XML Document Example

```
<book price="55">
  <publisher>Addison-Wesley</publisher>
  <author>Serge Abiteboul</author>
  <first-name>Rick</first-name>
  <last-name>Hull</last-name>
  <author>Victor Vianu</author>
  <title>Foundations of Databases</title>
  <year>1995</year>
</book>
```

XML Document as a Tree

Root Element

Descendent or self

XPath Query Example

```
//bib/book/year
```

Result:

```
<year>1995</year>
<year>1998</year>
```

```
//bib/paper/year
```

Result:

Empty
(No elements named paper)
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()`
- `<!-- this is a comment -->`

Selecting Unknown Nodes

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

  * selects every node element
  - Excluding text nodes

Accessing attributes

- `/bib/book/@price`  
  Result:  
  `"55"

  `@*` can be used to select any attribute

Operators

- General comparison operators:  
  `=, !=, >, <, >=, <=`
  In case of comparing a sequence with a single value returns true if there is one value matching to required comparison.

- Order comparison:  
  `>>, <<, is`
  Comparing the relative order of elements in the document.
  Not Defined for comparing sequences.

Selecting using Boolean Expressions

- `/bib/book/author[first-name = "Rick"]`
  Result:
  ```xml
  <first-name>Rick</first-name>
  <last-name>Hull</last-name>
  </author>
  ```
Selecting using Boolean Expressions

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

In this case it is equivalent to :

- //book[@price<60 and publisher = "Addison-Wesley"]
  - Not always equivalent!!
  - as we will see later with different Boolean expressions

Existential Boolean Expressions

- /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.
    - /bib/book/author[first-name][not(last-name)]

Functions

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

- //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?

- //author[1][2]
- //author[2][1]
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

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/child::bib/child::book</td>
<td>selects all <code>book</code> elements under <code>bib</code></td>
</tr>
<tr>
<td>/child::bib/child::book/attribute::price</td>
<td>selects the <code>price</code> attribute of <code>book</code> elements under <code>bib</code></td>
</tr>
<tr>
<td>/self::node()/descendant-or-self::node()/child::title</td>
<td>selects all <code>title</code> elements under <code>node</code> and <code>child</code> nodes</td>
</tr>
<tr>
<td>/descendant-or-self::node()/child::author/parent::node()</td>
<td>selects all <code>author</code> elements under <code>node</code> and <code>child</code> nodes, where <code>node</code> is also an ancestor of <code>child</code></td>
</tr>
</tbody>
</table>

Shortcuts

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>child::</td>
<td>Selects the current node</td>
</tr>
<tr>
<td>attribute::</td>
<td>Selects the current node</td>
</tr>
<tr>
<td>self::node()</td>
<td>Selects the current node</td>
</tr>
<tr>
<td>parent::node()</td>
<td>Selects the current node</td>
</tr>
<tr>
<td>/descendant-or-self::node/</td>
<td>selects all <code>node</code> elements under <code>node</code></td>
</tr>
<tr>
<td>[position() = number]</td>
<td>selects <code>node</code> elements with position <code>number</code></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()<last()]]
- //book[count(author[1]) = count(author[1] | author[last()]) ]

Examples

- For which books the year of publication is the most recent?
  - //book[ not(year < //book/year) ]

Exam Question

- 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

Exam Question - Solution

- 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.