Query Languages for XML

- XPath
- XSLT
- XQuery

Common Querying Tasks

- Filter, select XML values
  - Navigation, selection, extraction
- Merge, integrate values from multiple XML sources
  - Joins, aggregation
- Transform XML values from one schema to another
  - XML construction
Query Languages

- XPath
  - Common language for navigation, selection, extraction
  - Used in XSLT, XQuery, XML Schema, ...
- XSLT: XML ⇒ XML, HTML, Text
  - Loosely-typed scripting language
  - Format XML in HTML for display in browser
  - Highly tolerant of variability/errors in data
- XQuery 1.0: XML ⇒ XML
  - Strongly-typed query language
  - Large-scale database access
  - Safety/correctness of operations on data

XML data: Running example

XML input: www.a.b/bib.xml

```
<book year="1996">
  <title>HTML</title>
  <author><last>Lee</last> <first>T.</first></author>
  <author><last>Smith</last> <first>C.</first></author>
  <publisher>Addison-Wesley</publisher>
  <price>59.99</price>
</book>

<book year="2003">
  <title>WMD</title>
  <author><last>Bush</last> <first>G.</first></author>
  <publisher>white house</publisher>
</book>
```
**DTD**

```xml
<!ELEMENT bib (book*) >
<!ELEMENT book (title, (author+ | editor+),
  publisher?, price?) >
<!ATTLIST book year CDATA #required >
<!ELEMENT author (last, first)> 
<!ELEMENT editor (last, first, affiliation)> 
<!ELEMENT publisher (#PCDATA) > 
```

**Data model**

Node-labeled, ordered tree

```
<table>
<thead>
<tr>
<th>bib</th>
</tr>
</thead>
<tbody>
<tr>
<td>book</td>
</tr>
<tr>
<td>title author author publisher phone @year</td>
</tr>
<tr>
<td>@year title author publisher</td>
</tr>
<tr>
<td>last first last first</td>
</tr>
<tr>
<td>last first</td>
</tr>
</tbody>
</table>
```
XPath

W3C standard: www.w3.org/TR/xpath

- Navigating an XML tree and finding parts of the tree (node selection and value extraction)

Given an XML tree T and a context node n, an XPath query Q returns
- the set of nodes reachable via Q from the node n in T – if Q is a unary query
- truth value indicating whether Q is true at n in T – if Q is a Boolean query.

- Implementations: XALAN, SAXON, Berkeley DB XML – freeware, which you can play with
- A major element of XSLT, XQuery and XML Schema
- XPath 2.0 (Turing-Complete)

XPath constructs

XPath query Q:
- Tree traversal: downward, upward, sideways
- Relational/Boolean expressions: qualifiers (predicates)
- Functions: aggregation (e.g., count), string functions

//author[last="Bush"]
Downward traversal

Syntax:

\[ Q ::= \ . | l | @l | Q/Q | Q | //Q | /Q | Q[q] \]

\[ q ::= Q | Q \ op \ c | q \ and \ q | q \ or \ q | not(q) \]

- \( . \): self, the current node
- \( l \): either a tag (label) or \(*\): wildcard that matches any label
- \( @l \): attribute
- \( /, | \): concatenation (child), union
- \( // \): descendants or self, “recursion”
- \( [q] \): qualifier (filter, predicate)
  - \( \ op =, !=, <=, <, >, >=, > \)
  - \( c \): constant
  - \( and, or, not() \): conjunction, disjunction, negation

Existential semantics: \(/bib/book[last=“Bush”]\)

Examples:

- parent/child: \(/bib/book\)
- ancestor//descendant: \(/bib/last, //last\)
- wild card: \(/bib/book/\cdot\)
- attributes: \(/bib/book/@year\)
- attributes with wild cards: \(///book/@*\)
- union: \(/book/(editor \mid author)\)

Are \(/book/(editor \mid author)\) and \(/(editor \mid author)\) “equivalent” at context nodes (1) root, (2) book, (3) author?
Filters (qualifiers)

✓ //book[price]/title -- titles of books with a price
✓ //book<title and author and not(price)}/title
titles of books with authors, title but no price
✓ //book[author/last = "Bush"]/title
titles of books with an author whose last name is Bush
✓ //book[editor | author]/title
titles of books with with either an author or an editor
What is /[[@id]]? /[[not([@id])]? /[[not(//[not([@id])])]?  

Upward traversal

Syntax:
Q ::= . . . | ./Q | ancestor ::Q | ancestor-or-self::Q
✓ ./: parent
✓ ancestor, ancestor-or-self: recursion
Example:
✓ //author[../title = "WMD"]/last
    find the last names of authors of books with the title "WMD"
✓ ancestor :: book[/last="Bush"]
    find book ancestors with “Bush” as its last descendant
Are the following equivalent to each other (context node: a book)?
    ./book/author, .author
Sideways

Syntax:

Q ::= ... | following-sibling ::=Q | preceding-sibling::Q

- following-sibling: the next sibling
- preceding-sibling: the previous sibling
- position function: e.g., //author[position( ) < 2]

Example:

- following-sibling :: book //@last="Bush"]
  find the next book written by Bush
- preceding-sibling :: book//@last="Bush"]
  find the last book written by Bush

Query Languages for XML

- XPath
- XSLT
- XQuery
XSL (eXtensible Stylesheet Language)

W3C recommendation [www.w3.org/Style/XSL](http://www.w3.org/Style/XSL)

- Two separate languages:
  - XSLT: transformation language, Turing complete
  - a formatting language
- Purpose: stylesheet specification language
  - displaying XML documents: XML -> HTML
  - transforming/querying XML data: XML -> XML
- Implementations: SAXON, XALAN, …

See [http://www.oasis-open.org/cover/xsl.html](http://www.oasis-open.org/cover/xsl.html) for a number of implementations and demos.

XSL programs

XSL program: a collection of template rules

- template rule = pattern + template
- computation:
  - starts from the root
  - apply a pattern to each node. If it matches, execute the corresponding template (to construct XML/HTML), and apply templates recursively on its children.
- patterns:
  - match pattern: determine content – whether or not to apply the rule?
  - select pattern: identify nodes to be processed, set of nodes
An example XSLT program

Q1: Find titles and authors of all books published by Addison-Wesley after 1991.

```xml
<xsl:template match="/bib/book[@year > 1991 and publisher='Addison-Wesley']">
    <result>
        <title><xsl:value-of select="title" /></title>
        <xsl:for-each select="author"/>
        <author><xsl:value-of /></author>
    </xsl:for-each>
</result>
</xsl:template>
```

Basic XSLT constructs

- A collection of templates: `<xsl:template>`
- Match pattern: `match="/bib/book[@year > 1991 and publisher='Addison-Wesley']"`
- Select pattern: `select="title", xsl:for-each="author"`
- Value-of: string
- Constructing XML data:

```xml
<result>
    <title><xsl:value-of select="title" /></title>
    ...
</result>
```
**Patterns**

- match pattern: (downward) XPath
  - parent/child: bib/book
  - ancestor//descendant (_*): bib//last, //last, …
- select patterns: XPath

Example:

```xml
<xsl:template match="/bib/book/title">
  <result>
    <title> <xsl:value-of /> </title>
    <author> <xsl:value-of select="../author" /> </author>
  </result>
</xsl:template>
```

Note: first author only (without xsl:for-each)

---

**Apply templates**

Recursive processing:

```xml
<xsl:template match="XPath">
  . . .
  <xsl:apply-templates select="XPath"/>
  . . .
</xsl:template>
```

- Compare each selected child (descendant) of the matched source element against the templates in your program
- If a match is found, output the template for the matched node
- One can use `xsl:apply-templates` instead of `xsl:for-each`
- If the select attribute is missing, all the children are selected
- When the match attribute is missing, the template matches every node:
Rewriting Q1 with apply-templates

Selection and construction:

Q1: Find the titles and authors of all books published by Addison-Wesley after 1991.

```xml
<xsl:template match="/bib/book[@year > 1991 and publisher='Addison-Wesley']">
  <result>
    <title> <xsl:value-of select="title" /> </title>
    <xsl:apply-templates />
  </result>
</xsl:template>
```

```xml
<xsl:template match="author">
  <author><xsl:value-of select="." /></author>
</xsl:template>
```

Flow control in XSL

```xml
<xsl:template>
  <xsl:apply-templates />
</xsl:template>
```

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

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

```xml
<xsl:template match="c">
  <C> <xsl:value-of /> </C>
</xsl:template>
```
transformation

\[
\begin{align*}
\langle a \rangle \langle e \rangle \langle b \rangle \langle c \rangle & 1 \langle /c \rangle \\
& \langle c \rangle 2 \langle /c \rangle \\
& \langle /b \rangle \\
& \langle /e \rangle \\
& \langle c \rangle 4 \langle /c \rangle \\
\langle /a \rangle \\
\Rightarrow \\
\langle A \rangle \langle B \rangle \langle C \rangle & 1 \langle /C \rangle \\
& \langle C \rangle 2 \langle /C \rangle \\
& \langle /B \rangle \\
& \langle C \rangle 4 \langle /C \rangle \\
\langle /A \rangle
\end{align*}
\]

Divergence

XSL program may not terminate.

Add the following to the previous program:

\[
\begin{align*}
\text{<xsl:template match="e" >} \\
\text{ \ \ \ < xsl:apply-templates select="/" />} \\
\text{<\/xsl:template>}
\end{align*}
\]
**XSL default rules**

Implicitly included in all style sheets

Default rule for element tree: it recursively descends the element tree and applies templates to the children of all elements

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

* | /: for any element node and the root node

However, once an explicit rule for the parent of any element is present, this rule will not be activated for the element.

---

**Optional elements**

Q2: Find all book titles, and prices where available

```xml
<xsl:template match="/bib/book[title]">
  <result>
    <title> <xsl:value-of select="title" /> </title>
    <xsl:if test="./[price]">
      <price> <xsl:value-of select="price" /> </price>
    </xsl:if>
  </result>
</xsl:template>
```

- conditional test: `xsl:if`
- `.: current node, XPath`
indexing

Q3: for each book, find its title and its first two authors, and returns <et-al/> if there are more than two authors

```xml
<xsl:template match="/bib/book">
  <result>
    <title> <xsl:value-of select="title" /> </title>
    <xsl:apply-templates select="author" />
  </result>
</xsl:template>

<xsl:template match="author[position() < 2]">
  <author> <xsl:value-of /> </author>
</xsl:template>

<xsl:template match="author[position() = 2]">
  <et-al />
</xsl:template>
```

sorting

Q4: find the titles of all books published by Addison-Wesley after 1991, and list them alphabetically.

```xml
<xsl:template match="/bib/book[@year > 1991 and publisher='Addison-Wesley']">
  <title> <xsl:value-of select="title" /> </title>
  <xsl:apply-templates>
    <xsl:sort select="title" />
  </xsl:apply-templates>
</xsl:template>
```

Key: title

✓ xsl:sort: used together with xsl:for-each or xsl:apply-templates
XML to HTML: display

Q5: generate a HTML document consisting of the titles and authors of all books.

```xml
<xsl:template match="/">
  <html>
    <head> <title> Books </title>   </head>
    <body> <ul> <xsl:apply-templates select="bib/book"></ul></body>
  </html>
</xsl:template>

<xsl:template match="book">
  <li> <b> <xsl:value-of select="title" />,  </b>
    <xsl:for-each select="author"> <em><xsl:value-of /> </em> 
    <xsl:for-each select="author" <em><xsl:value-of /> </em> 
    <xsl:for-each select="author" <br> 
    </li>
</xsl:template>
```

Query Languages for XML

- XPath
- XSLT
- XQuery
XQuery

W3C working draft [www.w3.org/TR/xquery]

Functional, strongly typed query language: Turing-complete
✓ XQuery = XPath + …
  for-let-where-return (FLWR) ~ SQL’s SELECT-FROM-WHERE
  Sort-by
  XML construction (Transformation)
  Operators on types (Compile & run-time type tests)
  + User-defined functions
    Modularize large queries
    Process recursive data
  + Strong typing
    Enforced statically or dynamically
✓ Implementation: GALAX
  [http://www-db.research.bell-labs.com/galax/]

FLWR Expressions

For, Let, Where, OrderBy, return

Q1: Find titles and authors of all books published by Addison-Wesley after 1991.

<answer>{
  for $book in /bib/book
     where $book/@year > 1991 and $book/publisher='Addison-Wesley'
     return <book>
       <title> {$book/title } </title>,
       for $author  in $book/author   return 
         <author> {$author } </author>
     </book>
}</answer>

✓ for loop; $x: variable
✓ where: condition test; selection
✓ return: evaluate an expression and return its value
### join

Find books that cost more at Amazon than at BN

<answer>

```xml
let $amazon := doc("http://www.amazon.com/books.xml"),
    $bn  :=  doc("http://www.bn.com/books.xml")
for    $a  in $amazon/books/book,
        $b  in $bn/books/book
where    $a/isbn = $b/isbn    and     $a/price > $b/price
return   <book> {$a/title, $a/price, $b/price }  <book>
</answer>
```

- ✓ let clause
- ✓ join: of two documents

### Conditional expression

Q2: Find all book titles, and prices where available

<answer>

```xml
for $book  in /bib/book
return   <book>
        <title>  {$book/title } </title>,
        { if  $book[price]
            then <price> {$book/price } </price>
            else ( ) }
    </book>
</answer>
```
Indexing

Q3: for each book, find its title and its first two authors, and returns <et-al/> if there are more than two authors

<answer>{
  for $book in /bib/book
  return <book>
    <title> {$book/title } </title>,
    { for $author in $book/author[position( ) <= 2] return <author> {$author } </author> }
    { if (count($book/author) > 2 then <et-al/> else ( ) }
  </book>
}</answer>

Order by

Q4: find the titles of all books published by Addison-Wesley after 1991, and list them alphabetically.

<answer>{
  for $book in /bib/book
  where $book/@year > 1991 and $book/publisher='Addison-Wesley'
  order by $book/title
  return <book>
    <title> {$book/title } </title>,
    for $author in $book/author return <author> {$author } </author>
  </book>
}</answer>
Grouping

Q5: For each author, find titles of books he/she has written

<answer>
  for $author$ in distinct(/bib/book/author)
    return <author name="{$author}" >{
      for $book$ in /bib/book
        where $book/author = $author
        return <title> {$book/title } </title>
    </author>
}</answer>

✓ Constructing attributes: <author name="{$author}" >
✓ Grouping: for $book$ in /bib/book ...

Recursion

Consider a part DTD

<!--ELEMENT part (part*)-->
<!--ATTLIST part name CDATA #required-->
<!--ATTLIST part cost CDATA #required-->

part – subpart hierarchy

Given a part element, we want to find the total cost of the part – recursive computation that descends the part hierarchy
function

define function total (element part $part)
returns element part {
    let $subparts :=
    for $s in $part/part return total($s)
    return {
        <part name="$part/@name"
            cost="$part/@cost + sum($subparts/@cost)">
        } </part>
    }

✓ recursive function: it recursively descends the hierarchy of $part
✓ $subparts: a list
✓ $part: parameter

Summary and Review

Query languages for XML
✓ XPath: navigating an XML tree
✓ XSLT: XML transformations – can be used as a query language
✓ XQuery: XML query language

Very powerful (as opposed to relational algebra); however, query processing/optimization is hard – open issue!

Homework:
✓ Write queries on the school document you created, using XPath, XSLT and XQuery; display the query answers in HTML
✓ Find some queries in XPath, XSLT and XQuery that are not expressible in SQL, even when relational data is considered (i.e., relational data represented in a canonical form in XML)