XML and Semi-structured data

What is XML?

- Text annotation/marking language ("eXtensible Markup Language")

```
<BOOK genre="Science" format="Hardcover">
  <AUTHOR>
    <FIRSTNAME>Rich</FIRSTNAME>
    <LASTNAME>Feynman</LASTNAME>
  </AUTHOR>
  <TITLE>The Character of Physical Law</TITLE>
  <PUBLISHED>1980</PUBLISHED>
</BOOK>
```

- Think of markup as meta-data (data about data)
- Resulting document is structured like a tree

Success of XML

- Ability to represent “varying format data” (semi-structured)
- Ability to introduce new tags, led to publication of “standards” for many sub-areas.

**data exchange**

- Example: Chemical Markup Language

```
<molecule>
  <weight>234.5</weight>
  <Spectra>…</Spectra>
  <Figures>…</Figures>
</molecule>
```
**Semi-structured Data Management**

\[ \mathcal{L}_{\text{tell}} \xrightarrow{\text{TELL}} \mathcal{L}_{\text{question}} \xrightarrow{\text{ASK}} \mathcal{L}_{\text{answer}} \]

\[ \mathcal{L}_{\text{tell}} \equiv \text{XML document } \]
\[ \mathcal{L}_{\text{question}} \equiv \text{XPath, ( XQuery ) } \]
\[ \mathcal{L}_{\text{answer}} \equiv \text{XML document } \]
\[ \mathcal{L}_{\text{declare}} \equiv \text{DTD, XML Schema} \]

**The syntax of XML**

(Silberschatz or Kiffer text better)

**Sample XML document**

```xml
<bibliography>
  <book>
    <title> Foundations… </title>
    <author> Abiteboul </author>
    <author> Hull </author>
    <author> Vianu </author>
    <publisher> Addison Wesley </publisher>
    <year> 1995 </year>
  </book>
  ...
</bibliography>
```

**XML Terminology**

- tags: `book`, `title`, `author`, ...
- elements:
  - `<book>…</book>`
  - `<author>…</author>`
- elements are nested and bottom out at data values (hence form a tree)
- empty element: `<red></red>` abbrev. `<red/>`
- an XML document must be a single element ("root")

(a well formed XML document has properly nested matching tags)
More XML: Attributes

```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
- At most one occurrence of attribute per element
- Attribute value is a single string
  (Multiple values for an attribute separated by blank/tab)
  
  Good attributes are meta-data:
  - `status = "outOfPrint"`, `language = "English"`, `categ = "fiction"`

Example doc for XPath Queries

```xml
<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>
```

* XML Namespaces

- A way to share tags, attributes,…
- [http://www.w3.org/TR/REC-xml-names](http://www.w3.org/TR/REC-xml-names)
- `name ::= localtag OR prefix:tag`

```xml
<book xmlns:isbn="http://www.isbn.org/def">
  <title>…</title>
  <number>15</number>
  <isbn:number>…</isbn:number>
</book>
```

A local “number” tag, meaning how many books are in stock

The “number” tag of ISBN, which is the unique number assigned to each book edition by an international agency.

The ordered tree view of an XML document

(draw on board)
One way to represent Relational Data in XML

XML:

```
<persons>
  <row>
    <name>John</name>
    <phone>3634</phone>
  </row>
  <row>
    <name>Sue</name>
    <phone>6343</phone>
  </row>
  <row>
    <name>Dick</name>
    <phone>6363</phone>
  </row>
</persons>
```

XML vs Relational Data

- XML is self-describing
- Schema elements become part of the data (tags):
  ```xml
  <person> <name> </name> <phone> </phone> ...
  </person>
  ```
- So XML is more flexible because do not have to follow slavishly a single flat schema:
  **Semistructured Data**

Semistructured Data

- Fields may be **missing**
  ```xml
  <person> <name>bob</name> <phone>5-4544</phone> </person>
  <person> <name>anna</name> </person>
  ```
- Fields may be **repeated**
  ```xml
  <person> <name>bob</name> <phone>5-4544</phone> <phone>3-5436</phone> </person>
  ```
- Fields may be **nested**
  ```xml
  <person> <name> <first>bob</first> <last>jones</last> </name> </person>
  ```
- Fields may be **heterogeneous**
  ```xml
  <name> <first>bob</first> <last>jones</last> </name>
  <name> <first>bob</first> <mid>t</mid> <last>jones</last> </name>
  ```
- Collections may be **heterogeneous**
  ```xml
  <persons> <teacher> ... </teacher> <student> ... </student> </persons>
  ```

DTD – Document Type Definition

- A DTD is a schema/grammar for XML data
- A DTD says what elements and attributes are required or optional
  - Defines the formal structure of the doc
**DTD – An Example**

```xml
<!ELEMENT Basket (Cherry+, (Apple | Orange)* ) >
<!ELEMENT Cherry EMPTY>
<!ELEMENT Apple EMPTY>
<!ELEMENT Orange EMPTY>
<!ATTLIST Cherry flavor CDATA #REQUIRED>
<!ATTLIST Apple color CDATA #REQUIRED>
<!ATTLIST Orange location 'Florida' >
```

2 documents:

```xml
<Basket>
  <Apple/>
  <Cherry flavor='good'/>
  <Orange/>
</Basket>

<Basket>
  <Cherry flavor='good'/>
  <Apple color='red'/>
  <Apple color='green'/>
</Basket>
```

**DTD: !ELEMENT**

```xml
<!ELEMENT Basket (Cherry+, (Apple | Orange)* ) >
```

- **Name**
- **Children**

- !ELEMENT declares an element name, and what children elements it should have
- Content types:
  - Other elements
  - #PCDATA (parsed character data)
  - etc

**DTD: !ATTLIST**

```xml
<!ATTLIST   Cherry   flavor   CDATA   #REQUIRED>
```

- **Element**
- **Attribute**
- **Type**
- **Flag**

- !ATTLIST defines a list of attributes for an element
- * Attributes can be of different types, can be required or not required, and they can have default values.

**DTD – !ELEMENT in general**

- A regular expression describing the content has the following structure:
  - \( \exp_1, \exp_2, \exp_3, \ldots, \exp_k \): An ordered list of regular expressions
  - \( \exp^* \): An optional expression with zero or more occurrences
  - \( \exp^+ \): An expression with one or more occurrences
  - \( \exp^? \): An optional expression with zero or one occurrence
  - \( \exp_1 \mid \exp_2 \mid \ldots \mid \exp_k \): A set of alternative expressions
Attribute types in DTDs

Types include:
- CDATA = string
- (Monday | Wednesday | Friday) = enumeration
- ID = key
- IDREF = foreign key
- IDREFS = foreign keys separated by space

/* We will not discuss these in the course */

* DTDs as Grammars

- A DTD = a grammar
- A valid XML document = a parse tree for that grammar

* Problem with DTD: not in XML notation!!!
There is more advanced notation: XML Schema (not in this course!)

(*Relationship between DTD & Extended BNF*)

<!DOCTYPE paper [
  <!ELEMENT paper (section*)>
  <!ELEMENT section ((title,section*) | text)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT text (#PCDATA)>
]

Equivalent extended BNF

paper ::= section *
section ::= ( title section * ) | text
title ::= string
text ::= string

(* XML Schema *)

DTD: <!ELEMENT paper (title,author*,year,(journal,conference))>

becomes

<xsd:element name="paper" type="papertype"/>
<xsd:complexType name="papertype">
  <xsd:sequence>
    <xsd:element name="title" type="xsd:string"/>
    <xsd:element name="author" minOccurs="0"/>
    <xsd:element name="year"/>
    <xsd:choice>
      <xsd:element name="journal"/>
      <xsd:element name="conference"/>
    </xsd:choice>
  </xsd:sequence>
</xsd:complexType>
**XPath**

- [http://www.w3.org/TR/xpath/](http://www.w3.org/TR/xpath/)

**Querying XML Documents**

(based on notes by D.Suciu/UofW)

**Example doc for XPath Queries**

```xml
<bib>
  <book>
    <publisher> Addison-Wesley </publisher>
    <author> Serge Abiteboul </author>
    <author> Rick </author>
    <author> Hull </author>
  </book>
  <book>
    <publisher> Freeman </publisher>
    <author> Jeffrey D. Ullman </author>
    <author> Victor Vianu </author>
  </book>
<bib>
```

**Corresponding tree for XPath** *(a bit different, to make things resemble UNIX nested file paths)*

![Corresponding tree for XPath](image)
**Corresponding tree for XPath** (a bit different, to make things resemble UNIX nested file paths)

(draw on board)

**XPath:** Simple Expressions (matching element nodes)

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

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

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

*Result:* empty  (there were no papers)

**XPath:**

One way to think about it is that each initial part of a path “marks” certain nodes in the tree as being acceptable – part of the current collection. The next step in the path unmarks these and marks as acceptable only those children of previously marked nodes which pass some additional test.

(* In the full XPath, one can go from marked nodes to their descendants, parents, ancestors, left and right siblings.)*

**XPath: Restricted Kleene Closure**

```
//author
```

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

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

*Result:* `<first-name> Rick </first-name>`
**Xpath: Wildcard**

`//author/*`

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

* `Matches any element`

`/*/*/author` “authors at 3rd level”

**Xpath: matching Text Nodes**

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

Result: Serge Abiteboul  
Jeffrey D. Ullman

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

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

`/bib/book/*/name()`

**Xpath: matching Attribute Nodes**

`/bib/book/@price`

Result: `price="55"`
@price means that there is a price attribute with a value present

**Xpath: Qualifiers**

`//author[first-name]`

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

[first-name] = “has first-name element”
**Xpath: More Qualifiers**

\[
/bib/book/author[first-name][address[//zip][city]]/lastname
\]

*Result:* \(<lastname> \ldots </lastname>\)

"lastname of author

(which has firstname and

address (which has zip below it) and has city)"

---

**Xpath: Qualifiers with conditions on values**

\[
/bib/book[@price < "60"]
\]

\[
/bib/book[author/@age < "25"]
\]

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

---

**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/..` matches the parent of 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…

---

**XQuery**

- http://www.w3.org/TR/xquery/
**FLWR ("Flower") Expressions**

```
for ... let... for... let...
where...
return...
```

**XQuery**

“Find all book titles published after 1995”:

```
for $x in document("bib.xml")/bib/book
where $x/year > 1995
return $x/title
```

**Result:**

```
<title>Principles of Database and Knowledge Base Systems</title>
```

**XQuery: nested queries**

“For each author of a book by AW, list all books she published:”

```
for $a in document("bib.xml")
  /bib/book[publisher="AW”]/author
return <ans>
  { $a,
    for $t in /bib/book[author=$a]/title
    return $t
  }
</ans>
```

**Beware of forgetting the { and }; they mean “evaluate nested expression”**

```
<ans>
  <author>Serge Abiteboul</author>
  <title>Foundations of Databases</title>
</ans>
<ans>
  <author><first-name>Rick</first-name>  <last-name>Hull</last-name></author>
  <title>Foundations of Databases</title>
</ans>
<ans>
  <author>Victor Vianu</author>
  <title>Foundations of Databases</title>
</ans>
<ans>
  <author>Jeffrey D. Ullman</author>
  <title>Principles of Database and Knowledge Base Systems</title>
</ans>
```
XQuery: let expressions

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

- let $x := expr$ -- binds $x$ once to the entire list $expr$
  - Useful for common subexpressions and for aggregations

XQuery

“Find books whose price is larger than average”:

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

XQuery

```xquery
<count = a (aggregate) function that returns the number of elements in its argument set

<big_publishers>
  for $p$ in document("bib.xml")//publisher
  let $b := document("bib.xml")/book[publisher = $p]
  where count($b) >= 1
  return $p
</big_publishers>
```

If-Then-Else

```xquery
for $h$ in //catalogoue
return <catalogue>
  {$h/title,
   if $h/@type = "Journal"
   then $h/editor
   else $h/author
  }
</catalogue>
```
**Existential Quantifiers**

"Books which have some paragraph containing both the words sailing and windsurfing"

```
for $b$ in //book
where some $p$ in $b$//para satisfies
        contains($p$, "sailing")
        and contains($p$, "windsurfing")
return $b$/title
```

**Universal Quantifiers**

"Books in which all paragraphs contain the word sailing"

```
for $b$ in //book
where every $p$ in $b$//para satisfies
        contains($p$, "sailing")
return $b$/title
```

**e.g., Flattening**

- “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>
    <title> { $x$ } </title>
    <author> { $y$ } </author>
</answer>
```

**Result:**

```
<answer>
    <title> abc </title>
    <author> efg </author>
</answer>
<answer>
    <title> abc </title>
    <author> hkj </author>
</answer>
```
e.g., Re-grouping

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

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

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

What about duplicate authors?

- Same, but eliminate duplicate authors:

```xml
for $b$ in document("bib.xml")/bib
let $a$ := distinct-values($b/book/author/text() )
for $x$ in $a$
return
<answer>
    <author> { $x$ } </author>
    { for $y$ in $b/book[author=$x]/title
        return $y$ }
</answer>
```

distinct-values eliminates duplicates (but must be applied to a collection of text values, not of elements)

SQL and XQuery Side-by-side

‘Find all product names, prices’

**Product(pid, name, maker, price)**

```xml
<db>
    <Product>
        <row>
            <pid>1234</pid> <name>"bulb"</name> <maker>
            <row/>
        </row>
    </Product>
</db>
```

**SQL**

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

**XQuery**

```xml
<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 ???)
**Query Producing a Well-Formed Answer**

```
<myQuery>
  { for $x in document("db.xml")/db/Product/row
    return <row>
      { $x/name, $x/price }
    </row>
  }
</myQuery>
```

**Xquery’s Answer**

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

**Now it is well-formed!**

**SQL and XQuery Side-by-side**

**“Find all products made in Seattle”**

**SQL**

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

**Compact XQuery**

```
for $x in /db/Product/row[maker="Seattle"],
  $y in /db/Company/row[city="Seattle"]
return $x/name
```

**XQuery**

```
for $x in document("db.xml")/db/Product/row
return <row>
  { $x/name, $x/price }
</row>
```

**Now it is well-formed!**

```
<product>
  <row>
    <pid>123</pid>
    <name>abc</name>
    <maker>efg</maker>
  </row>
  ...
</product>
```
SQL and XQuery Side-by-side

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

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

```xquery
for $r in document("db.xml")/db,
  $c in $r/Company[row[revenue<1000000]]
return
  <proudCompany>
    <companyName> {$c/name } </companyName>
    <numberOfExpensiveProducts>
      { count($r/Product[row[maker=$c/cid][price>100]] ) }
    </numberOfExpensiveProducts>
  </proudCompany>
```

SQL and XQuery Side-by-side

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

```sql
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
```

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