“The reason that so many people are excited about XML is that so many people are excited about XML.”

XML Background

- **eXtensible Markup Language**
- **Roots are HTML and SGML**
  - HTML mixes formatting and semantics
  - SGML is cumbersome
- **XML is focused on content**
  - Designers (or others) can create their own sets of tags.
  - These tag definitions can be exchanged and shared among various groups (DTDs, XSchema).
  - XSL is a companion language to specify presentation.
- **<Opinion> XML is ugly </Opinion>**
  - Intended to be generated and consumed by applications --- not people!

From HTML to XML

- HTML describes the presentation
- XML describes the content

Example in XML

```
<bibliography>
  <book>
    <title> Foundations of Databases </title>
    <author> Abiteboul </author>
    <year> 1995 </year>
  </book>
  ...
</bibliography>
```

XML as a Wire Format

- People quickly figured out that XML is a convenient way to exchange data among applications.
  - E.g. Ford’s purchasing app generates a purchase order in XML format, e-mails it to a billing app at Firestone.
  - Firestone’s billing app ingests the email, generates a bill in XML format, and e-mails it to Ford’s bank.
- Emerging standards to get the “e-mail” out of the picture: SOAP, WSDL, UDDI...
  - The basis of “Web Services” --- potential impact is tremendous.
- **Why is it catching on?**
  - It’s just text, so...
    - Platform, Language, Vendor agnostic
    - Easy to understand, manipulate and extend.
    - Compare this to data trapped in an RDBMS.
What’s this got to do with Databases?

• Given that apps will communicate by exchanging XML data, then databases must at least be able to:
  - Ingest XML formatted data
  - Publish their own data in XML format

• Thinking a bit harder:
  - XML is kind of a data model.
  - Why convert to/from relational if everyone wants XML?

• More cosmically:
  - Like evolution from spoken language to written language!

• The (multi-) Billion Dollar Question:
  - Will people really want to store XML data directly?
  - Current opinion: All major vendors say Yes, or at least, “Maybe”

Another (partial) Example

```
<Invoice>
  <Buyer>
    <Name> ABC Corp. </Name>
    <Address> 123 ABC Way </Address>
  </Buyer>
  <Seller>
    <Name> Goods Inc. </Name>
    <Address> 17 Main St. </Address>
  </Seller>
  <ItemList>
    <Item> widget </Item>
    <Item> thingy </Item>
    <Item> jobber </Item>
  </ItemList>
</Invoice>
```

Can View XML Document as a Tree

```
Invoice as a tree

Invoice
  |--- Buyer
  |     |--- Name
  |     |--- Address
  |--- Seller
  |     |--- Name
  |     |--- Address
  |--- Itemlist
  |     |--- Item
  |     |--- Item
```

Mapping to Relational

• Relational systems handle highly structured data

```
...               ...               ...
article NULL 0
author NULL 1
book 1 E.F. Codd NULL
pages 2 377-387 NULL
...               ...               ...
```

New splinters from XML

• Very expensive to store variable document types

• Difficult to search trees that are broken into tables

Mapping to Relational I

• Question: What is a relational schema for storing XML data?
• Answer – Depends on how “Structured” it is...
• If unstructured - use an “Edge Map”
Mapping to Relational II

- Can leverage Schema (or DTD) information to create relational schema.
- Sometimes called “shredding”
- For semi-structured data use hybrid with edge map for overflow.

STORED table
(author, year, journal, ...)

Overflow buckets

Document Type Definitions (DTDs)

- Grammar for describing the allowed structure of XML Documents.
- Specify what elements can appear and in what order, nesting, etc.
- DTDs are optional (!)
- Many “standard” DTDs have been developed for all sorts of industries, groups, etc.
- e.g. NITF for news article dissemination
- DTDs are being replaced by XSchema (more in a moment)

XML Schemas, etc.

- XML Documents can be described using XSchema
  - Has a notion of types and typechecking
  - Introduces some notions of IC’s
  - Quite complicated, controversial ... But will replace simpler DTDs
- XML Namespaces
  - Can import tag names from others
  - Disambiguate by prefixing the namespace name
    - i.e. use:price is different from eurozone:price

Other XML features

- Elements can have “attributes” (not clear why).
  
  \(<Price currency="USD">1.50</Price>\)

- XML docs can have IDs and IDREFs, URIs
  - reference to another document or document element
- Two APIs for interacting with/ parsing XML Docs:
  - Document Object Model (DOM)
    - A tree “object” API for traversing an XML doc
    - Typically for Java
  - SAX
    - Event-Driven: Fire an event for each tag encountered during parse.
    - May not need to parse the entire document.

Queries XML

- Xpath
  - A single-document language for “path expressions”
- XSLT
  - XPath plus a language for formatting output
- XQuery
  - An SQL-like proposal with XPath as a sub-language
  - Supports aggregates, duplicates, ...
  - Data model is lists, not sets
  - “reference implementations” have appeared, but language is still not widely accepted.
- SQL/XML
  - the SQL standards community fights back

DTD Example (partial)

```
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY % datetime.tz "CDATA">
<!ENTITY % string "CDATA">
<!ENTITY % nmtoken "CDATA">  <!-- Any combo of XML name chars. -->
<!ENTITY % xmlLangCode "%nmtoken;">  <!-- Any combo of XML name chars. -->
<!ELEMENT SupplierID (#PCDATA)>
<!ATTLIST SupplierID
  domain %string;  #REQUIRED
>
<!ELEMENT Comments (#PCDATA)>
<!ELEMENT ItemSegment (ContractItem+)>
<!ATTLIST ItemSegment
  segmentKey %string;  #IMPLIED
>
<!ELEMENT Contract (SupplierID+, Comments?, ItemSegment+)>
<!ATTLIST Contract
  effectiveDate %datetime.tz;  #REQUIRED
  expirationDate %datetime.tz;  #REQUIRED
>```

Here’s a DTD for a Contract

Elements contain others:

<table>
<thead>
<tr>
<th>Characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>? = 0 or 1</td>
</tr>
<tr>
<td>* = 0 or more</td>
</tr>
<tr>
<td>+ = 1 or more</td>
</tr>
</tbody>
</table>
XPath

- Syntax for tree navigation and node selection
  - Navigation is defined by "paths"
  - Used by other standards: XSLT, XQuery, XPointer, XLink
- / = root node or separator between steps in path
- * matches any one element name
- @ references attributes of the current node
- // references any descendant of the current node
- [] allows specification of a filter (predicate) at a step
- [n] picks the nth occurrence from a list of elements.

The fun part: Filters can themselves contain paths

XPath Examples

- Parent/ Child (’) and Ancestor/ Descendant (“//”):
  /catalog/product//msrp
- Wildcards (match any single element):
  /catalog/*/msrp
- Element Node Filters to further refine the nodes:
  - Filters can contain nested path expressions
  //product[price/msrp < 300]/name
  //product[price/msrp < /dept/@budget]/name
  - Note, this last one is a kind of "join"

XQuery

<result>
  FOR $x in /bib/book
  WHERE $x/year > 1995
  RETURN <newtitle>
    $x/title
  </newtitle>
</result>

Main Construct (replaces SELECT-FROM-WHERE):

- FLWR Expression: FOR-LET-WHERE-RETURN

<table>
<thead>
<tr>
<th>FOR/LET Clauses</th>
<th>Ordered List of tuples</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHERE Clause</td>
<td>Filtered list of tuples</td>
</tr>
<tr>
<td>RETURN Clause</td>
<td>XML data: Instance of Xquery data model</td>
</tr>
</tbody>
</table>

XQuery

<big_publishers>
  FOR $p IN distinct(document("bib.xml")//publisher)
  LET $b := document("bib.xml")/book[publisher = $p]
  WHERE count($b) > 100
  RETURN $p
</big_publishers>

distinct = a function that eliminates duplicates
count = a (aggregate) function that returns the number of elms

XQuery

- 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 subexpressions and for aggregations
Nested Queries

- Invert the hierarchy from publishers inside books to books inside publishers

```xml
FOR $p$ IN distinct(//publisher)
RETURN <publisher name={$p/text}>
  {FOR $b$ IN //book[publisher = $p]
    RETURN <book>
      {$b/title}
      {$b/price}
    </book>
  }
</publisher>
```

Operators Based on Global Ordering

```xml
expr1 {BEFORE}
AFTER
expr2
```

- Returns nodes in expr1 that are before (after) nodes in expr2
- Find procedures where no anesthesia occurs before the first incision

```xml
FOR $proc$ IN //section[title = "Procedure"]
WHERE empty($proc//anesthesia BEFORE ($proc//incision)[1])
RETURN $proc
```

Advantages of XML vs. Relational

- ASCII makes things easy
  - Easy to parse
  - Easy to ship (e.g., across firewall, via email, etc.)
- Self-documenting
  - Metadata (tag names) come with the data
- Nested
  - Can bundle lots of related data into one message
  - (Note: object-relational allows this)
- Can be sloppy
  - don't have to define a schema in advance
- Standard
  - Lots of free Java tools for parsing and merging XML
    - Expect lots of Microsoft tools (C#) for same
  - Tremendous Momentum!

What XML does not solve

- XML doesn't standardize metadata
  - It only standardizes the metadata language
    - Not that much better than agreeing on an alphabet
    - E.g., my <price> tag vs. your <price> tag
      - Mine includes shipping and federal tax, and is in $US
      - Yours is manufacturer's list price in ¥Japan
    - XML Schema is a proposal to help with some of this
- XML doesn't help with data modeling
  - No notions of IC’s, FD’s, etc.
  - In fact, encourages non-first-normal-form!
- You will probably have to translate to/from XML (at least in the short term)
  - Relational vendors will help with this ASAP
  - XML “features” (nesting, ordering, etc.) make this a pain
  - Flatten the XML if you want data independence (?)

Reminder: Benefits of Relational

- Data independence buys you:
  - Evolution of storage -- vs. XML?
  - Evolution of schema (via views) -- vs. XML?
- Database design theory
  - IC’s, dependency theory, lots of nice tools for ER
- Remember, databases are long-lived and reused
  - Today's “nesting” might need to be inverted tomorrow!
- Issues:
  - XML is good for transient data (e.g., messages)
  - XML is fine for data that will not get reused in a different way (e.g., Shakespeare, database output like reports)
  - Relational is far cleaner for persistent data (we learned this with OODBs)
- Will benefits of XML outweigh these issues??????

More on XML

- 100s of books published
- Each seems to be 1000 pages
- Try some websites
  - xml.org provides a business software view of XML
  - xml.apache.org has lots of useful shareware for XML
  - www.ibm.com/developerworks/xml/ has shareware, tutorials, reference info
  - xml.com is the O’Reilly resource site
  - www.w3.org/XML/ is the official XML standard site
  - the most standardized XML dialects are:
    - Ariba’s Commerce XML (“cxml”, see cxml.org)
    - RosettaNet (see rosettanet.org)
    - Microsoft trying to enter this arena (BizTalk, now .NET)