XML Query Languages

© Leonidas Fegaras
University of Texas at Arlington

Query Languages for XML

- Need a language for XML data for
  - extracting fragments (querying)
  - restructuring (data transformation)
  - integrating (e.g., combining multiple XML documents)
  - browsing
  - presentation (e.g., from XML to HTML)
- We will first learn XPath
  - used in extracting fragments from a single document
  - many XML query languages are based on XPath
- We will then learn XSLT
  - for extracting, restructuring, and presentation over a single document
- We will focus later on XQuery
  - a full-fledged query language
  - much like SQL

XPath

- Describes a single navigation path in an XML document
- Selects a sequence of nodes reachable by the path
  - the order of nodes is the document order (which is the preorder of the XML tree: every node occurs before its children)
- Main construct: axis navigation
- The / step returns the document root (the entire document)
- An XPath consists of one or more navigation steps separated by /
- A navigation step is a triplet
  
  axis :: node-test [ predicate ]

- Each navigation path is evaluated relative to a context node
- Examples:
  
  / child::bib / descendant::author
  / descendant::book [ child::author / child::name = 'Smith' ] / child::title

Most people use shorthands

/bib/author

/book[author/name='Smith']/title

Axis

- Forward axes
  - child
  - descendant
  - attribute
  - self
  - descendant-or-self
  - following-sibling
  - following
- Reverse axes
  - parent
  - ancestor
  - preceding-sibling
  - preceding
  - ancestor-or-self
Node Test

- **person** any element node whose name is person
- **@*** any element node regardless of its name
- **@price** any attribute whose name is price
- **@x** any attribute, regardless of its name
- **node()** any node
- **text()** any text node
- **string()** the text content of the node at any depth
- **element()** any element node
- **element(person)** any element node whose tagname is person
- **element(person, surgeon)** any element node whose tagname is person, and whose type annotation is surgeon
- **attribute()** any attribute node
- **attribute(price)** any attribute whose name is price
- **attribute(*, xs:decimal)** any attribute whose type annotation is xs:decimal, regardless of its name.

Abbreviated Syntax

- The attribute axis child:: can be omitted
  - `section/para` is an abbreviation for `child::section/child::para`
  - `section/@id` is an abbreviation for `child::section/attribute::id`
- ... unless the axis step contains an attribute test; then the default axis is attribute
  - `section/attribute(id)` is shorthand for `child::section/attribute::attribute(id)`
- The attribute axis attribute:: can be abbreviated by @
  - `para[@type="warning"]` is shorthand for `child::para[attribute::type="warning"]`
- // is replaced by `/descendant-or-self::node()/
  - `div/para` is shorthand for `child::div/descendant-or-self::node()/child::para`, or better child::div/descendant::para
- .. is short for `parent::node()`
  - `.//title` is short for `parent::node()/child::title`

Most Common Steps

<table>
<thead>
<tr>
<th>XPath step</th>
<th>full XPath syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>@tagname</td>
<td>/ child::@tagname</td>
</tr>
<tr>
<td>/*</td>
<td>/ child::any</td>
</tr>
<tr>
<td>/@tagname</td>
<td>/ descendant::@tagname</td>
</tr>
<tr>
<td>/@</td>
<td>/ descendant::any</td>
</tr>
<tr>
<td>/@@attname</td>
<td>/ attribute::@attname</td>
</tr>
<tr>
<td>/@*</td>
<td>/ attribute::any</td>
</tr>
<tr>
<td></td>
<td>self::node()</td>
</tr>
<tr>
<td>..</td>
<td>parent::node()</td>
</tr>
</tbody>
</table>

Examples:

- `/book/chapter/section`
- `/chapter/*`
- `/book/author/@*`

Semantics

**step1 / step2**

- Evaluate step1 to get a sequence of nodes
- Bind current context (.) to each node in the sequence
- Evaluate step2 using this binding; get a new sequence of nodes
- Concatenate all these sequences into one sequence
- Eliminate duplicates
- Sort by document order
Atomization

- How do we extract a value from a node?
  - //gradstudent[gpa > 3.5]/name
- Arithmetic expressions/comparisons extract the value of the node before the apply the operation using fn:data
  - eg: fn:data(<a>2</a>) = "2"
- Strings are cast to numerical values
  - eg: "3" < 4 is true
- Effective boolean value:
  - () \( = 0 \) fn:false() are all false
  - fn:true() is true
  - any nonzero number or nonempty string is true
  - any nonempty sequence where the first item is a node is true
  - otherwise, error

Predicates

- Existential semantics:
  - true, if the resulting sequence is not empty
- Many variations
  - //book[10]
  - //book[last]
  - //book[author]
  - //book[author/name]
  - //book[author/name="Smith"]
  - //book[price>35.0]
- Examples
  - /bib/book[@price < 100]/title
  - /bib/book[author/text()]
  - //author[name/firstname="John",lastname="Smith"]/title
  - /bib//author[name/firstname][address/zip=12345][city]/name/lastname

Using XPath in Java

```java
import javax.xml.xpath.*;
import org.xml.sax.InputSource;

class XPath {
    public static void main ( String[] args ) throws Exception {
        String xpath_query = "//gradstudent[name/lastname="Smith"]/name";
        XPathFactory xpathFactory = XPathFactory.newInstance();
        XPath xpath = xpathFactory.newXPath();
        InputSource inputSource = new InputSource("cs.xml");
        XPathExpression query = xpath.compile(xpath_query);
        String result = query.evaluate(inputSource);
        System.out.println(result);
    }
}
```

XSL Transformation

- XSL stands for eXtensible Stylesheet Language
- XSLT: XSL Transformations
- XSLT is a stylesheet specification language for converting XML documents into various forms (XML, HTML, plain text)
- Can transform each XML element into another element, add new elements into the output file, or remove elements
- Can rearrange and sort elements, test and make decisions about which elements to display, and much more
- Uses XPath to navigate through a document:

```xml
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <students>
            <xsl:copy-of select="/student/name"/>
        </students>
    </xsl:stylesheet>
```
XSLT Templates

- XSL uses XPath to define parts of the source document that match one or more predefined templates.
- When a match is found, XSLT will transform the matching part of the source document into the result document.
- The parts of the source document that do not match a template will end up unmodified in the result document (they will use the default templates).

Form:

```xml
<xsl:template match="XPath expression">
  ...
</xsl:template>
```

The default (implicit) templates visit all nodes and strip out all tags:

```xml
<xsl:template match="" | /">
  <xsl:apply-templates/>
</xsl:template>
```

```xml
<xsl:template match="text() | @*">
  <xsl:value-of select="."/>
</xsl:template>
```

Copy the Entire Document

```xml
<xsl:stylesheet version="1.0" xmlns:xsl=http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="text()">
    <xsl:value-of select="."/>
  </xsl:template>
  <xsl:template match="@*">
    <xsl:element name="name(.)">
      <xsl:apply-templates/>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

Other XSLT Elements

- `<xsl:value-of select="XPath expression"/>
  select the value of an XML element and add it to the output stream of the transformation, e.g. `<xsl:value-of select="/books/book/author/"/>
- `<xsl:copy-of select="XPath expression"/>
  copy the entire XML element to the output stream of the transformation
- `<xsl:apply-templates match="XPath expression"/>
  apply the template rules to the elements that match the XPath expression
- `<xsl:element name="XPath expression"/>
  ... `<xsl:element>
  add an element to the output with a tag-name derived from the XPath expression

Example:

```xml
<xsl:stylesheet version="1.0" xmlns:xsl=http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="employee">
    <b>
      <xsl:apply-templates select="node()"/>
    </b>
  </xsl:template>
</xsl:stylesheet>
```

More on XSLT

- **Conflict resolution**: more specific templates overwrite more general templates. Templates are assigned default priorities, but they can be overwritten using priority="n" in a template.
- **Modes can be used to group together templates. No mode is an empty mode:**
  ```xml
  <xsl:template match="..." mode="A"/>
  <xsl:apply-templates mode="B"/>
  </xsl:template>
  ```
- **Conditional and loop statements:**
  ```xml
  <xsl:if test="XPath predicate"> body </xsl:if>
  <xsl:for-each select="XPath"> body <xsl:for-each/>
  ```
- **Variables can be used to name data:**
  ```xml
  <xsl:variable name="x" value="value"/>
  Variables are used as ($x) in XPaths.
  ```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL-Transform">
  <xsl:template match="/">
    <html xmlns="http://www.w3.org/1999/xhtml">
      <head>
        <title>Best Graduate Students</title>
      </head>
      <body>
        <h1>Best Graduate Students</h1>
        <table border="1">
          <tr><th>First Name</th><th>Last Name</th><th>Department</th><th>Graduation Year</th></tr>
          <tr><td>Aaron</td><td>Aaron</td><td>Computer Science</td><td>2005</td></tr>
          <tr><td>Bob</td><td>Bob</td><td>Mathematics</td><td>2006</td></tr>
          <tr><td>Cathy</td><td>Cathy</td><td>Physics</td><td>2007</td></tr>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>