From HTML to XML

- HTML describes the presentation of the content
- XML describes only the content

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

Separation of content from presentation allows the content to be presented easily in different looks.

Other nice features of XML

- **Portability**: Just like HTML, you can ship XML data across any platforms
  - Relational data requires heavy-weight protocols, e.g., JDBC
- **Flexibility**: You can represent any information (structured, semi-structured, documents, …)
  - Relational data is best suited for structured data
- **Extensibility**: Since data describes its own schema, you can change it easily
  - Relational schema is rigid and difficult to change
- **“Publishability”**: We need new data models, query languages, query processing and optimization techniques

XML terminology

- **Tag names**: book, title, author
- **Start tags**: `<book>`, `<title>`, `<author>`
- **End tags**: `</book>`, `</title>`, `</author>`
- An element is enclosed by a pair of start and end tags: `<book>…</book>`
  - Elements can be nested: `<book>…<title>…</title>…</book>`
  - Empty elements: `<is_textbook/>`
  - Can be abbreviated: `<is_textbook>`
- Elements can also have attributes: `<book ISBN="…" price="80.00">`

Well-formed XML documents

A well-formed XML document

- Follows XML lexical conventions
  - Wrong: `<section> We show that x < 0… </section>`
  - Right: `<section> We show that x &lt; 0… </section>`
- Contains a single root element
- Has tags that are properly matched and elements that are properly nested
  - Right: `<section>…<subsection>…</subsection>…</section>`
  - Wrong: `<section>…<subsection>…</subsection>…</section>`

More XML features

- **Comments**: `<!-- Comments here… -->`
- **CDATA**: `<![CDATA[Tags: <book>, …]]>`
- **ID’s and references**
  - `<person id="012">name="Homer"</person>`
  - `<person id="034">father="012" mother="034" name="Bart"</person>`
- **Namespaces** allow external schemas and qualified names
  - `<book xmlns:myCitationStyle="http://…">myCitationStyle:title</book>`
- **Processing instructions** for apps: `<? java applet…?>`
- And more…
Valid XML documents

- A valid XML document conforms to a Document Type Definition (DTD)
  - A DTD is optional
- A DTD specifies
  - A grammar for the document
  - Constraints on structures and values of elements, attributes, etc.

```xml
<?xml version="1.0"?>
<DOCTYPE book [ 
  <!ELEMENT book (title, author*, publisher?, section+)>
  <!ATTLIST book ISBN CDATA #REQUIRED>
  <!ATTLIST book year CDATA #IMPLIED>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT author (#PCDATA)>
  <!ELEMENT section (#PCDATA | title | section)>
]>
```

Semistructured data model of Lore

- Graph-based, unordered, edge-labeled

```
<table>
<thead>
<tr>
<th>Author</th>
<th>Title</th>
<th>Book</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>War and Peace</td>
<td>10.50</td>
</tr>
</tbody>
</table>
```

Ordered tree model of YAT

- Tree-based, ordered, node-labeled, with references

```
<table>
<thead>
<tr>
<th>Book</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
</tr>
<tr>
<td>Author</td>
</tr>
</tbody>
</table>
```

Data models for XML

- Graph and tree models used in research
  - Semistructured model of Lore and TSIMMIS (Stanford)
  - Ordered tree model of YAT (INRIA)
- Document Object Model (DOM)
  - Object-oriented programming interface for XML
- XML Infoset
- Data models for various XML query languages
  - Data model defined by XML Query Working Group for XPath and XQuery

Query languages for XML

<table>
<thead>
<tr>
<th>Data model</th>
<th>Real XML</th>
<th>Idealized XML</th>
<th>Simple graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPath, XQL</td>
<td>XML-QL, Lorel, YATL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XSLT, Quilt (XQuery)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Expressive power

- Navigation, selection, SPJ, regexp, OQL, conditional, recursion
XML-QL

- Data model: a (totally) ordered or a (totally) unordered graph
- Query language
  - WHERE clause to bind variables and test predicates
  - CONSTRUCT clause to build output XML structures
- Features
  - XML patterns, regexp path expressions
  - Joins on multiple input sources
  - Skolem functions for grouping

XML-QL: XML patterns

- Retrieve the titles of the books written by Abiteboul before 2000

WHERE
- Scan bib.xml
- match the pattern to obtain all (S, ISBN, S) bindings
- Select those that pass the predicate
- Construct output for each (S, ISBN, S) binding obtained in WHERE

CONSTRUCT
- resultBook ISBN=$isbn
- resultTitle=$t

XML-QL: joins

- Retrieve all reviews for the books written by Abiteboul

WHERE
- Retrieve the titles of the books written by Abiteboul, together with their reviews, if any

WHERE
- If a book has no review, it will not be in the output
- Book title appears multiple times if there are multiple reviews

XML-QL: outerjoins (slide 1)

- Use nested queries with outerjoin semantics

WHERE
- Okay for subquery to return empty result

XML-QL: outerjoins (slide 2)

- What kind of elements are found in the content of the element corresponding to the book with ISBN of 10?

Regexp that matches any sequence of elements
XML-QL: Skolem functions (slide 1)

- Retrieve the titles of all books, grouped first by year and then by publisher

```
WHERE
<bib>
  <book year=$y>
    <title>$t</title>
    <publisher>$p</publisher>
  </book>
</bib> IN "bib.xml",

CONSTRUCT
  <bookByYear id=F1($y)>
    <bookByYearPublisher id=F2($y,$p)>
      <bookTitle>$t</bookTitle>
    </bookByYearPublisher>
  </bookByYear>
```

Automatic fusion of all elements with the same id attribute

```
<bookByYear id=F1(1995)>
  <bookByYearPublisher id=F2(1995, Addison Wesley)>
    <title>Intro to DB</title>
  </bookByYearPublisher>
</bookByYear>

<bookByYear id=F1(1995)>
  <bookByYearPublisher id=F2(1995, Addison Wesley)>
    <title>Intro to Java</title>
  </bookByYearPublisher>
</bookByYear>

<bookByYear id=F1(1995)>
  <bookByYearPublisher id=F2(1995, Addison Wesley)>
    <title>Intro to DB</title>
    <title>Intro to Java</title>
  </bookByYearPublisher>
</bookByYear>
```

XML-QL: Skolem functions (slide 2)

- Use of Skolem functions may lead to unintended sharing and even cycles

```
WHERE
<bib>
  <book year=$y>
    <title>$t</title>
    <publisher>$p</publisher>
  </book>
</bib> IN "bib.xml",

CONSTRUCT
  <bookByYear id=F1($y)>
    <bookByPublisher id=F2($p)>
      <bookTitle>$t</bookTitle>
    </bookByPublisher>
  </bookByYear>
```

```
bookByPublisher id=F2(Addison Wesley)
bookTitle bookTitle

bookByYear id=F1(1995)
bookByPublisher id=F2(Addison Wesley)
bookTitle

bookByYear id=F1(2000)
bookByPublisher id=F2(Addison Wesley)
bookTitle

bookByYear id=F1(1995)
bookByPublisher id=F2(Addison Wesley)
bookTitle

bookByYear id=F1(2000)
bookByPublisher id=F2(Addison Wesley)
bookTitle
```

XML-QL: summary

- Advantages
  - XML patterns look very familiar
  - Can express selection, projection, join, grouping
  - Can construct deeply nested XML elements

- Limitations
  - Problems with arbitrary use of Skolem functions
  - Preserving structure and hierarchy is difficult
  - No disjunction, aggregation, quantifiers, etc.
  - Data model ignores some XML details

XPath

- W3C recommendation; building block for other W3C standards (XSLT, Xlink, XPointer, XQuery, …)
- A query is an expression (location path)
  - Consists of a series of location steps separated by “/”
  - Describes a single navigation path (starting from a context node) in the input XML document
  - Returns a list of nodes in the input

- Example
  - Context node: root of the document bib.xml
  - Location path with three location steps:
    - child::bib/child::book[@ISBN=10]/descendent::section[position()=1]
    - Returns the first section in the book with ISBN 10

XPath: location steps

- Each location step consists of an axis, a node test, and a list of predicates
- Axes
  - Self, attribute, parent, child, ancestor (or self), desendent (or self), following, following-sibling, preceding, prededing-sibling, namespace
- Node test
  - Name test (e.g., book, section, *)
  - Type test (e.g., text(), comment(), node())
- Example
**XQuery**

- XQuery is currently a W3C working draft defined by XML Query Working Group
  - Data model is defined to work with XPath
  - A query expression can be
    - XPath expressions
    - FLWR (🗘) expressions
    - Quantified expressions
    - Aggregation, conditional, sorting, filter, and more...

**XQuery: FLWR expressions**

- Retrieve the titles of books written by Abiteboul before 2000, together with their inventory
  
  ```xml
  FOR $b IN document("bib.xml")/book[@year<2000]
  WHERE $b/author/lastname="Abiteboul"
  ```

**XQuery: quantified expressions**

- Find titles of books in which XML is mentioned in the some section
  
  ```xml
  FOR $b IN //book
  WHERE (SOME $section IN $b//section SATISFIES contains($section, "XML"))
  RETURN $b/title
  ```

- Find titles of books in which XML is mentioned in every section
  
  ```xml
  FOR $b IN //book
  WHERE (EVERY $section IN $b//section SATISFIES contains($section, "XML"))
  RETURN $b/title
  ```

**XQuery: aggregation**

- List each publisher and the average price of its books
  
  ```xml
  FOR $publisher IN DISTINCT(document("bib.xml")//publisher)
  LET $price := AVG(document("inventory.xml")//book[publisher=$publisher]/@price)
  RETURN <resultPublisher><publisher>$publisher/text()</publisher><averagePrice>$price</averagePrice></resultPublisher>
  ```

**XQuery: conditional, sorting**

- Make a list of books ordered by their titles; for journals, include the editors, and for others, include the authors
  
  ```xml
  FOR $b IN /book
  RETURN <book><title>$b/title/text()</title>
  IF $b/@type = "journal" THEN $b/editor ELSE $b/author</book>
  SORTBY (title)
  ```
XQuery: filter

- Filter returns a shallow copy of the nodes selected by the filtering expression, preserving any relationships that exist among them.

Before Filtering: $doc

After Filtering: filter($doc(\node{A} | \node{B})

- Generate a table of contents

```xquery```

```xquery```

XQuery: summary

- Learn from previous experience
- Make sure it is useful
  - Stick to real XML
  - Leverage existing standards (e.g., XPath)
- Make sure it is semantically clean
  - Still need work (e.g., confusing, implicit type casting in XPath)
  - `/book[title=""]` also returns books with no title because empty set is cast to empty string!
- To become the SQL for XML?

XSLT

- W3C recommendation
- XML-to-XML rule-based transformation language
- An XSLT program is an XML document itself
- Used mostly as a stylesheet language

```xquery```

XSLT program

- An XSLT program is a valid XML document containing
  - Elements in the `<xsl>` namespace
  - Elements in user namespace
- The result of evaluating an XSLT program on an input XML document = the XSLT document where each `<xsl>` element has been replaced with the result of its evaluation
- Uses XPath as a sub-language

XSL elements

- Elements describing rewriting rules
  - `<xsl:template>`
- Elements describing rule execution control
  - `<xsl:apply-template>`
  - `<xsl:call-template>`
- Elements describing instructions
  - `<xsl:for-each>`, `<xsl:if>`, `<xsl:sort>`, etc.

<xsl:template>

- Basic XSLT concept; describes a rewriting rule
- Example: a rule to convert a titled book published in 1995 to a `resultTitle` element in the output

```xml```

- Example: a rule to “render” a book title in italics in HTML

```xml```
Another XSLT example

- Table of contents again

```xml
<template match="$/book">
  <resultTitle><value-of select="title"/></resultTitle>
  <template match="author">
    <author><xsl:value-of select="." /></author>
  </template>
  <template match="section">
    <h2>Section <number><xsl:value-of select="." /></h2>
    <ol><xsl:apply-templates select="section" /></ol>
  </template>
</template>
```

XSLT: summary

- A stylesheet language, but could be considered a query language too
  - Very expressive: full recursion; easily non-terminating
  - Is it “declarative” enough?
  - How much optimization is possible?

System issues with XML

- XML publishing
  - Publish existing data in relational databases as XML
  - XML views over relational data
- XML storage
- XML query processing
  - Again, where? How does it differ from traditional database and IR query processing?
- XML indexing, views, etc.