An introduction to XML

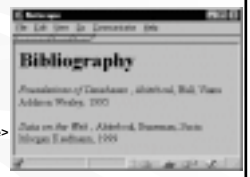
CPS 296.1

Topics in Database Systems

From HTML to XML

- HTML describes the presentation of the content

```
<h1>Bibliography</h1>
<p><i>Foundations of Databases</i>
Abiteboul, Hull, and Vianu
<br>Addison Wesley, 1995…
```



- XML describes only the content

```
<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

```
<bibliography>
  <book ISBN="10" price="80.00">
    <title>Foundations of Databases</title>
    <is_textbook/>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vianu</author>
    <publisher>Addison Wesley</publisher>
    <year>1995</year>
  </book>…
</bibliography>
```

- 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></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 < 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>…</section>…</subsection>

More XML features

- Comments: <!-- Comments here…-->
- CDATA: <![CDATA[Tags: <book>, …]]>
- ID's and references
  - <person id="o12"><name>Homer</name>…</person>
    <person id="o34"><name>Marge</name>…</person>
    <person id="o56" father="o12" mother="o34"><name>Bart</name></person>…
- Namespaces allow external schemas and qualified names
  - <book xmlns:myCitationStyle="http://…/mySchema">
    <myCitationStyle:title>…</myCitationStyle:title>
    <myCitationStyle:author>…</myCitationStyle:author>…
    </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.

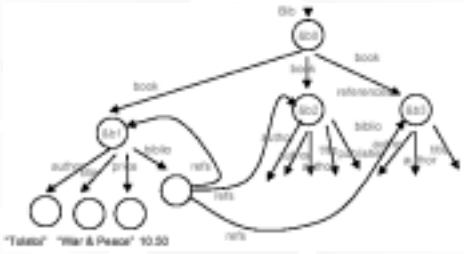
```
<? XML version="1.0"?>
<DOCTYPE book [
        <!ELEMENT book (title, author*, publisher?, section+)>
        <!ATTLIST book ISBN CDATA #REQUIRED>
        <!ATTLIST book year CDATA #IMPLIED>
        <!ELEMENT title (#PCDATA)>
        <!ELEMENT author (#PCDATA)>
        <!ELEMENT section (#PCDATA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


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

Semistructured data model of Lore

- Graph-based, unordered, edge-labeled



Ordered tree model of YAT

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

Data model of XML Query WG

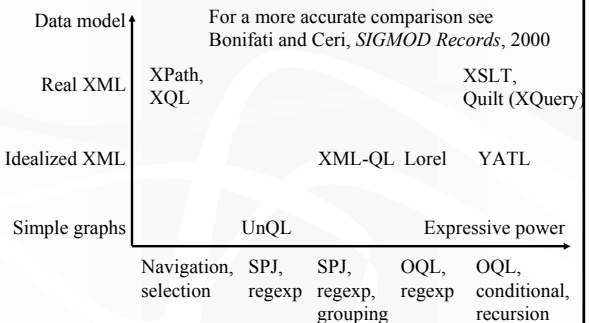
- Conceptually, also tree-based, ordered, and with references
- Functional notation (think ML); no explicit data structure
- Example

```
<book price="10.50">
  <title>…</title><author>…</author><author>…</author>…
</book>
```

→

e_1 = elemNode(qnameValue(null, "book"), { a_1 }, [e_2, e_3, e_3])
a_1 = attrNode(qnameValue(null, "price"), decimalValue(10.50))
e_2 = elemNode(qnameValue(null, "title"), …)
e_3 = elemNode(qnameValue(null, "author"), …)
e_4 = elemNode(qnameValue(null, "author"), …)

Query languages for XML



XML-QL

- Deutsch, Fernandez, Florescu, Levy, and Suciu. "XML-QL: A Query Language for XML." *WWW*, 1999
- 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

Scan bib.xml: match the pattern to obtain all ($y, $isbn, $t) bindings

Select those that pass the predicate

Construct output for each ($y, $isbn, $t) binding obtained in WHERE

```
WHERE
  <bib>
    <book year=$y ISBN=$isbn>
      <title>$t</title>
      <author><lastname>Abiteboul</lastname></author>
    </book>
  </bib> IN "bib.xml",
  $y < 2000
CONSTRUCT
  <resultBook ISBN=$isbn>
    <resultTitle>$t</resultTitle>
  </resultBook>
```


XML-QL: joins

- Retrieve all reviews for the books written by Abiteboul

```
WHERE
  <bib>
    <book ISBN=$isbn>
      <author><lastname>Abiteboul</lastname></author>
    </book>
  </bib> IN "bib.xml",
  <reviews>
    <review ISBN=$isbn></review> ELEMENT_AS $e
  </reviews> in "reviews.xsl"
CONSTRUCT
  $e
```


XML-QL: outerjoins (slide 1)

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

```
WHERE
  <bib>
    <book ISBN=$isbn>
      <title>$t</title>
      <author><lastname>Abiteboul</lastname></author>
    </book>
  </bib> IN "bib.xml",
  <reviews>
    <review ISBN=$isbn></review> ELEMENT_AS $e
  </reviews> in "reviews.xsl"
CONSTRUCT
  <resultBookWithReview ISBN=$isbn>
    <title>$t</title>
    $e
  </resultBookWithReview>
```

If a book has no review, it will not be in the output

Book title appears multiple times if there are multiple reviews

What is wrong?

XML-QL: outerjoins (slide 2)

- Use nested queries with outerjoin semantics

Okay for subquery to return empty result

```
WHERE
  <bib>
    <book ISBN=$isbn>
      <title>$t</title>
      <author><lastname>Abiteboul</lastname></author>
    </book>
  </bib> IN "bib.xml",
CONSTRUCT
  <resultBook ISBN=$isbn>
    <title>$t</title>
    (WHERE
      <reviews>
        <review ISBN=$isbn></review> ELEMENT_AS $e
      </reviews> IN "reviews.xsl"
    CONSTRUCT
      $e)
  </resultBook>
```


XML-QL: regexp, metadata queries

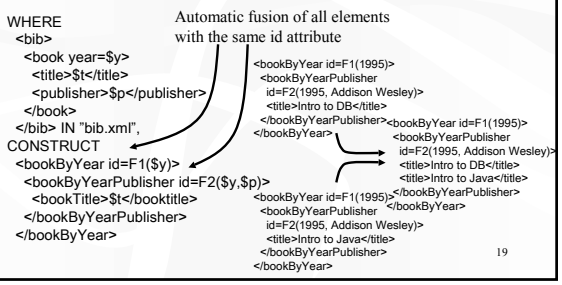
- What kind of elements are found in the content of the element corresponding to the book with ISBN of 10?

Regexp that matches any sequence of elements

```
WHERE
  <$*>
    <book ISBN=$isbn>
      <$tagname></>
    </book>
  </> IN "bib.xml",
CONSTRUCT
  <result>$tagname</result>
```

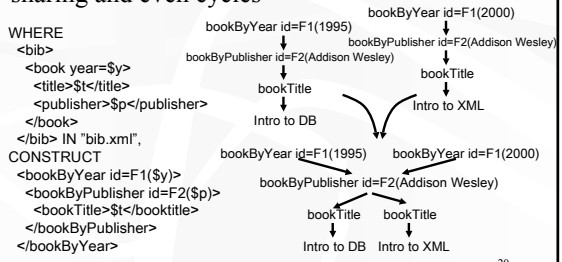
XML-QL: Skolem functions (slide 1)

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



XML-QL: Skolem functions (slide 2)

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



XML-QL: summary

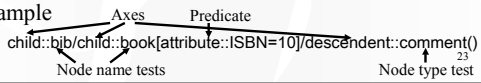
- Advantages
  - XML patterns look very familiar
  - Can express selection, projection, join, grouping
  - Can construct deeply nested XML elements
- Limitations
  - Problems with arbitrary use of Skolem functions
  - Preserving structure and hierarchy is difficult
  - No disjunction, aggregation, quantifiers, 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[./attribute::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), descendent (or self), following, following-sibling, preceding, preceding-sibling, namespace
- Node test
  - Name test (e.g., book, section, *)
  - Type test (e.g., text(), comment(), node())
- Example



XPath: abbreviated syntax

.	CN
..	CN/parent::*
book	CN/child::book
book/@ISBN	CN/child::book/attribute::ISBN
//section	ROOT/descendent-or-self::section
book//section	CN/child::book/descendent-or-self::node()/child::section
section[1]	CN/child::section[position()=1]

- Example
  - Verbose: child::bib/child::book[attribute::ISBN=10]/descendent::comment()
  - Abbreviated: bib/book[@ISBN=10]//comment()
- Tricky example
  - bib//section[1] is not the same as bib/descendent::section[1]

XQuery

- Chamberlin, Robie, and Florescu. "Quilt: An XML Query Language for Heterogeneous Data Sources." *WebDB*, 2000
- 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: path expressions

- Based on XPath
  - Example: find all references to the book with ISBN 20 in the book with ISBN 10

```
document("bib.xml")/book[@ISBN=10]//
        bookref[@ISBN=20]
```

- Extended with a de-reference operator (among other things)
  - Example: find all references to books written by Abiteboul in the book with ISBN 10

```
document("bib.xml")/book[@ISBN=10]//
        bookref[@ISBN=>book/author="Abiteboul"]
```


XQuery: FLWR expressions

- Retrieve the titles of books written by Abiteboul before 2000, together with their inventory

```
FOR     $b IN document("bib.xml")//book[@year<2000]
LET     $i := document("inventory.xml")//inventory[@ISBN=$b/@ISBN]
WHERE $b/author/lastname="Abiteboul"
RETURN <resultBook ISBN=$b/@ISBN>
            <title>$b/title/text()</title>
            <inventory>$i/text()</inventory>
        </resultBook>
```

Or,

```
FOR     $b IN document("bib.xml")//book[@year<2000]
        $i IN document("inventory.xml")//inventory
WHERE $b/author/lastname="Abiteboul" AND $b/@ISBN=$i/@ISBN
RETURN <resultBook ISBN=$b/@ISBN>
          <title>$b/title/text()</title>
          <inventory>$i/text()</inventory>
       </resultBook>
```


XQuery: quantified expressions

- Find titles of books in which XML is mentioned in the some section

```
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

```
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

```
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>
```

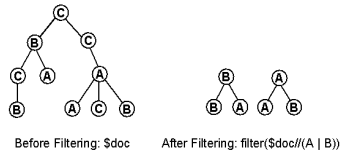

XQuery: conditional, sorting

- Make a list of books ordered by their titles; for journals, include the editors, and for others, include the authors

```
FOR     $b IN //book
RETURN <book>
          <title>$b/title</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



- Generate a table of contents

```
FILTER (document("XMLBook.xml")//(section | section/title | section/title/text()))
```

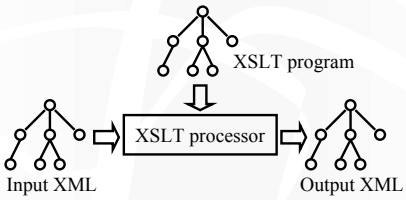

XQuery: summary

- 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



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

```
<xsl:template match="book[@year='1995'][title]">
  <resultTitle><xsl:value-of select="title"/></resultTitle>
</xsl:template>
```

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

```
<xsl:template match="title[parent::book]">
  <i><xsl:value-of select="."/></i>
</xsl:template>
```

<xsl:template> instantiation

```
<xsl:template match="book[@year='1995'][title]">
  <resultTitle><xsl:value-of select="title"/></resultTitle>
</xsl:template>
```

```
<book ISBN="10" year="1995">
 <title>Intro to DB</title>
 <author>…</author>
 <section>A DBMS is a …</section>
 <section>Relational algebra …</section>
 <section>Query optimization …</section>
</book>
```

```
<book ISBN="11" year="2000">
 <title>Intro to XML</title>
 <author>…</author>
 <section>XML has emerged …</section>
 <section>XQuery is a …</section>
</book>
```

Input XML

Rule applies:

```
<resultTitle>Intro to DB</resultTitle>
```

Rule does not apply; default behavior:

```
Intro to XML
…
XML has emerged …
XQuery is a …
```

Output XML

<xsl:apply-templates>

```
<xsl:template match="*|/">
  <xsl:apply-templates select="./node()"/>
</xsl:template>
```

→ Implicit built-in rule: recursively apply templates to child elements

```
<xsl:template match="@*|text()">
  <xsl:value-of select="."/>
</xsl:template>
```

→ Implicit built-in rule: for attribute or text nodes, just print out the string value

```
<xsl:template match="section">
  <h2>Section <xsl:value-of select="@number"/></h2>
  <xsl:value-of select="."/>
</xsl:template>
```

→ User rule: for a section, output its section number and then content

```
<xsl:template match="book">
  <h1><xsl:value-of select="title"/></h1>
  <xsl:apply-template select="section"/>
</xsl:template>
```

→ User rule: for a books, output its title then format its sections

collection
- book
  - title
  - author
  - section
  - section
- CD
  - title
  - artist
  - track
- book
- …

Note that the last three rules override the first

Another XSLT example

- Table of contents again

```
<xsl:template match="/">
 <xsl:apply-templates select="//book"/>
</xsl:template>

<xsl:template match="book">
  <h1><xsl:value-of select="title"/></h1>
  <ol><xsl:apply-templates select="section"/></ol>
</xsl:template>

<xsl:template match="section">
  <li>Section <xsl:value-of select="@number"/>
  <xsl:if test="section">
    <ol><xsl:apply-templates select="section"/></ol>
  </xsl:if>
  </li>
</xsl: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
  - Where? Files? Relational database? Object-oriented database? Special-purpose XML database?
- XML query processing
  - Again, where? How does it differ from traditional database and IR query processing?
- XML indexing, views, etc.