Querying XML

CMT220
Databases & Modelling

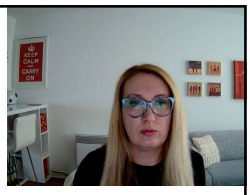
Cardiff School of **Computer Science & Informatics**

http://www.cs.cf.ac.uk

1

1

Lecture



- in this module we learnt about
  - structuring data using a relational data model
  - querying the data stored in relational databases
- in the previous lecture we learnt about using XML to structure data using tags
- in this lecture we will learn how to query such data
- we will cover two languages:
  - XPath a language for navigating through an XML document
  - XQuery a language for querying XML data

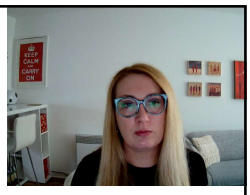
2

2

XPath

3

XPath

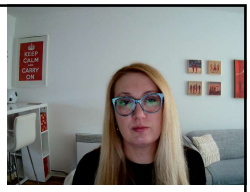


- XPath is used to navigate through elements and attributes in an XML document
- XPath uses path expressions to select nodes in an XML document
  - they look very much like the expressions used when working with a traditional computer file system
- XPath also includes over 100 built–in functions
  - string values, numeric values, date and time comparison, node and QName manipulation, sequence manipulation, Boolean values, etc.

4

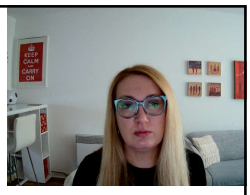
4

Nodes



- in XPath, there are seven kinds of nodes:
  - element
  - attribute
  - text
  - namespace
  - processing instruction
  - comment
  - document node
- XML documents are treated as trees of nodes
- the top-most element of the tree is called the root element

Nodes



```
<?xml version="1.0" encoding="UTF-8"?>
```

root --------> `<bookstore>`

attribute

```
  <book>
    <title lang="en">Harry Potter</title>
```

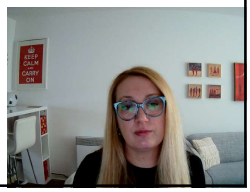
element ----------> `<author>J K. Rowling</author>`

```
    <year>2005</year>
    <price>29.99</price>
  </book>
</bookstore>
```

Atomic values

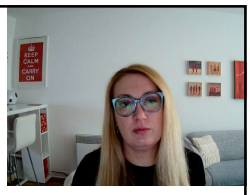
```
<bookstore>
 <book>
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
 </book>
</bookstore>
```

- atomic values are nodes with no children or parent
- e.g.
  - J K. Rowling
  - "en"



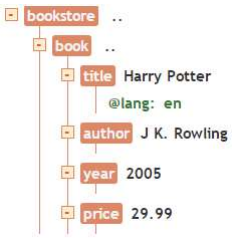
7

Relationships between nodes



- parent
- child
- sibling
- ancestor
- descendant

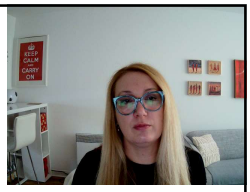
```
<bookstore>
 <book>
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
 </book>
</bookstore>
```



8

8

XPath syntax



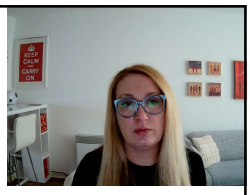
- a node is selected by following a path
- we will use the following example to illustrate the use of paths:

```
<?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>
```

9

9

Path expressions

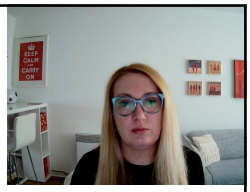


Expression	Description
nodename	select all nodes with the name *nodename*
/	select from the root node
//	select all nodes descending from the current node that match the selection criteria
.	select the current node
..	selects the parent of the current node
@	select attribute

10

10

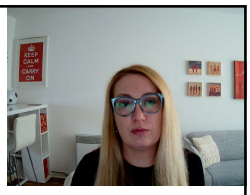
Examples



Path expression	Comment
bookstore	select all nodes with the name *bookstore*
/bookstore	select the root element *bookstore*
bookstore/book	selects all book elements that are children of bookstore
bookstore//book	selects all book elements that are descendant of the bookstore element
//book	select all book elements no matter where they are
//@lang	select all attributes that are named *lang*

11

Predicates



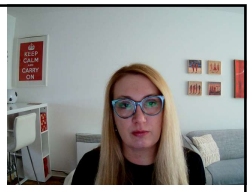
- predicates are used to find:
  - a specific node, or
  - a node that contains a specific value
- predicates are embedded in square brackets
  - e.g. select the first book element that is the child of the bookstore element

    /bookstore/book[1]

12

12

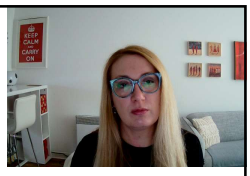
Examples



Path expression	Comment
/bookstore/book[last()]	select the last book element that is the child of the bookstore element
/bookstore/book [position()<3]	select the first two book elements that are children of the bookstore element
//title[@lang]	select all title elements that have an attribute named lang
//title[@lang='en']	select all title elements that have a "lang" attribute with a value of "en"
/bookstore/book [price>35.00]/title	select all title elements of the book elements of the bookstore element that have a price element with a value >35.00

13

Unknown nodes



- XPath wildcards can be used to select unknown XML nodes

Wildcard	Description
*	match any element node
@*	match any attribute node
node()	match any node

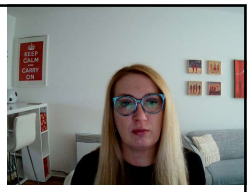
- e.g.

Path expression	Comment
/bookstore/*	selects all elements that are children the bookstore element
//*	selects all elements in the document
//title[@*]	selects all title elements that have at least one attribute

14

14

Multiple paths



- operator | can be used within an XPath expression to select multiple paths, e.g.

Path expression	Comment
//book/title | //book/price	select all title AND price elements of all book elements
//title | //price	selects all title AND price elements in the document
/bookstore/book/title | //price	select all title elements of the book element of the bookstore element AND all the price elements in the document

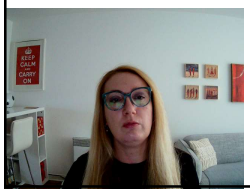
15

15

XPath axis

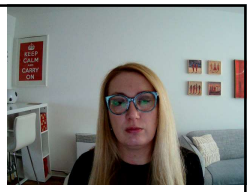
- an axis defines a node–set relative to the current node

Axis name	Description
self	the current node
attribute	all attributes of the current node
namespace	all namespace nodes of the current node
parent	the parent of the current node
child	all children of the current node
ancestor	all ancestors of the current node
ancestor–or–self	as above + the current node itself
descendant	all descendants of the current node
descendant–or–self	as above + the current node itself
following	everything in the document after the closing tag of the current node
following–sibling	all siblings after the current node
preceding	all nodes that appear before the current node in the document, except ancestors, attribute nodes and namespace nodes
preceding–sibling	all siblings before the current node



16

Location path



- a location path can be absolute or relative
  - an absolute location path starts with a slash (/)
  - a relative location path does not start with a slash
- a location path consists of one or more steps, each separated by a slash, e.g.
  - /step/step/... absolute location path
  - step/step/... relative location path
- each step is evaluated against the nodes in the current node–set

17

17

Location path

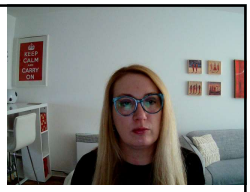
- a step in a location path consists of:
  - an axis
  - a node–test ... identifies a node–set within an axis
  - ≥0 predicates ... to further refine the selected node–set
- the syntax for a location step is:

```
axisname::nodetest[predicate]
```

18

18

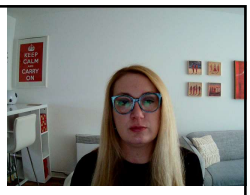
Examples



Location path	Comment
child::book	all book nodes that are children of the current node
attribute::lang	the lang attribute of the current node
attribute::*	all attributes of the current node
child::node()	all children of the current node
child::*	all elements that are children of the current node
child::text()	all text node children of the current node
descendant::book	all book nodes that are descendants of the current node
child::*/child::price	all price grandchildren of the current node

19

XPath operators

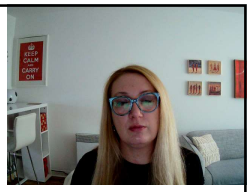


- an XPath expression can return:
  - node–set
  - string
  - Boolean
  - number
- these returned values may be combined using the XPath operators

20

20

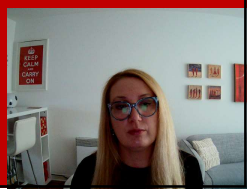
Operator	Description	Example
|	union of two node–sets	//book | //cd
+	addition	6 + 4
–	subtraction	6 – 4
*	multiplication	6 * 4
div	division	8 div 4
mod	division remainder	5 mod 2
=	equal	price=9.80
!=	not equal	price!=9.80
<	less than	price<9.80
<=	less than or equal to	price<=9.80
>	greater than	price>9.80
>=	greater than or equal to	price>=9.80
or	logical or	price=9.80 or price=9.70
and	logical and	price>9.00 and price<9.90



21

21

XQuery



22

XQuery

- a language for finding and extracting elements and attributes from XML documents
- XQuery is to XML what SQL is to database tables
- designed to query XML data
- built on XPath expressions

Select all books with a price greater than £30 from the book collection stored in books.xml

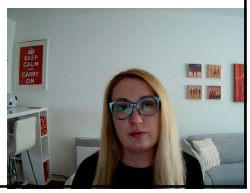
```
for      $x in doc("books.xml")/bookstore/book
where    $x/price>30
order by $x/title
return   $x/title
```

23

23

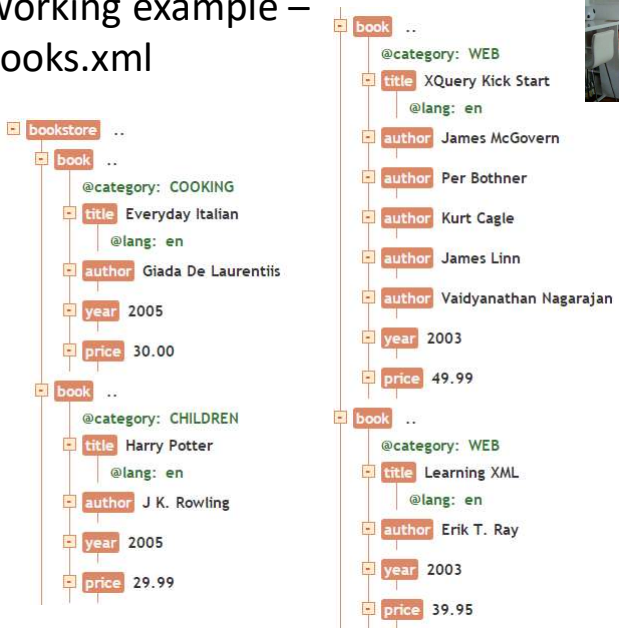
XQuery syntax

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

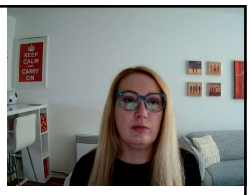


24

Working example – books.xml



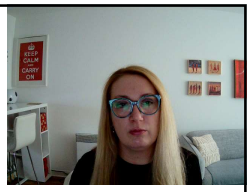
Selecting nodes



- XQuery uses:
  - functions ... to extract data from XML documents
  - path expressions ... to navigate through elements in an XML document
  - predicates ... to limit the extracted data from XML documents
- e.g. doc("books.xml")/bookstore/book[price<30]

function path predicate

FLWOR expressions



- e.g. path expression:

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

- result:

```
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```

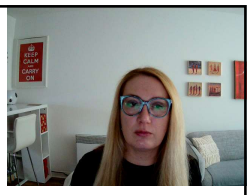
- the following FLWOR expression does exactly the same:

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

27

27

FLWOR expressions



- with FLWOR we can sort the result, e.g.

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

- result:

```
<title lang="en">Learning XML</title>
<title lang="en">XQuery Kick Start</title>
```

28

28

FLWOR expressions

- FLOWR expression is to XQuery what SELECT statement is to SQL
- FLWOR stands for **F**or, **L**et, **W**here, **O**rder by, **R**eturn
- only return is mandatory

Clause	Description
for	binds a variable to each item returned by the in expression
let	assigns variables
where	specifies search criteria
order by	specifies the sort order of the result
return	specifies what to return in the result

29

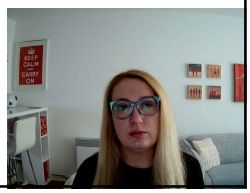
29

The `for` clause

- the `for` clause binds a variable to each item returned by the `in` expression
- multiple `for` clauses can be used in the same FLWOR expression
- the `for` clause results in iteration
- the `at` keyword can be used to count the iterations, e.g.

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

```
<book>1. Everyday Italian</book>
<book>2. Harry Potter</book>
<book>3. XQuery Kick Start</book>
<book>4. Learning XML</book>
```



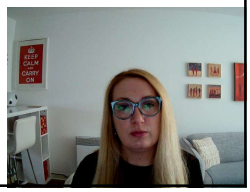
30

The `let` clause

- the `let` clause allows variable assignments
- ... to avoid repeating the same expression many times
- the let clause does **not** result in iteration
- example:

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

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

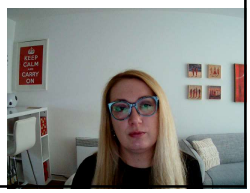


31

The `where` clause

- the `where` clause is used to specify one or more criteria for the result
- example:

```
for     $x in doc("books.xml")/bookstore/book
where $x/price>30 and $x/price<100
return $x/title
```



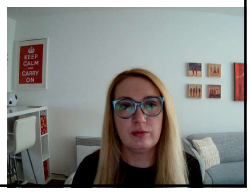
32

The order clause

- the `order by` clause is used to specify the sort order of the result
- e.g. order the result by category and title:

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

```
<title lang="en">Harry Potter</title>
<title lang="en">Everyday Italian</title>
<title lang="en">Learning XML</title>
<title lang="en">XQuery Kick Start</title>
```



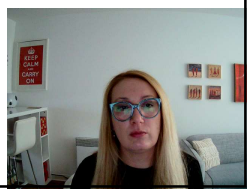
33

The return clause

- the `return` clause specifies what is to be returned
- example:

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

```
<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```



34

Conditional expressions

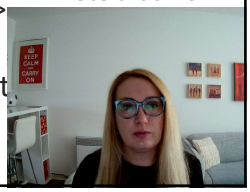
- if–then–else expressions are allowed in XQuery
- parentheses around the if expression are required
- else is required, but it can be just else ()
- e.g.

```
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>
```

note that we

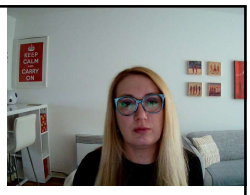
- result:

```
<adult>Everyday Italian</adult>
<child>Harry Potter</child>
<adult>XQuery Kick Start</adult>
<adult>Learning XML</adult>
```



35

Comparisons



- there are two ways of comparing values:

1. general comparison =, !=, <, <=, >, >=
2. value comparison eq, ne, lt, le, gt, ge

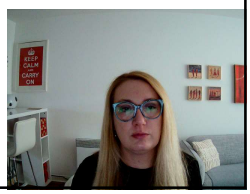
- examples:
- $bookstore//book/@q > 10
  - returns true if any q attributes have a value >10
- $bookstore//book/@q gt 10
  - returns true if there is only one q attribute returned by the expression, and its value is >10
  - if more than one q is returned, an error occurs

36

36

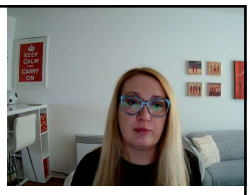
Functions

- XQuery and XPath share the same data model and support the same functions and operators



37

Operator	Description	Example
|	union of two node–sets	//book | //cd
+	addition	6 + 4
–	subtraction	6 – 4
*	multiplication	6 * 4
div	division	8 div 4
mod	division remainder	5 mod 2
=	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	logical or	price=9.80 or price=9.70
and	logical and	price>9.00 and price<9.90



38

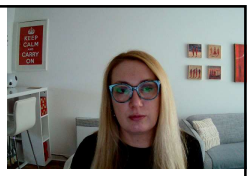
38

Function type	Example	Comment
accessor	fn:base-uri(node)	returns the value of the base–uri property of the specified node
error and trace	fn:trace(value, label)	used to debug queries
numeric	fn:round(num)	rounds the number argument to the nearest integer
string	fn:concat(string, string, ...)	returns the concatenation of the strings
anyUri	fn:resolve-uri(relative, base)	takes a base URI and a relative URI as arguments, and constructs an absolute URI
Boolean	fn:not(arg)	logical not
duration/date/ time	fn:dateTime(date,time)	converts the arguments to a date and a time
QName	fn:QName(uri, name)	takes a namespace URI and a qualified name as arguments, and constructs a QName value
node	fn:root(node)	returns the root of the tree to which the specified node belongs.
sequence	fn:reverse((item, item, ...))	returns the reversed order of the items specified
context	fn:position()	returns the index position of the node that is currently being processed

39

39

User–defined functions



- users can also define their own functions in XQuery:

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

- use the `declare function` keywords
- the name of the function must be prefixed
- the data types are defined in XML Schema
- the function body must be surrounded by curly braces

40

40

User–defined functions

- example:

```
declare function local:minPrice($p as xs:decimal?,
$d as xs:decimal?)
as xs:decimal?
{
  let $disc := ($p * $d) div 100
  return ($p – $disc)
};
```

- function call:

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

41

41