XPath & XQuery

CPS 196.3
Introduction to Database Systems

Query languages for XML

- **XPath**
  - Path expressions with conditions
  - Building block of other standards (XQuery, XSLT, XLink, XPointer, etc.)
- **XQuery**
  - XPath + full-fledged SQL-like query language
- **XSLT**
  - XPath + transformation templates

Example DTD and XML

```xml
<?xml version="1.0"?>
<!DOCTYPE bibliography [
<!ELEMENT bibliography (book+)>]
<!ELEMENT book (title, author*, publisher?, year?, section*)>
<!ATTLIST book ISBN CDATA #REQUIRED>
<!ATTLIST book price CDATA #IMPLIED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT section (title, (#PCDATA)?)>
]>
<bibliography>
  <book ISBN="ISBN-10" price="80.00">
    <title>Foundations of Databases</title>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vianu</author>
    <publisher>Addison Wesley</publisher>
    <year>1995</year>
    <section>…</section>…
  </book>
</bibliography>
```
A tree representation

XPath

* XPath specifies path expression that match XML data by navigating down (and occasionally up and across) the tree
* Example
  * Query: /bibliography/book/author
    * Like a UNIX directory
    * Result: all author elements reachable from root via the path /bibliography/book/author

Basic XPath constructs

/   separator between steps in a path
name  matches any child element with this tag name
*    matches any child element
@name  matches the attribute with this name
@*    matches any attribute
//    matches any descendent element (including the current element itself)
.    matches the current element
..   matches the parent element
Simple XPath examples

- All book titles
  `/bibliography/book/title`
- All book ISBN numbers
- All title elements, anywhere in the document
  `//title`
- All section titles, anywhere in the document
- Authors of bibliographical entries (suppose there are articles, reports, etc. in addition to books)
  `/bibliography/*/author`

Predicates in path expressions

- [condition] matches the current element if condition evaluates to true on the current element
- Books with price lower than $50
  `/bibliography/book[@price<50]`
  - XPath will automatically convert the price string to a numeric value for comparison
- Books with author “Abiteboul”
- Books with a publisher child element
  `/bibliography/book[publisher]`
- Prices of books authored by “Abiteboul”

More complex predicates

Predicates can have and's and or's
- Books with price between $40 and $50
  `/bibliography/book[40<=@price and @price<=50]`
- Books authored by “Abiteboul” or those with price lower than $50
Predicates involving node-sets

/bibliography/book[author='Abiteboul']

- There may be multiple authors, so author in general returns a node-set (in XPath terminology)
- The predicate evaluates to true as long as it evaluates true for at least one node in the node-set, i.e., at least one author is “Abiteboul”
- Tricky query
  /bibliography/book[author='Abiteboul' and author!='Abiteboul']
  - Will it return any books?

XPath operators and functions

Frequently used in conditions:
- +, -, *, div, mod
- contains(\textit{x}, \textit{y}) true if \textit{x} contains \textit{y}
- count(node-set) counts the number nodes in node-set
- position() returns the position of the current node in the currently selected node-set
- last() returns the size of the currently selected node-set
- name() returns the tag name of the current element

More XPath examples

- All elements whose tag names contain “section” (e.g., “subsection”)
  //*[contains(name(), ‘section’)]
- Title of the first section in each book
  /bibliography/book/section[position()=1]/title
  - A shorthand: /bibliography/book/section[1]/title
- Title of the last section in each book
  /bibliography/book/section[position()=last()]/title
- Books with fewer than 10 sections
  /bibliography/book[count(section)<10]
- All elements whose parent’s tag name is not “book”
De-referencing IDREF’s

`id(identifier)` returns the element with the unique `identifier`.

- Suppose that books can make references to other books:
  ```xml
  <section><title>Introduction</title>
  XML is a hot topic these days; see <bookref ISBN="ISBN-10"/> for more details...
  </section>
  ```
- Find all references to books written by "Abiteboul" in the book with "ISBN-10":
  ```xml
  //bookref[id(@ISBN)/author='Abiteboul']
  ```

General XPath location steps

- Technically, each XPath query consists of a series of location steps separated by `/`.
- Each location step consists of:
  - An axis: one of `self`, `attribute`, `parent`, `child`, `ancestor`, `ancestor-or-self`, `descendent`, `descendent-or-self`, `following`, `following-sibling`, `preceding`, `preceding-sibling`, and `namespace`.
  - A node test: either a name test (e.g., `book`, `section`, `*`) or a type test (e.g., `text()`, `node()`, `comment()`), separated from the axis by `::`
  - Zero or more predicates (or conditions) enclosed in square brackets.

Example of verbose syntax

Verbose (axis, node test, predicate):
```xml
/child::bibliography
/descendent-or-self::node()
/child::title
```

Abbreviated:
```xml
```
- `child` is the default axis
- `//` stands for `/descendent-or-self::node()`
XQuery

- XPath + full-fledged SQL-like query language
- XQuery expressions can be
  - XPath expressions
  - FLWR (:block) expressions
  - Quantified expressions
  - Aggregation, sorting, and more…

A simple XQuery based on XPath

Find all books with price lower than $50
<result>
{  
document("bib.xml")/bibliography/book[price<50]
}
</result>

- Things outside {}'s are copied to output verbatim
- Things inside {}'s are evaluated and replaced by the results
  - document("bib.xml") specifies the document to query
  - The XPath expression returns a set of book elements
  - These elements (including all their descendents) are copied to output

FLWR expressions

- Retrieve the titles of books written by "Abiteboul" published before 2000, together with their publisher

  <result>
  
  for $b in document("bib.xml")/bibliography/book  
  let $p := $b/publisher  
  where $b/year < 2000  
  return  
  <book>  
  { $b/title }  
  { $p }  
  </book>  
  </result>
An equivalent formulation

- Retrieve the titles of books written by "Abiteboul" published before 2000, together with their publisher

```xml
<result>
  for $b in document('bib.xml')/bibliography/book[year<2000]
  return
    <book>
      { $b/title }
      { $b/publisher }
    </book>
</result>
```

Yet another formulation

- Retrieve the titles of books written by "Abiteboul" published before 2000, together with their publisher

```xml
<result>
  for $b in document('bib.xml')/bibliography/book,
  $p in $b/publisher
  where $b/year < 2000
  return
    <book>
      { $b/title }
      { $p }
    </book>
</result>
```

Existentially quantified expressions

(some $var in node-set satisfies condition)
- Can be used in where as a condition

- Find titles of books in which XML is mentioned in some section

```xml
<result>
  for $b in document('bib.xml')//book
  where (some $section in $b//section satisfies contains($section, "XML"))
  return ($b/title)
</result>
```
Universally quantified expressions

\((\text{every } var \text{ in } \text{node-set satisfies condition})\)

- Can be used in \textit{where} as a condition
- Find titles of books in which XML is mentioned in every section

\[
\begin{align*}
\text{result} & = \\
\text{for } b \text{ in document("bib.xml")//book} & \text{ where (every } section \text{ in } b//section \text{ satisfies contains($section, \text{ "XML"}$)}) & \text{return } b//title
\end{align*}
\]

Aggregation

- List each publisher and the average prices of all its books

\[
\begin{align*}
\text{result} & = \\
\text{for } pub \text{ in distinct-values(document("bib.xml")//publisher) } & \text{let } price := \text{avg(document("bib.xml")/book[publisher=$pub]/@price)} & \text{return } \langle\text{publisherpricing} \rangle \{ pub \} \langle\text{avgprice} \rangle \{ price \} \rangle \langle/\text{publisherpricing} \rangle
\end{align*}
\]

- \texttt{distinct-values(node-set)} removes duplicates
- Two elements are considered duplicates if their names, attributes, and "normalized contents" are equal (still under active discussion)
- \texttt{avg(node-set)} computes the average of node-set (assuming each node in node-set can be converted to a numeric value)

Ordering and sorting

- An XPath expression always returns a node-set in document order
- \texttt{for} loop will respect the ordering of nodes in a node-set
- Use \texttt{sort by (sort-by-expression-list)} to output results in a user-specified order
- List all books with price higher than $100, in order by first author; for books with the same first author, order by title

\[
\begin{align*}
\text{result} & = \\
\text{document("bib.xml")//book[@price>100} & \text{sort by (author[1], title)} & \text{]}</result>
\]
A tricky sorting example

- List titles of all books, sorted by their prices
  
  \[
  \text{result}\{ 
  \text{document("bib.xml")//book sort by (@price))/title} 
  \}\text{result}\]

- What is wrong?
- Correct versions

Summary

- Many, many more features not covered in class
- XPath is fairly mature and stable
  - Already a W3C recommendation
  - Implemented in many systems
  - Used in many other standards
- XQuery is still evolving
  - Still a W3C working draft
  - Some vendors are coming out with implementations
  - To become the SQL for XML?
  - Wait… Where did the join go?