XML Document Example

Root element
<book price="55">
  <publisher>Addison-Wesley</publisher>
  <author>Serge Abiteboul</author>
  <author><first-name>Rick</first-name><last-name>Hull</last-name></author>
  <author><first-name>Victor</first-name><last-name>Vianu</last-name></author>
  <title>Principles of Database and Knowledge Base Systems</title>
  <year>1995</year>
</book>

Text element
<this is a comment>
</text>

XPath Query Example

• /bib/book/year
Result:
<year>1995</year>
<year>1998</year>

• /bib/paper/year
Result:
Empty
(No elements named paper)

Outline

• XML Document
  – Tree structure
• Xpath Queries
  – Selecting descendants and attributes
  – Functions
  – Shortcuts
• Examples and exam questions

XPath as a Tree

Root Node
<book price="55"
  <publisher>Addison-Wesley</publisher>
  <author>Serge Abiteboul</author>
  <author><first-name>Rick</first-name><last-name>Hull</last-name></author>
  <author><first-name>Victor</first-name><last-name>Vianu</last-name></author>
  <title>Principles of Database and Knowledge Base Systems</title>
  <year>1995</year>
</book>

Descendent or self

• /author
Result:
<author>Serge Abiteboul</author>
<author><first-name>Rick</first-name><last-name>Hull</last-name></author>
<author><first-name>Victor</first-name><last-name>Vianu</last-name></author>
<author>Jeffrey D. Ullman</author>

• /bib/first-name
Result:
<first-name>Rick</first-name>
Selecting non-element nodes

- text() – Returns text nodes
- node() – Returns any node
- comment() – Returns comment nodes

- What /bib/book/author/text() will return?
- Author names in one text node:
  - Serge Abiteboul
  - Victor Vianu
  - Jeffrey D. Ullman

- Rick Hull does not appear because there is no text node under the author element
- //comment() – Returns comment nodes
- @* can be used to select any attribute

Selecting Unknown Nodes

- //author/*
  - Result:
    - <first-name>Rick</first-name><last-name>Hull</last-name>

  - The * selects every node element
    - Excluding text nodes

Accessing attributes

- /bib/book/@price
- Result: “55”

- @* can be used to select any attribute

Operators

- >>>b returns true
  - if node a appears after node b in the document.
- <<< will check the opposite

- For example:
  - //last-name >>>//first-name true
  - //first-name >>> //last-name false

Selecting using Boolean Expressions

- ./bib/book/author[first-name = “Rick”]
  - Result:
    - <author>
      <first-name>Rick</first-name><last-name>Hull</last-name></author>
Selecting using Boolean Expressions

- \(/\text{book[@price<60][publisher = "Addison-Wesley"]}\)
  - the semantic meaning of \([\ldots]\) –
  - First filter by the first expression and then
  - filter by the second.

In this case it is equivalent to:
- \(/\text{book[@price<60 and publisher = "Addison-Wesley"]}\)
  - Not always equivalent!

Existential Boolean Expressions

- \(/\text{bib/book[@first-name][address[zip][city]]/last-name}\)

Functions

- The query \(/\text{bib/book[2]}\)
  - Is equivalent to \(/\text{bib/book[position() = 2]}\)
- position() returns the position of the node with respect to its context node

- \(/\text{author[last()]}\)
  - last() return the position of the last node in the set of nodes
  - with respect to the context node.

Functions- Equivalence(?) Example

Are the following queries equivalent?

- \(/\text{author[1][2]}\)
- \(/\text{author[2][1]}\)

Functions- Equivalence(?) Example

- \(/\text{bib/book/author[first-name][address[zip][city]]/last-name}\)

Result:
- Last names of authors having
  - first name and
  - address containing zip and city.
- [first-name] is a Boolean condition

- Is it possible to return authors having first
  name only (and not last name)? Yes.
  - \(/\text{bib/book/author[first-name][not(last-name)]}\)
**Functions**

- `/bib/*[name()="book"]`
  - Equivalent to `/bib/book`
  - name() return the name of the node
- `/bib/book[count(author)>1]`
  - Return books having more than 1 author.
- `id("a0130353000")`
  - id() return nodes having the given id as input

**Axes**

<table>
<thead>
<tr>
<th>AxisName</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ancestor</td>
<td>Selects all ancestors (parent, grandparent, etc.) of the current node</td>
</tr>
<tr>
<td>ancestor-or-self</td>
<td>Selects all ancestors (parent, grandparent, etc.) of the current node and the current node itself</td>
</tr>
<tr>
<td>attribute</td>
<td>Selects all attributes of the current node</td>
</tr>
<tr>
<td>child</td>
<td>Selects all children of the current node</td>
</tr>
<tr>
<td>descendant</td>
<td>Selects all descendants (children, grandchildren, etc.) of the current node</td>
</tr>
<tr>
<td>descendant-or-self</td>
<td>Selects all descendants (children, grandchildren, etc.) of the current node and the current node itself</td>
</tr>
</tbody>
</table>

**Axes - Examples**

- `/child:/bib/child:book`
- `/child:/bib/child:book/attribute:price`
- `@self::node()/descendant-or-self::node()/child::title`
- `//author/[number = position() = number]`

**Shortcuts**

<table>
<thead>
<tr>
<th>Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>child::</td>
</tr>
<tr>
<td>attribute::</td>
</tr>
<tr>
<td>self::node()</td>
</tr>
<tr>
<td>parent::node()</td>
</tr>
<tr>
<td>./descendant-or-self::node/</td>
</tr>
</tbody>
</table>

**Equivalence Question**

Are the following queries equivalent?

- `//author[1]`
- `//author/.[1]`

- No! `//author/.[1]` returns all authors in the document because the `/` make every author node to be its own context node.
Examples

//author/descendant-or-self::node()/child::zip
⇔ //author//zip
⇔ //author/descendant::zip

What will be returned in the following?
• //book/publisher/parent::*//author
• //book[author[position() lt last()]]
• //book[count(author[1]) = count(author[1] | author[last()]) ]

Exam Question

<ELEMENT tournament (game*)>
<ELEMENT game (GID, date, player, player, move*)>
<ELEMENT player (name, country, color)>
<ELEMENT move(no, status, piece, from, to)>
<ELEMENT piece(type, color)>

• To simplify the display of the DTD, assume that the elements GID, date, name, country, color, no, status, from, to, type are PCDATA.
• Write an XPath query that finds the name of the player who moved a piece during move number 7 of game number 99. The query must find the player by comparing the color in which he plays to the color of the tool that was moved during the course of the discussion.

Exam Question - Solution

• Game 99:
  • //game[GID=99]
  • The color of move 7 in game 99:
    • //game[GID=99]/move[no=7]/piece/color
  • The name of the player who moved at move 7 in game 99:
    • //game[GID=99]/player[color = ../move[no=7]/piece/color]/name