Data Bases II
XPath

Michele Beretta

michele.beretta@unibg.it

Query languages for XML

XML can be considered a *semi-structured data model*, andd a set of XML documents can be considered a *large data collection*.

We can **query** XML with specific languages:

- **XPath**, a simple selection language
- **XQuery**, a rich query language
- **XSLT**, used for document transformations (not discussed)

XPath

Uses expression to select nodes in an XML document.

Expression	Description
Node name	Select all child nodes of the node
/	Select from the root node
//	Select nodes from the current node no matter where they are
,	Select the current node
..	Select the parent node
@	Select attributes

XPath – path expressions

A path expression starts from the root of the document, e.g., `doc("books.xml")` returns the root element and all its content.

Starting from the root it's possible to express path expressions such as

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

which returns the **sequence** of all book elements in the document.

Example in the next slide.

XPath – path expressions

```
<bookstore>
  <book available="Y">
    <title>Il Signore degli Anelli</title>
    <author>J.R.R. Tolkien</author>
    <publisher>Bompiani</publisher>
  </book>
  <book available="N">
    <title>Il nome della rosa</title>
    <author>Umberto Eco</author>
    <publisher>Bompiani</publisher>
  </book>
  <book available="Y">
    <title>Metamorfosi</title>
    <author>F. Kafka</author>
    <publisher>Feltrinelli</publisher>
  </book>
</bookstore>
```

After running `/bookstore/book` we get

```
<book available="Y">
  <title>Il Signore degli Anelli</title>
  <author>J.R.R. Tolkien</author>
  <publisher>Bompiani</publisher>
</book>
<book available="N">
  <title>Il nome della rosa</title>
  <author>Umberto Eco</author>
  <publisher>Bompiani</publisher>
</book>
<book available="Y">
  <title>Metamorfosi</title>
  <author>F. Kafka</author>
  <publisher>Feltrinelli</publisher>
</book>
```

XPath – path expressions

```
<bookstore>
  <book available="Y">
    <title>Il Signore degli Anelli</title>
    <author>J.R.R. Tolkien</author>
    <publisher>Bompiani</publisher>
  </book>
  <book available="N">
    <title>Il nome della rosa</title>
    <author>Umberto Eco</author>
    <publisher>Bompiani</publisher>
  </book>
  <book available="Y">
    <title>Metamorfosi</title>
    <author>F. Kafka</author>
    <publisher>Feltrinelli</publisher>
  </book>
</bookstore>
```

After running

`/bookstore/book[publisher="Bompiani"]/title`

we get

```
<title>Il Signore degli Anelli</title>
<title>Il nome della rosa</title>
```

XPath – path expressions

```
<bookstore>
  <book available="Y">
    <title>Il Signore degli Anelli</title>
    <author>J.R.R. Tolkien</author>
    <publisher>Bompiani</publisher>
  </book>
  <book available="N">
    <title>Il nome della rosa</title>
    <author>Umberto Eco</author>
    <publisher>Bompiani</publisher>
  </book>
  <book available="Y">
    <title>Metamorfosi</title>
    <author>F. Kafka</author>
    <publisher>Feltrinelli</publisher>
  </book>
</bookstore>
```

After running `//author` we get

```
<author>J.R.R. Tolkien</author>
<author>Umberto Eco</author>
<author>F. Kafka</author>
```

XPath – path expressions

```
<bookstore>
  <book available="Y">
    <title>Il Signore degli Anelli</title>
    <author>J.R.R. Tolkien</author>
    <publisher>Bompiani</publisher>
  </book>
  <book available="N">
    <title>Il nome della rosa</title>
    <author>Umberto Eco</author>
    <publisher>Bompiani</publisher>
  </book>
  <book available="Y">
    <title>Metamorfosi</title>
    <author>F. Kafka</author>
    <publisher>Feltrinelli</publisher>
  </book>
</bookstore>
```

After running `/bookstore/book[2]` we get

```
<book available="N">
  <title>Il nome della rosa</title>
  <author>Umberto Eco</author>
  <publisher>Bompiani</publisher>
</book>
```

XPath – path expressions

```
<bookstore>
  <book available="Y">
    <title>Il Signore degli Anelli</title>
    <author>J.R.R. Tolkien</author>
    <publisher>Bompiani</publisher>
  </book>
  <book available="N">
    <title>Il nome della rosa</title>
    <author>Umberto Eco</author>
    <publisher>Bompiani</publisher>
  </book>
  <book available="Y">
    <title>Metamorfosi</title>
    <author>F. Kafka</author>
    <publisher>Feltrinelli</publisher>
  </book>
</bookstore>
```

After running `/bookstore/book[2]/*` we get

```
<title>Il nome della rosa</title>
<author>Umberto Eco</author>
<publisher>Bompiani</publisher>
```

This returns all elements (with any tagname) contained in the second book.

XPath – path expressions

All we have seen, but more rigidly described.

Given `e1/e2`, the semantics are:

1. Evaluate `e1` and return a sequence of nodes
2. For each node in this sequence
    1. Bind `.` to that node
    2. Evaluate `e2` with this binding
3. Concatenate the partial results
4. Eliminate duplicates
5. Sort by document order

XPath – axes

An axis defines a node-set relative to the current node.

- `ancestor`: selects all ancestors (parent, grandparent, etc.) of the current node
- `ancestor-or-self`: selects all ancestors (parent, grandparent, etc.) of the current node and the current node itself
- `attribute`: selects all attributes of the current node
- `child`: selects all children of the current node
- `descendant`: selects all descendants (children, grandchildren, etc.) of the current node
- `descendant-or-self`: selects all descendants (children, grandchildren, etc.) of the current node and the current node itself
- `following`: selects everything in the document after the closing tag of the current node
- `following-sibling`: selects all siblings after the current node
- `parent`: selects the parent of the current node
- `preceding`: selects everything in the document that is before the start tag of the current node
- `preceding-sibling`: selects all siblings before the current node
- `self`: selects the current node

XPath – axes

Axes can be missing (by default, child):

- `$x/child::person` is equivalent to `$x/person`

Short-hands:

- `$x/descendant-or-self::*/child::name` → `$x//name`
- `$x/parent::*` → `$x/..`
- `$x/attribute::year` → `$x/@year`
- `$x/self::*` → `$x/.`

XPath – expression and predicates

`/bookstore`	Select the root element bookstore
`bookstore/book`	Select all book elements that are children of bookstore
`//book`	Select all book elements no matter where they are
`bookstore//book`	Select all book elements that are *descendants* of bookstore
`//@lang`	Select all attributes that are named lang
`/bookstore/book[1]`	Select the first book element that is a child of bookstore
`/bookstore/book[=1]`	As above
`/bookstore/book[last()]`	Select the last book element that is a child of bookstore
`/bookstore/book[last()-1]`	Self explanatory
`/bookstore/book[position()<3]`	Select the first two book elements that are children of bookstore
`/bookstore/book[<3]`	As above
`//title[@lang="en"]`	Select all titles that have a `lang` attribute with a value of `en`
`/bookstore/book[price]`	Select all books that have a price element
`/bookstore/book[price>35]`	Select all books that have a price element with a value greater than 35

XPath – more about predicates

With `[]` you can:

- Filter by position
  - `//book[3]`
  - `//book[3]/author[1]`
  - `//book[3]/author[2 to 4]`
- Filter by predicate
  - `//book[author/firstname = "ronald"]`
  - `//book[@price < 25]`
  - `//book[count(author[@gender = "female"]) > 0]`
- Do an existential filter
  - `//book[author]`

XPath – wildcards

XPath allows for some wildcards.

- `*` matches any element node
- `@*` matches any attribute node
- `node()` matches any node of any kind

XPath – alternative paths

You can specify alternatives with `|`.

For example

`//book/title | //book/price`	Select all `title` and `price` elements of all books
`//title | //price`	Select 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 `price` elements in the document