Announcements

Readings:
- Simeon, Wadler: The essence of XML. POPL 2003. Review due Wednesday
- Other suggested readings:
  - http://www.xml.com/pub/98/10/guide0.html
  - http://www.w3.org/TR/2004/WD-xquery-use-cases-20031002/
- Guest Lecturer: Alan Fekete, Transactions, Monday, Wednesday

Outline

- XML: syntax, semantics, data, DTDs
- XPath
- XQuery

XML Syntax

```xml
<book price="55" currency="USD">
  <title>Foundations of Databases</title>
  <author>Abiteboul</author>
  ...
  <year>1995</year>
</book>
```

Attributes are alternative ways to represent data.

```xml
<person id="o555" name="Jane"/>
<person id="o456" name="Mary">
  <children idref="o123 o555"/>
</person>
<person id="o123" mother="o456" name="John"/>
```

XML Syntax: attributes are alternative ways to represent data.

XML Syntax: oids and references in XML are just syntax.
**XML Semantics: a Tree!**

```
<data>
  <person id="0555">
    <name> Mary </name>
    <address>
      <street> Maple </street>
      <no> 345 </no>
      <city> Seattle </city>
    </address>
  </person>

  <person>
    <name> John </name>
    <address> Thailand </address>
    <phone> 23456 </phone>
  </person>
</data>
```

**Order matters!!**

---

**XML Data**

- **XML is self-describing**
  - Schema elements become part of the data
  - Rational schema: persons(name, phone)
    - In XML `<persons>`, `<name>`, `<phone>` are part of the data, and are repeated many times
  - Consequence: XML is much more flexible
  - XML = semi-structured data

**Relational Data as XML**

```
person

<table>
<thead>
<tr>
<th>name</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>3634</td>
</tr>
<tr>
<td>Sue</td>
<td>6343</td>
</tr>
<tr>
<td>Dick</td>
<td>6363</td>
</tr>
</tbody>
</table>
```

**XML is Semi-structured Data**

- Missing attributes:
  ```
  <person>
    <name> John </name>
    <phone> 1234 </phone>
  </person>
  
  <person>
    <name> Joe </name>
  </person>
  ```
  - Could represent in a table with nulls

```
  name: phone
  John: 1234
  Joe: -
```

- Repeated attributes
  ```
  <person>
    <name> Mary </name>
    <phone> 2345 </phone>
    <phone> 3456 </phone>
  </person>
  ```
  - Two phones!

- Impossible in tables:
  ```
  name: phone:
  Mary: 2345
  ??
  ```

**XML is Semi-structured Data**

- Attributes with different types in different objects
  ```
  <person>
    <name> John </name>
    <phone> 1234 </phone>
  </person>
  ```
  - Structured name!

- Nested collections (non 1NF)
- Heterogeneous collections:
  - `<db>` contains both `<book>`s and `<publisher>`s
Document Type Definitions (DTD)

- Part of the original XML specification
- To be replaced by XML Schema
  - Much more complex
- An XML document may have a DTD
- XML document:
  - well-formed = if tags are correctly closed
  - Valid = if it has a DTD and conforms to it
- Validation is useful in data exchange

Very Simple DTD

```xml
<!DOCTYPE company [
  <ELEMENT company ( ((person|product)*) )>
  <ELEMENT person ( ssn, name, office, phone? )>
  <ELEMENT ssn ( #PCDATA )>
  <ELEMENT name ( #PCDATA )>
  <ELEMENT office ( #PCDATA )>
  <ELEMENT phone ( #PCDATA )>
  <ELEMENT product ( pid, name, description? )>
  <ELEMENT pid ( #PCDATA )>
  <ELEMENT description ( #PCDATA )>
]>
```

Very Simple DTD

Example of valid XML document:

```xml
<company>
  <person>  <ssn> 123456789 </ssn>  
  <name> John </name>  
  <office> B432 </office>  
  <phone> 1234 </phone>  
</person>
  <person>  <ssn> 987654321 </ssn>  
  <name> Jim </name>  
  <office> B123 </office>  
</person>
  <product> ... </product>
</company>
```

DTD: The Content Model

- Content model:
  - Complex = a regular expression over other elements
  - Text-only = #PCDATA
  - Empty = EMPTY
  - Any = ANY
  - Mixed content = (#PCDATA | A | B | C)*

DTD: Regular Expressions

```
sequence

<table>
<thead>
<tr>
<th>DTD</th>
<th>XML</th>
</tr>
</thead>
</table>
| <ELEMET name (firstName, lastName)>
| optional |
| <ELEMENT name (firstName?, lastName?)>
| Kleene star
| <ELEMENT person (name, phone*)>
| alternation
| <ELEMENT person (name, phone,email*)>
```

Very Simple DTD

```xml
<!DOCTYPE company [
  <ELEMENT company ( ((person|product)*) )>
  <ELEMENT person ( ssn, name, office, phone? )>
  <ELEMENT ssn ( #PCDATA )>
  <ELEMENT name ( #PCDATA )>
  <ELEMENT office ( #PCDATA )>
  <ELEMENT phone ( #PCDATA )>
  <ELEMENT product ( pid, name, description? )>
  <ELEMENT pid ( #PCDATA )>
  <ELEMENT description ( #PCDATA )>
]>
```
Very Simple DTD

Example of valid XML document:

```xml
<company>
  <person> <ssn> 123456789 </ssn> <name> John </name> <office> B432 </office> <phone> 1234 </phone> </person>
  <person> <ssn> 987654321 </ssn> <name> Jim </name> <office> B123 </office> </person>
</company>
```

Attributes in DTDs

```xml
<!ELEMENT person (ssn, name, office, phone?)>
<!ATTLIST person age CDATA #REQUIRED
  id ID           # REQUIRED
  manager IDREF #REQUIRED
  manages IDREFS #REQUIRED
>
<person age="25">
  <name> ... </name>
  <product> ... </product>
</person>
```

Attributes in DTDs

Types:
- CDATA = string
- ID = key
- IDREF = foreign key
- IDREFS = foreign keys separated by space
- (Monday | Wednesday | Friday) = enumeration
- NMTOKEN = must be a valid XML name
- NMTOKENS = multiple valid XML names
- ENTITY = you don’t want to know this

Attributes in DTDs

Kind:
- #REQUIRED = optional
- #IMPLIED = default value
- value #FIXED = the only value allowed

Using DTDs

- Must include in the XML document
- Either include the entire DTD:
  - `<!DOCTYPE rootElement [ ....... ]>`
- Or include a reference to it:
  - `<!DOCTYPE rootElement SYSTEM "http://www.myDTD.org"`
- Or mix the two... (e.g. to override the external definition)
Querying XML Data

- **XPath** = simple navigation through the tree
- **XQuery** = the SQL of XML
- **XSLT** = recursive traversal
  - will not discuss in class

Sample Data for Queries

```xml
<bib>
  <book>
    <publisher>Addison-Wesley</publisher>
    <author>Serge Abiteboul</author>
    <author>Rick Hull</author>
    <author>Victor Vianu</author>
    <author>Jeffrey D. Ullman</author>
    <title>Foundations of Databases</title>
    <year>1995</year>
  </book>
  <book>
    <publisher>Freeman</publisher>
    <author>Jeffrey D. Ullman</author>
    <title>Principles of Databases and Knowledge Base Systems</title>
    <year>1998</year>
  </book>
</bib>
```

Data Model for XPath

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

Result: 1995 <year>1998 <year>

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

Result: empty (there were no papers)

XPath: Simple Expressions

```
//author
```

Result: <author>Serge Abiteboul</author> <author>Rick Hull</author> <author>Victor Vianu</author> <author>Jeffrey D. Ullman</author>

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

Result: Rick <first-name>

XPath: Restricted Kleene Closure

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

Result: Serge Abiteboul Victor Vianu Jeffrey D. Ullman

Rick Hull doesn’t appear because he has first-name, last-name

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

XPath: Text Nodes
Xpath: Wildcard

`//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 to be an attribute

Xpath: Predicates

`/bib/book/author[firstname]`

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

Xpath: More Predicates

`/bib/book[author/@age < 25]`

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

Xpath: More Predicates

`/bib/book[@price < 60]`

`/bib/book/author[@age < 25]`

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/last-name` matches...
XQuery

• Based on Quilt, which is based on XML-QL
• Uses XPath to express more complex queries

Sample Data for Queries (more or less)

FOR-WHERE-RETURN

Find all book titles published after 1995:

Result:

FOR-WHERE-RETURN

Equivalently (perhaps more geekish)

FOR-WHERE-RETURN

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

We can construct whatever XML results we want!
Answer

<answer>
  <what> How to cook a Turkey </what>
  <when> 2003 </when>
</answer>

<answer>
  <what> Cooking While Watching TV </what>
  <when> 2004 </when>
</answer>

<answer>
  <what> Turkeys on TV </what>
  <when> 2002 </when>
</answer>


FOR-WHERE-RETURN

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

FOR $x$ IN document("bib.xml")/bib/book
RETURN <answer>
  <title> $x$/title/text() </title>
  <year> $x$/year/text() </year>
</answer>

XQuery: Nesting
For each author of a book by Morgan Kaufmann, list all books she published:

FOR $b$ IN document("bib.xml")/bib,
  $a$ IN $b/book[publisher/text()="Morgan Kaufmann"]/author
RETURN <result>
  { $a,
    FOR $t$ IN $b/book[author/text()=$a/text()] 
    RETURN $t
  }
</result>

In the RETURN clause comma concatenates XML fragments.

XQuery

Result:

XQuery

For all books with more than 3 authors:

FOR $x$ IN document("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
Print all authors who published more than 3 books

FOR $b$ IN document("bib.xml")/bib,
  Sa IN $b$/book/author/text()
WHERE count($b/book/author/text()=$a)>3
RETURN <author> {  $a  } </author>

What’s wrong?

Aggregates
Be aware of duplicates!

FOR $b$ IN document("bib.xml")/bib,
  Sa IN distinct-values($b/book/author/text())
WHERE count($b/book/author/text()=$a)>3
RETURN <author> {  $a  } </author>

XQuery
Find books whose price is larger than average:

FOR $b$ in document("bib.xml")/bib
LET $a:=avg($b/book/price/text())
FOR $x$ in $b/book$
WHERE $x/price/text() > $a$
RETURN $x$

LET binds a variable to one value.
FOR iterates a variable over a list of values
We will come back to that

FOR-WHERE-RETURN
• “Flatten” the authors, i.e. return a list of
  (author, title) pairs

FOR $b$ IN document("bib.xml")/bib/book,
  $x$ IN $b/title/text()$, $y$ IN $b/author/text()$
RETURN $<answer>
  $<author>  $x  </author>$
  $<title>  $y  </title>$
</answer>

FOR-WHERE-RETURN
• For each author, return all book titles he/she wrote

FOR $b$ IN document("bib.xml")/bib,
  $x$ IN $b/book/author/text()$
RETURN $<answer>
  $<author>  $x  </author>$
  $<title>  $y  </title>$
</answer>

What about duplicate authors?

FOR-WHERE-RETURN
• Same, but eliminate duplicate authors:

FOR $b$ IN document("bib.xml")/bib
LET $a := distinct-values($b/book/author/text())$
FOR $x$ IN $a$
RETURN $<answer>
  $<author>  $x  </author>$
  $<title>  $y  </title>$
</answer>
FOR-WHERE-RETURN

• Same thing:

```
FOR $b IN document("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, price) Find all product names, prices, sort by price

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

```
FOR $x in document("db.xml")/db/Product/row
ORDER BY $x/price/text()
RETURN <answer>
  { $x/name, $x/price }
</answer>
```

Answers

```
Name  Price
abc   7
def   23

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

Producing a Well-Formed Answer

```
<myQuery>
{ FOR $x in document("db.xml")/db/Product/row
ORDER BY $x/price/text()
RETURN <answer>
  { $x/name, $x/price }
</answer>
}
</myQuery>
```

XQuery’s Answer

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

Now it is well-formed!

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

SQL and XQuery Side-by-side

Product(pid, name, maker, price) Find all products made in Seattle

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

```
FOR $x in document("db.xml")/db, $y in $x/Company/row,
    WHERE $x/maker/text()=$y/cid/text() and $y/city/text() = "Seattle"
RETURN { $x/name }
```

```
FOR $y in /db/Company/row[city/text()="Seattle"], $x in /db/Product/row[maker/text()=$y/cid/text()]
RETURN { $x/name }
```
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
GROUP BY y.cid, y.name
HAVING count(*) > 30
```

SQL and XQuery Side-by-side

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

```
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
```

FOR v.s. LET

FOR
- Binds node variables
- iteration

LET
- Binds collection variables
- one value

FOR v.s. LET

```
FOR $x IN /bib/book
RETURN <result> { $x } </result>
```

Returns:
```
<result>
  <book>
    <title>Result 1</title>
    <author>Author 1</author>
  </book>
</result>
```

```
LET $x = /bib/book
RETURN <result> { $x } </result>
```

Returns:
```
<result>
  <book>
    <title>Result 1</title>
    <author>Author 1</author>
  </book>
</result>
```

XQuery

Summary:
- FOR-LET-WHERE-RETURN = FLWR
- FORLET Clauses
  - List of tuples
  - WHERE Clause
    - List of tuples
  - RETURN Clause
    - Instance of Xquery data model
Collections in XQuery

- Ordered and unordered collections
  - /bib/book/author/text() is 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...)

```
RETURN <result> ($b/author) </result>
```

XML from/to Relational Data

- XML publishing:
  - relational data → XML

- XML storage:
  - XML → relational data

Client/server DB Apps

- Tuple streams
- Network
- Application
- SQL

XML Publishing

- Tuple streams
- XML
- Web
- Application
- SQL
- XPath/ XQuery

XML Publishing

- Relational schema:
  - Student(sid, name, address)
  - Course(cid, title, room)
  - Enroll(sid, cid, grade)

```
<Student>
    <sid>1</sid>
    <name>John Doe</name>
    <address>123 Main St, Seattle</address>
</Student>

<Course>
    <cid>CS101</cid>
    <title>Operating Systems</title>
    <student>John Doe</student>
</Course>

Enroll
```

Group by courses: redundant representation of students

Other representations possible too
XML Publishing

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)>
```

Now we write an XQuery to export relational data to XML

Note: result is the right DTD

```xml
<xmlview>
  <FOR $x IN /db/Course/row
    RETURN <course>
      <title> { $x/title/text() } <title>
      <room> { $x/room/text() } <room>
      <FOR $y IN /db/Enroll/row[cid/text() = $x/cid/text()]
        { $y/Student/row[sid/text() = $x/sid/text()]
          RETURN <student>
            <name> { $y/name/text() } <name>
            <address> { $y/address/text() } <address>
            <grade> { $y/grade/text() } <grade>
          </student>
        }
      </FOR>
    </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
```

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

XML Storage

- Most often the XML data is small
  - E.g. a SOAP message
  - Parsed directly into the application (DOM API)
- Sometimes XML data is large
  - Need to store/process it in a database
- The XML storage problem:
  - How do we choose the schema of the database?

Three solutions:

- Schema derived from DTD
- Storing XML as a graph: “Edge relation”
- Store it as a BLOB
  - Simple, boring, inefficient
  - Won’t discuss in class
Designing a Schema from DTD

Design a relational schema for:

```xml
<!DOCTYPE company [< !ELEMENT company (person | product)* >
< !ELEMENT person (ssn, name, office?, phone*) >
< !ELEMENT name (#PCDATA)>
< !ELEMENT ssn ( #PCDATA )>
< !ELEMENT office ( #PCDATA )>
< !ELEMENT phone ( #PCDATA )>
< !ELEMENT product (pid, name, (price, availability | description)*)
< !ELEMENT pid ( #PCDATA )>
< !ELEMENT description ( #PCDATA )>
]
```

First, construct the DTD graph:

We ignore the order

Next, design the relational schema, using common sense.

Person(ssn, name, office)
Phone(ssn, phone)
Product(pid, name, price, avail., descr.)

Which attributes may be NULL ? (Look at the DTD)

What happens to queries:

```sql
FOR $x IN /company/product[description]  
RETURN <answer> { $x/name, $x/description } </answer>
```

SELECT Product.name, Product.description  
FROM Product  
WHERE Product.description IS NOT NULL

Storing XML as a Graph

Sometimes we don’t have a DTD:
- How can we store the XML data ?

Every XML instance is a tree
- Store the edges in an Edge table
- Store the #PCDATA in a Value table

Can be ANY XML data (don’t know DTD)

<table>
<thead>
<tr>
<th>Source</th>
<th>Tag</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>db</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>book</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>title</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>author</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>book</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>title</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>author</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>db</td>
</tr>
<tr>
<td>1</td>
<td>book</td>
</tr>
<tr>
<td>2</td>
<td>title</td>
</tr>
<tr>
<td>3</td>
<td>Complex guide</td>
</tr>
<tr>
<td>4</td>
<td>Chamberlin</td>
</tr>
<tr>
<td>5</td>
<td>Author</td>
</tr>
<tr>
<td>6</td>
<td>Value</td>
</tr>
</tbody>
</table>
Storing XML as a Graph

What happens to queries:

```
FOR $x$ IN /db/book[author/text()="Chamberlin"]
RETURN $x$/title
```

Storing XML as a Graph

What happens to queries:

```
SELECT vtitle.value
FROM Edge xdb, Edge xbook, Edge xauthor, Edge xtitle,
Value vauthor, Value vtitle
WHERE xdb.source = 0 and xdb.tag = 'db'
and xbook.dest = xbook.source
and xbook.tag = 'book'
and xauthor.dest = xauthor.source
and xauthor.tag = 'author'
and xtitle.dest = xtitle.source
and xtitle.tag = 'title'
and vauthor.value = 'Chamberlin'
and vtitle.source
```

Storing XML as a Graph

Edge relation summary:
- Same relational schema for every XML document:
  Edge(Source, Tag, Dest)
  Value(Source, Val)
- Generic: works for every XML instance
- But inefficient:
  - Repeat tags multiple times
  - Need many joins to reconstruct data

Other XML Topics

- Name spaces
- XML API:
  - DOM = "Document Object Model"
- XML languages:
  - XSLT
- XML Schema
- Xlink, XPointer
- SOAP

Available from [www.w3.org](http://www.w3.org) (but don’t spend rest of your life reading those standards!)

Old&New XML Research at UW

- Processing:
  - Query languages (XML-QL, a precursor of XQuery)
  - Tbolvi
  - XML updates
- XML publishing/storage
  - SilkRoute: silkroute.sourceforge.net
  - STORED
- XML tools
  - XML Compressor: Xmill - a very popular tool
  - XML Toolkit (sort, sagg, xgrep, xstrafe, etc): xmltk.sourceforge.net
- Theory:
  - Typing/typing
  - XPath/XQuery containment