Outline

- XML parsers
- XPath
- XQuery
- XML publishing

- Background (reading)
  - http://www.w3.org/TR/xmlquery-use-cases/ several Xquery examples
  - http://www.galaxquery.org/ nice command line tool
XML Parsers - Overview

• What do we do if we need to “read” an XML document?
  – This is typically called parsing
    • Navigate through XML trees
    • Construct XML trees
    • Output XML
  – SOAP libraries use this technique to read and write XML messages

Two XML Parsers

• Two main APIs for XML parsers
  – DOM (Document Object Model)
    • Tree structure
  – SAX (Simple Api for Xml)
    • Read individual tags
• Built for programmers who do not want to write their own parsers
• Available for multiple languages
DOM

- Platform- and language-neutral interface
- Allows to build documents, navigate their structure, and add, modify, or delete elements and content.

DOM Example

![XML Document and Document object tree]

Figure 1: Hierarchical structure of a document object
DOM Cont.

- **Tree** for XML works fine since XML is hierarchically organised
- Every XML document can be represented as a tree

Some Insights in API

```java
org.w3c.dom.Document

NodeList getElementsByTagName(java.lang.String tagname)

  Returns a NodeList of all the Elements with a given tag name in the order in which they are encountered in a preorder traversal of the Document tree.

Element getDocumentElement()

  This is a convenience attribute that allows direct access to the child node that is the root element of the document.
```

http://java.sun.com/webservices/jaxp/dist/1.1/docs/api/org/w3c/dom/package-summary.html
Some Insights in API

org.w3c.dom.Node

NodeList
getChildNodes()

A NodeList that contains all children of this node.

Node
getFirstChild()

The first child of this node.

Node
getLastChild()

The last child of this node.

Node
getParentNode()

The parent of this node.

import java.io.IOException; // Exception handling
import org.w3c.dom.*; // DOM interface
import org.apache.xerces.parsers.DOMParser; // Parser (to DOM)

class Hello {
    public static void main(String[] args) {
        String filename = args[0];
        System.out.print("The document element of " + filename + " is ... ");
        try {
            DOMParser dp = new DOMParser();
            dp.parse(filename);
            Document doc = dp.getDocument();
            Element docElm = doc.getDocumentElement();
            System.out.println(docElm.getNodeName() + ".");
        } catch (Exception e) {
            System.out.println("\nError: " + e.getMessage());
        }
    }
}

http://www.troubleshooters.com/tpromag/200103/codexercises.htm
<?xml version="1.0"?>
<workers>
  <contractor>
    <info lname="albertson" fname="albert" ssno="123456789"/>
    <job>C++ programmer</job>
    <hiredate>1/1/1999</hiredate>
  </contractor>
  <contractor>
    <info lname="bartholemew" fname="bart" ssno="223456789"/>
    <job>Technology Director</job>
    <hiredate>1/1/2000</hiredate>
    <firedate>1/11/2000</firedate>
  </contractor>
  <partner>
    <info lname="carlson" fname="carl" ssno="323456789"/>
    <job>labor law</job>
    <hiredate>10/1/1979</hiredate>
  </partner>
  <contractor>
    <info lname="denby" fname="dennis" ssno="423456789"/>
    <job>cobol programmer</job>
    <hiredate>1/1/1959</hiredate>
  </contractor>
  <employee>
    <info lname="edwards" fname="eddie" ssno="523456789"/>
    <job>project manager</job>
    <hiredate>4/4/1996</hiredate>
  </employee>
  <partner>
    <info lname="fredericks" fname="fred" ssno="623456789"/>
    <job>intellectual property law</job>
    <hiredate>10/1/1991</hiredate>
  </partner>
</workers>

class ContractorLastNamePrinter {
  ContractorLastNamePrinter(Document doc) {
    System.out.println();
    try {
      //*** GET DOCUMENT ELEMENT BY NAME ***
      NodeList nodelist = doc.getElementsByTagName("workers");
      Element elm = (Element) nodelist.item(0);

      //*** GET ALL contractors BELOW workers ***
      NodeList contractors = elm.getElementsByTagName("contractor");
      for(int i = 0; i < contractors.getLength(); i++) {
        Element contractor = (Element) contractors.item(i);

        //*** NO NEED TO ITERATE info ELEMENTS, ***
        //*** WE KNOW THERE'S ONLY ONE ***
        Element info =
            (Element)contractor.getElementsByTagName("info").item(0);
        System.out.println("Contractor last name is " + info.getAttribute("lname"));
      }
    } catch (Exception e) {
      System.out.println("ContractorLastNamePrinter() error: " + e.getMessage());
    }
  }
}
SAX

• Access to XML information as a sequence of events
  – Document is scanned from start to end
• Faster than DOM
• You can create your own object model
• You are responsible to interpret all the objects read by the parser

SAX Events

• the start of the document is encountered
• the end of the document is encountered
• the start tag of an element is encountered
• the end tag of an element is encountered
• character data is encountered
• a processing instruction is encountered
<purchase-order>
  <date>2005-10-31</date>
  <number>12345</number>
  <purchased-by>
    <name>My name</name>
    <address>My address</address>
  </purchased-by>
  <order-items>
    <item>
      <code>687</code>
      <type>CD</type>
      <label>Some music</label>
    </item>
    <item>
      <code>129851</code>
      <type>DVD</type>
      <label>Some video</label>
    </item>
  </order-items>
</purchase-order>

import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

private static final class SaxHandler extends DefaultHandler {
    // invoked when document-parsing is started.
    public void startDocument() throws SAXException {
        System.out.println("Document processing started");
    }
    // notifies about finish of parsing:
    public void endDocument() throws SAXException {
        System.out.println("Document processing finished");
    }
    // we enter to element 'qName':
    public void startElement(String uri, String localName,
                              String qName, Attributes attrs) throws SAXException {
        if (qName.equals("purchase-order")) {
        } else if (qName.equals("date")) {
            /* if (...) */
        } else {
            throw new IllegalArgumentException("Element "+
                qName + " is not allowed here");
        }
    }
    // we leave element 'qName' without any actions:
    public void endElement(String uri, String localName, String qName)
        throws SAXException {
        // do nothing;
    }
}
Outline

- XML parsers
- XPath
- XQuery
- XML publishing

Querying XML Data

- XPath = simple navigation through XML tree
- XQuery = the SQL of XML
- XSLT = recursive traversal
  - eXtensible Stylesheet Language Transformation
  - will not discuss

- XQuery and XSLT build on XPath
Sample Data for Queries

```xml
<bib>
  <book>
    <publisher> Addison-Wesley </publisher>
    <author> Serge Abiteboul </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>
```

Data Model for XPath

The root element is `bib`. The root is `The root element`. The XPath data model is shown in the diagram.
XPath: Simple Expressions

/bib/book/year

Result:  
\(<\text{year}> 1995 </\text{year}>\)  
\(<\text{year}> 1998 </\text{year}>\)

/bib/paper/year

Result:  empty  (there were no papers)

XPath: Restricted Kleene Closure

//author

Result:  
\(<\text{author}> \text{Serge Abiteboul} </\text{author}>\)  
\(<\text{author}> \text{Rick} </\text{first-name}> \text{Hull} </\text{last-name}>\)

/bib//first-name

Result:  \(<\text{first-name}> \text{Rick} </\text{first-name}>\)
XPath: Text Nodes

\[ /\text{bib/book/author/text()} \]

Result: 
Serge Abiteboul
Jeffrey D. Ullman

Rick Hull doesn’t appear because he has firstname, lastname

Functions in XPath:
- `text()` = matches the text value
- `node()` = matches any node (= * or @* or `text()`)
- `name()` = returns the name of the current tag

XPath: Wildcard

\[ //\text{author/*} \]

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

* Matches any element
XPath: Attribute Nodes

/bib/book/@price

Result: “55”

@price means that price is has to be an attribute

XPath: Predicates

/bib/book/author[first-name]

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

Predicate corresponds to an IF/THEN statement. If it is true, the Element will be selected!

General: parent[child someTestHere]
**XPath: More Predicates**

/bib/book/author[firstname][address[.//zip][city]]/lastname

Result: <lastname> … </lastname>
<lastname> … </lastname>

**XPath: More Predicates**

/bib/book[@price < “60”]

/bib/book[author/@age < “25”]

/bib/book[author/text()]
XPath: Summary

bib matches a bib element
* matches any element
/ matches the root 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
paper/book matches a paper or a book
@price matches a price attribute
bib/book/@price matches price attribute in book, in bib
bib/book[@price<“55”]/author/lastname matches…

Outline

• XML parsers
• XPath
• XQuery
• XML publishing
XQuery Motivation

• Query is a strongly typed query language
• Builds on XPath
• XPath expressivity insufficient
  – no join queries (as in SQL)
  – no changes to the XML structure possible
  – no quantifiers (as in SQL)
  – no aggregation and functions

FLWR (“Flower”) Expressions

• XQuery uses XPath to express more complex queries

```xml
for ...
let...
where...
return...
```
Sample Data for Queries

```xml
<bib>
  <book>
    <publisher> Addison-Wesley </publisher>
    <author> Serge Abiteboul </author>
    <author> Rick </author>
    <author> Hull </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>
```

Basic FLWR

Find all book titles published after 1995:

```xml
<bib> { 
  for $x in doc("bib.xml")/bib/book
  where $x/year/text() > 1995 
  return $x/title 
} </bib>
```

Result:

```xml
<bib><title> Principles of Database and Knowledge Base Systems </title></bib>
```
FLWR vs. XPath expressions

Equivalently

```xml
for $x in doc("bib.xml")/bib/book[year/text() > 1995]/title
return $x
```

And even shorter:

```xml
doc("bib.xml")/bib/book[year/text() > 1995]/title
```

Result Structuring

- Find all book titles and the year when they were published:

```xml
for $x in doc("bib.xml")/bib/book
return <answer>
   <title>{ $x/title/text() } </title>
   <year>{ $x/year/text() } </year>
</answer>
```

Braces {  } denote evaluation of enclosed expression
Result Structuring

- Notice the use of “{“ and “}”
- What is the result without them?

```xml
for $x in doc("bib.xml")/bib/book
return <answer>
  <title> $x/title/text() </title>
  <year> $x/year/text() </year>
</answer>
```

XQuery Joins and Nesting

For each author of a book by Addison-Wesley, list all books she published:

```xml
for $b in doc("bib.xml")/bib,
  $a in $b/book[publisher/text()="Addison-Wesley"]/author
return <result>
  { $a,
    for $t in $b/book[author/text()=$a/text()]/title
    return $t
  }
</result>
```

In the `return` clause comma concatenates XML fragments
XQuery Nesting

Result:

```xml
<result>
  <author>Jones</author>
  <title> abc </title>
  <title> def </title>
</result>
<result>
  <author>Smith</author>
  <title> ghi </title>
</result>
```

Aggregates

Find all books with more than 3 authors:

```xml
for $x in doc("bib.xml")/bib/book
where count($x/author)>3
return $x
```

- **count** = a function that counts
- **avg** = computes the average
- **sum** = computes the sum
- **distinct-values** = eliminates duplicates
Aggregates

Same thing:

```
for $x in doc("bib.xml")/bib/book[count(author)>3]
return $x
```

Aggregates

Print all authors who published more than 3 books – be aware of duplicates!

```
for $b in doc("bib.xml")/bib,
   $a in distinct-values($b/book/author/text())
where count($b/book[author/text()=$a])>3
return <author> { $a } </author>
```
Aggregates

Find books whose price is larger than average:

```xml
for $b in doc("bib.xml")/bib
let $a:=avg($b/book/price/text())
for $x in $b/book
where $x/price/text() > $a
return $x
```

Result Structure

“Flatten” the authors, i.e. return a list of (author, title) pairs

```xml
for $b in doc("bib.xml")/bib/book,
$x in $b/title/text(),
$y in $b/author/text()
return <answer>
    <title> { $x } </title>
    <author> { $y } </author>
</answer>
```

Result:

```xml
<answer>
    <title> abc </title>
    <author> efg </author>
</answer>
<answer>
    <title> abc </title>
    <author> hkj </author>
</answer>
```
Result Structure

For each author, return all titles of her/his books

```
for $b in doc("bib.xml")/bib, $x in $b/book/author/text()
return
<answer>
  <author> $x </author>
  { for $y in $b/book[author/text()=$x]/title
    return $y }
</answer>
```

Result:
```
<answer>
  <author> efg </author>
  <title> abc </title>
  <title> klm </title>
  . . . .
</answer>
```

What about duplicate authors?

Result Structure

Eliminate duplicates:

```
for $b in doc("bib.xml")/bib, $x in distinct-values($b/book/author/text())
return
<answer>
  <author> $x </author>
  { for $y in $b/book[author/text()=$x]/title
    return $y }
</answer>
```
**SQL and XQuery Side-by-side**

**Product(pid, name, maker)**
**Company(cid, name, city)**

Find all products made in Seattle

**SQL**

```
SELECT x.name
FROM Product x, Company y
WHERE x.maker=y.cid
    and y.city="Seattle"
```

**XQuery**

```
for $r in doc("db.xml")/db,
    $x in $r/Product/row,
    $y in $r/Company/row
where
    $x/maker/text()=$y/cid/text()
    and $y/city/text() = "Seattle"
return { $x/name }
```

---

**XML Example**

```xml
<db>
    <product>
        <row>
            <pid> ??? </pid>
            <name> ??? </name>
            <maker> ??? </maker>
        </row>
        <row> .... </row>
    </product>
    ....
</db>
```
XQuery Variables

- **for $x in expr** -- binds $x to each value in the list expr

- **let $x := expr** -- binds $x to the entire list expr
  - Useful for common sub-expressions and for aggregations

XQuery: LET

Find all publishers that published more than 100 books:

```xml
<big_publishers>
{   for $p in distinct-values(//publisher/text())
    let $b := /db/book[publisher/text() = $p]
    where count($b) > 100
    return <publisher> { $p } </publisher>
}
</big_publishers>
```

$b is a collection of elements, not a single element

count = a (aggregate) function that returns the number of elms
FOR vs. LET

FOR
• Binds *node variables* → iteration

LET
• Binds *collection variables* → one value

---

FOR

```xml
for $x$ in /bib/book
return <result> { $x } </result>
```

LET

```xml
let $x$ := /bib/book
return <result> { $x } </result>
```

---

Returns:

```
<result> <book>...</book> </result>
<result> <book>...</book> </result>
<result> <book>...</book> </result>
...
```

Returns:

```
<result> <book>...</book> </result>
<book>...</book>
<book>...</book>
...

</result>
```
Collections in XQuery

- Ordered and unordered collections
  - `/bib/book/author/text()` = an ordered collection: result is in document order
  - `distinct-values(/bib/book/author/text())` = an unordered collection: the output order is implementation dependent

- `let $a := /bib/book` → $a is a collection
- `$b/author` → a collection (several authors...)

```xquery
return <result> { $b/author } </result>
```

SQL and XQuery Side-by-side

Product(pid, name, maker, price) Find all product names, prices, sort by price

```sql
SELECT x.name, x.price
FROM Product x
ORDER BY x.price
```

```xquery
for $x in doc("db.xml")/db/Product/row
order by $x/price/text()
return <answer>
  { $x/name, $x/price }
</answer>
```
XQuery’s Answer

<answer>
   <name> abc </name>
   <price> 7 </price>
</answer>
<answer>
   <name> def </name>
   <price> 23 </price>
</answer>

Notice: this is NOT a well-formed document!
(WHY ???)

Producing a Well-Formed Answer

<myQuery>
{  for $x in doc("db.xml")/db/Product/row
    order by $x/price/text()
    return <answer>
        { $x/name, $x/price }
    </answer>
}
</myQuery>
XQuery’s Answer

```xml
<myQuery>
<answer>
  <name> abc </name>
  <price> 7 </price>
</answer>
<answer>
  <name> def </name>
  <price> 23 </price>
</answer>
.
.
</myQuery>

Now it is well-formed!
```

SQL and XQuery Side-by-side

For each company with revenues < 1M count the products over $100

```sql
SELECT y.name, count(*)
FROM Product x, Company y
WHERE x.price > 100 and x.maker=y.cid and y.revenue < 1000000
GROUP BY y.cid, y.name
```

```xml
for $r in doc("db.xml")/db,
  $y in $r/Company/row[revenue/text()<1000000]
return
  <proudCompany>
    <companyName> { $y/name/text() } </companyName>
    <numberOfExpensiveProducts>
      { count($r/Product/row[maker/text()=$y/cid/text()][price/text()>100]) } 
    </numberOfExpensiveProducts>
  </proudCompany>
```
SQL and XQuery Side-by-side

Find companies with at least 30 products, and their average price

SELECT y.name, avg(x.price) FROM Product x, Company y WHERE x.maker=y.cid GROUP BY y.cid, y.name HAVING count(*) > 30

for $r in doc("db.xml")/db, $y in $r/Company/row let $p := $r/Product/row[maker/text()=y/cid/text()] where count($p) > 30
return
  <theCompany>
    <companyName> { $y/name/text() } </companyName>
    <avgPrice> avg($p/price/text()) </avgPrice>
  </theCompany>

XQuery

Summary:

• FOR-LET-WHERE-RETURN = FLWR
Practical Example: Galax

```
$ more iis0.xq
<bib> {

    for $x in doc("bib.xml")/bib/book/author[first-name]
    return <result> {$x} </result>

} </bib>

$ galax-run iis0.xq
<bib><result><author><first-name>Rick</first-name>
    <last-name>Hull</last-name>
</author></result></bib>

http://www.galaxquery.org
```

Outline

- XML parsers
- XPath
- XQuery
- XML publishing
XML from/to Relational Data

- XML publishing:
  - relational data $\rightarrow$ XML
- XML storage:
  - XML $\rightarrow$ relational data

Client/server DB Apps

Relational Database $\rightarrow$ Network $\rightarrow$ Application

Tuple streams $\rightarrow$ SQL
XML Publishing

- Relational schema:
  - Student(sid, name, address)
  - Course(cid, title, room)
  - Enroll(sid, cid, grade)
First thing to do: design the DTD:

```xml
<!ELEMENT xmlview (course*)>
<!ELEMENT course (title, room, student*)>
<!ELEMENT student (name, address, grade)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT grade (#PCDATA)>
<!ELEMENT title (#PCDATA)>
```

Group by courses: redundant representation of students

Other representations possible too
Now we write an XQuery to export relational data → XML

Note: result is in the right DTD

```xml
<xmlview>
  { for $x in /db/Course/row
    return
    <course>
      <title> { $x/title/text() } </title>
      ...        
    </course>
  } 
</xmlview>
```

**XML Publishing**

Query: find Mary’s grade in Operating Systems

**XQuery**

```xml
for $x in /xmlview/course[title/text()=“Operating Systems”],
$y in $x/student/[name/text()=“Mary”]
return <answer> $y/grade/text() </answer>
```

**SQL**

```sql
SELECT Enroll.grade
FROM Student, Enroll, Course
WHERE Student.name=“Mary” and Course.title=“OS”
    and Student.sid = Enroll.sid and Enroll.cid = Course.cid
```

Can be done automatically
XML Publishing

How do we choose the output structure?
- Determined by agreement with partners/users
- Or dictated by committees
  - XML dialects (called *applications*) = DTDs
- XML Data is often nested, irregular, etc
- No normal forms for XML

Conclusion

- **XML parsers** are required for detailed usage of XML encoded data
- **XPath** provides a simple query language
- **XQuery** is an enhanced version of XPath (SQL like)