XPath – an XML query language

Some XML query languages:

- XML-QL
- XPath
- XQuery
- Many others

XML-QL

- http://www.w3.org/TR/NOTE-xml-ql (8/98)
- Features:
  - regular path expressions
  - patterns, templates
  - Skolem Functions
- Based on OEM data model

XML-QL (cont…)

- E.g. Retrieve all authors of books published by Morgan Kaufmann.

where
<book>
<publisher> <name> Morgan Kaufmann </name> </publisher>
<title> $T </title>
<author> $A </author>
</book> in "www.a.b.c/bib.xml"
construct $A

- Here **$T** and **$A** are variables.

XML-QL (cont.)

Constructing New XML Data

- E.g. Retrieve all authors and titles of books published by Morgan Kaufmann.

```
where <book>
          <publisher> <name> Morgan Kaufmann </> </>
          <title>  $T </>
          <author> $A </>
      </> in “www.a.b.c/bib.xml”
construct <result>
              <author> $A </>
              <title>  $T </>
           </>
```

- </> as the closing tag for the nearest unclosed tag.

XML-QL (cont.)

Consider the following simple XML data...

```
<book year="1991">
   <!-- A good introductory text -->
   <title> Advanced Compiler Design and Implementation
   </title>
   <author> <lastname> Muchnick </lastname> </author>
   <publisher> <name> Morgan Kaufmann </name> </publisher>
</book>

<book year="1995">
   <title> Active Database Systems </title>
   <author> <lastname> Ceri </lastname> </author>
   <author> <lastname> Widom </lastname> </author>
   <publisher> <name> Morgan Kaufmann </name> </publisher>
</book>
```

XML-QL (cont.)

The result of the query is as below...

```
<result>
   <author> <lastname> Muchnick </lastname> </author>
   <title> Advanced Compiler Design and Implementation
   </title>
</result>

<result>
   <author> <lastname> Ceri </lastname> </author>
   <title> Active Database Systems </title>
</result>

<result>
   <author> <lastname> Widom </lastname> </author>
   <title> Active Database Systems </title>
</result>
```

XPath

- http://www.w3.org/TR/xpath (11/99)
- Building block for other W3C standards:
  - XSL Transformations (XSLT)
  - XML Link (XLink)
  - XML Pointer (XPointer)
  - XML Query
  - XQuery
- Was originally part of XSL - EXtensible Stylesheet Language

XPath: An example of XPath Queries

A sample XML document with 2 books:

```
<bib>
   <book> <publisher> Addison-Wesley </publisher>
          <author> Serge Abiteboul </author>
          <author> <first-name> Rick </first-name>
                   <last-name> Hull </last-name>
          </author>
          <author> Victor Vianu </author>
          <title> Foundations of Databases </title>
          <year> 1995 </year>
   </book>
   <book price="55">
          <publisher> Freeman </publisher>
          <author> Jeffrey D. Ullman </author>
          <title> Principles of Database and Knowledge
                     Base Systems </title>
          <year> 1998 </year>
   </book>
</bib>
```

XPath: Simple Expressions

/bib/book/year

Result: **<year> 1995 </year>**
<year> 1998 </year>

Note: "/" is the child axis

/bib/paper/year

Result: empty (there were no papers)

XPath: Restricted Kleene Closure

```
//author
```

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

Note: "//" is the descendant axis

```
/bib//first-name
```

Result:
```
<first-name> Rick </first-name>
```

XPath: Text Nodes

/bib/book/author/text()

Result: **Serge Abiteboul**
Victor Vianu
Jeffrey D. Ullman

Note: text() – only matches text value

Note: Rick Hull doesn't appear as a result because he has firstname and lastname tags in the Author element.

XPath: Wildcard

//author/*

Result: **<first-name> Rick </first-name>**

<last-name> Hull </last-name>

Note: ***** *matches any element, the wildcard.*

Note: **Serge Abiteboul, Victor Vianu, and Jeffrey D. Ullman do not appear as results.**

XPath: Attribute Nodes

/bib/book/@price

Result: **“55”**

*Note: **@price** means that price has to be an attribute*

XPath: Qualifiers

/bib/book/author[first-name]

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

Note: Only matches author with a **first-name** tag as a sub-element. Output are author elements, **not** first-name.

XPath: More Qualifiers

```
/bib/book/author[first-name]
     [address[//zip][city]]/last-name
```

Result: `<last-name> . . . </last-name>`
`<last-name> . . . </last-name>`

```
/bib/book[@price < “60”]
```

```
/bib/book[author/@age < “25”]
```

```
/bib/book[author/text()]
```

```
//Part/*/*/subpart/../name
```

is the same as

```
//part/*/*[subpart]/name
```

where `..` means match the parent of the current element.

XPath: Summary

bib	Matches a bib element
*	Matches any element
/	Matches the root element
.	Match the current element
..	Match the parent of current element
/bib	Matches a bib element under root
bib/paper	Matches a paper in bib
bib//paper	Matches a paper in bib, at any depth
//paper	Matches a paper at any depth
@price	Matches a price attribute

XPath: Summary (cont.)

Paper | book Matches a paper or a book

bib/book/@price Matches price attribute in book, in bib

bib/book[@price<“55”]/author/last-name

Matches last name of book with price < 55

/bib/book/author[first-name]

Matches author with a first name tag

/bib/book/author[2]

Matches the second author of a book

/bib/book/author[last()]

Matches the last author of a book

XPath: More Details

- There are 7 **kinds of nodes**: element, attribute, text, namespace, processing-instruction, comments, and document (root) nodes.
- XPath includes over 100 **built-in-functions**, for string values, numeric values, date, time comparison, Boolean values, etc.
- XPath **axes**: An axis defines a node-set relative to the current node, such as:

  attribute, self, parent, ancestor, ancestor-or-self,

  child, descendant, descendant-or-self,

  following, following-sibling, preceding, preceding-sibling.

  (see next page for the meanings).

- XPath operators: Operators can be used in XPath expressions, such as:

  | + - * div = != < <= > >= or and mod

Axes of XPath

- **child** the children of the context node
- **descendant** all descendants (children, children's children, ...)
- **parent** the parent (empty if at the root)
- **ancestor** all ancestors from the root to the parent
- **following-sibling** siblings to the right
- **preceding-sibling** siblings to the left
- **following** all following nodes in the document
- **preceding** all preceding nodes in the document
- **attribute** the attributes of the context node
- **self** the context node itself
- **descendant-or-self** the union of descendant and self
- **ancestor-or-self** the union of ancestor and self

XPath: More Details (cont…)

- Example:
  - **child::author/child::lastname = author/lastname**
  - **child::author/descendant::zip = author//zip**
  - **child::author/parent::* = author/..**
  - **child::author/attribute::age = author/@age**

More on XPath functions

Core Function Library: Node-Set Functions

- number last()

  returns the context size from the expression evaluation context.

- number position()

  returns the context position from the expression evaluation context.

- number count(node-set)

  returns the number of nodes in the argument node-set.

- node-set id(object)

  selects elements by their unique ID, as declared in DTD.

More on XPath functions

Core Function Library: String Functions

- string string(object?) This converts an object to a string. If the argument is omitted, it defaults to the context node. ? indicates the argument is optional.
- string concat(string, string, string*) * indicates 0 to any of string as arguments.
- boolean start-with(string, string)
- boolean contains(string, string) Returns true if first string contains second string, otherwise it returns false
- string substring-before(string, string)
- string substring-after(string, string)
- string substring(string, from_position, to_position?) If the third argument is not specified, it returns the substring starting at the from_position to the end of the string. ? indicates the argument is optional.
- number string-length(string?) If the argument is omitted, it defaults to the context node converted to a string.
- string normalize-space(string?) strips leading and trailing whitespace and replacing sequences of whitespace characters by a single space.
- string translate(string, string, string)

An XML document

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<book>
 <chapter>
  <title>Various Fruits</title>
  <para> The next chapters introduce different kinds of fruits, like
    <fruit figref="fr_virg">strawberries</fruit> or <fruit figref="apple">apples</fruit>.
  </para>
 </chapter>
 <chapter>
  <title>Strawberries</title>
  <para> stre[a]w berige; stre[a]w straw + berie berry;
     perhaps from the resemblance of the runners of the plant to straws.
  </para>
  <para> A fragrant edible berry, of a delicious taste and commonly of a red colour.
  </para>
  <para> The common American strawberry is
    <figure caption="Fragaria virginiana" data="fr_virg.jpg" id="fr_virg">Fragaria virginiana</figure>,
      the European is
    <figure caption="Fragaria vesca" data="fr_vesca.jpg" id="fr_vesca">Fragaria vesca</figure>.
  </para>
 </chapter>
</book>
```

XPath query examples

- Select the figure elements without attributes:

  ```
  //figure[not(@*)]
  ```

- Select the chapters having three paragraphs:

  ```
  //chapter[count(.//para) = 3]
  ```

- Select the first paragraph of each chapter:

  ```
  //chapter//para[1]
  ```

- Select the first paragraph of all chapters:

  ```
  (//chapter//para)[1]
  ```

- Select the figures with an attribute caption 'Fragaria virginiana' from the second chapter:

  ```
  //chapter[2]//figure[@caption = 'Fragaria virginiana']
  ```

- Select the figures in chapters 2 through 5:

  ```
  //chapter[position() >= 2 and position() <= 5]//figure
  ```

- Select captions of figures that are referenced by figref attributes of fruit elements in the first chapter:

  ```
  id(//chapter[1]//fruit/@figref)[self::figure]/@caption
  ```

More XPath query examples

- Select chapters in which the word 'Strawberry' is mentioned in at least one paragraph:

```
//chapter[.//para[contains(.,'Strawberry')]]
```

- Select chapters in which the word 'Strawberry' is mentioned in every paragraph:

```
//chapter[count(.//para) =
          count(.//para[contains(.,'Strawberry')])]
OR
//chapter[not(.//para[not(contains(.,'Strawberry'))])]
```

- List the names of the second-level managers of all employees whose rating is 'Good':

```
id(id(/emp[rating =
        "Good"]/@mgr)[self::emp]/@mgr)[self::emp]/name
```