery – Update

Insert

- Insert an `ISBN` element after the `price` element of the first book.

```
insert node <isbn>1111</isbn> after fn:doc("books.xml")/bookstore/book[1]/price
```

- Insert an `ISBN` element before the `price` element of the first book.

```
insert node <isbn>1111</isbn> before fn:doc("books.xml")/bookstore/book[1]/price
```

- Insert an `ISBN` element into the `book[1]` node.

```
insert node <isbn>1111</isbn> into fn:doc("books.xml")/bookstore/book[1]
```

XQuery – Update

Insert

- Insert a new book as the first element into the `bookstore` node.

```
insert node
    <book category="NOVEL">
    <title lang="en">Under the Knife</title>
    <author>Tess Gerritsen</author>
    <year>1990</year>
    <price>20.00</price>
    </book>
as first into fn:doc("books.xml")/bookstore
```

XQuery – Update

Insert

- Insert a new attribute with the name of «gender» and a value of «female» for the author of the first book.

```
insert node attribute gender {'female'} into
fn:doc("books.xml")/bookstore/book[1]/author
```

- Update an existing attribute with a new one.

```
let $doc := doc('books.xml')/bookstore/book[1]
return replace value of node $doc/author/@gender with 'woman'
```

XQuery – Update

Delete

- Delete a node

```
let $doc := doc('books.xml')/bookstore/book[1]
return delete node $doc/author
```

- Delete a node’s attribute.

```
let $doc := doc('books.xml')/bookstore/book[1]
return delete node $doc/author/@gender
```

XQuery – Update

Replace

- Replace a node

```
let $doc := doc('books.xml')/bookstore/book[1]
return replace node $doc/author with <writer>Name Surname</writer>
```

- Replace the value of a node. (Same with updating attribute)

```
let $doc := doc('books.xml')/bookstore/book[1]
return replace value of node $doc/author/@gender with 'woman'
```

XQuery – Update

Rename

- Rename an element

```
let $doc := doc('books.xml')/bookstore/book[1]
return rename node $doc as ‘newBook'
```

- Renaming all `book` elements.

```
for $book in doc("books.xml")//bookstore/book
return rename node $book as 'newBook'
```

XQuery – Update

Transform

- Copies an element into a variable and modifies it. Note that; the original one stays same.

```
copy $c :=doc("books.xml")/bookstore/book[1]
modify (
  replace value of node $c/author with 'Emine S. Beder',
  replace value of node $c/title with 'Everyday Turkish',
  replace value of node $c/year with '2015',
  replace value of node $c/price with '10.00',
  insert node <author>Arda</author> into $c
)
return $c
```