Terminology reminder

- Data model: general conceptual way of structuring data
- Schema: structure of a particular database under a certain data model
- Instance: actual data conforming to a schema

What is XML (Extensible Markup Language)?

- **HTML was hugely successful due to**
  - simplicity -> can be learned easily
  - text based -> can be edited by any text editor. No need for a special tool
- **But, HTML is mainly for human consumption**
  - HTML tags are for formatting, not for meaning
  - e.g., `<table>`, `<ul>`, etc.

- **XML: data representation standard with "semantic" tag**

  - Write a simple XML example in a text editor
    ```xml
    <?xml version="1.0"?>
    <Book Edition="1">
      <Title>Database systems</Title>
      <Author>Hector Garcia-Molina</Author>
      <Price>$100</Price>
    </Book>
    ```

  - XML consists of three things:
    1. Tagged elements, which may be nested within one another
    2. Attributes on elements
    3. Text

  - Well-formed XML should have
    - single root element
    - matching tags
    - unique attribute name

  - XML DOM (Document Object Model): Tree-based model of XML data
    - To manipulate XML data, we need to build an abstract model based on XML text
    - Existing XML parsers read XML data file and create XML DOM tree
      - including JAXP you will use for project 2
    - one XML tag element becomes one node in the DOM and becomes a child node of its parent
    - a node may be associated with name and value
      - element name becomes the node name. element node does not have any value
    - any text inside an XML element creates a separate child "text node"
      - e.g., `<Price>$100</Price>` creates two nodes:
        - "Price" element node and its child text node of "$100"
• note: According to W3C DOM standard, a text node is created for empty white-space or new lines.
  • text node does not have a name but has text string as its value
    * any attribute of an element creates an "attribute" node
      • attribute node is not a child node
    • attribute name becomes the name of the node and attribute value becomes its value
      * each node is of particular type and may have name and value
        • type: element, text, attribute, comment, ...
        • name: Book, Title, Author, Edition, ...
        • value: Database system, 1, $100...
    * terminology: parent, children, descendant, ancestor

  – a (loose) superset of HTML, a (loose) subset of SGML
  – some say that XML is to data what Java is to programming.
  – text and tag based representation makes it
    * easier to understand
    * more widely compatible
    * more flexible with its data schema
  – de facto standard for data exchange
  – Take note:
    * XML can be clunky.
    * The full specification is enormous, but the basic idea is simple.

• Q: Is that it? What is there to "learn" about XML? What are the practical problems due to the use of XML?

• Issues
  – Q: XML is about data representation and exchange. How do we actually store, manage and query xml data?
    * A: Possible alternatives
      1. transform to RDB: XML -> RDB

  • Q: there is mismatch of data model. How to convert one to another?

  2. store data in XML: native XML data
  • Q: how should we store XML?

  – Q: what xml design is "good"? XML normal form?

  – Q: how to query xml?
    * XPath, XQuery
  – Q: how to specify schema?
* DTD, XML schema

- Q: How do we translate one XML data to another? How do we format it for presentation?
  * XSL (XML stylesheet language)
    + XSLT (XSL transformation), XSL-FO (XSL formatting objects)
- Q: anyway to avoid name conflict?
  * XML namespace
XML Namespaces

- A way to avoid name conflict
- XML Namespace allows specifying what we truly mean by a tag

**e.g.**) example without namespace and explain default namespace specification

```
<?xml version="1.0"?>
<Book Edition="1" xmlns="http://oak.cs.ucla.edu/cs144/">
  <Title>Database systems</Title>
  <Author>Hector Garcia-Molina</Author>
  <Price>$100</Price>
</Book>
```

- Note: The Namespace URL does not have to point to any real page.
  The URL is just the unique identifier of the namespace.

* Q: What namespace does element Title belong to?

* Q: What namespace does attribute Edition belong to?

  * Note: The default namespace does not apply to attributes. Unprefixed attributes belong to no namespace.

* Q: Is it possible to use different namespace for different elements?

**Price: [http://xml.com/shopping/](http://xml.com/shopping/)**

```
<?xml version="1.0"?>
<Book c:Edition="1" xmlns="http://oak.cs.ucla.edu/cs144/"
     xmlns:s="http://xml.com/shopping"
     xmlns:c="http://oak.cs.ucla.edu/cs144">
  <Title>Database systems</Title>
  <Author>Hector Garcia-Molina</Author>
  <s:Price>$100</s:Price>
</Book>
```

* Q: Do E1 and E2 belong to the same namespace?

```
<a:E1 xmlns:a="http://a.com/">
  <b:E2 xmlns:b="http://a.com/">
```

DTD (Document Type Definition)

Example

```xml
<?xml version="1.0"?>
<Bookstore>
  <Book ISBN="0130353000" Price="$65" Ed="2nd">
    <Title>First Course in Database Systems</Title>
    <Author>
      <First_Name>Jeffrey</First_Name>
      <Last_Name>Ullman</Last_Name>
    </Author>
  </Book>
  <Book ISBN="0130319953" Price="$75">
    <Title>Database Systems: Complete Book</Title>
    <Author>Hector Garcia-Molina</Author>
    <Author>
      <First_Name>Jeffrey</First_Name>
      <Last_Name>Ullman</Last_Name>
    </Author>
    <Remark>It's a great deal!</Remark>
  </Book>
</Bookstore>
```

- **Q: What can we say about the structure of the data?**
  - Q: Does Book element always have a title?
  - Q: Is it okay for a book to have multiple remarks?

- **DTD:**
  - a grammar that describes the legal attributes of elements and the legal ordering and nesting of the elements.
  - one way to describe the "schema" of an XML data instance

```
<!ELEMENT Bookstore (Book*)>
<!ELEMENT Book (Title, Author+, Remark?)>
<!ATTLIST Book ISBN CDATA #REQUIRED
```
<!DOCTYPE root-element [element declaration]> or
<!DOCTYPE root-element SYSTEM "example.dtd">

* **Note:** three important keywords: ELEMENT, ATTLIST and (#P)CDATA

  - `<!ELEMENT element-name (element-content)>`
  - `<!ATTLIST element-name attr-name attr-name attr-type default-value>`

    attr-type: CDATA, "0"|"1"|"2", ID, IDREF(S), ...
    default-value: value, #REQUIRED, #IMPLIED (=optional), ...

* The DTD is specified at the top of the document or in a separate file

* Some notes on (#P)CDATA: the details are messy, but overall,

  - #PCDATA is parsed, while CDATA is not. #PCDATA is for element content, CDATA is for attribute types. You cannot use them interchangeably.

  - Recommendation: Use CDATA for string-valued attributes, use #PCDATA for elements containing text.

  - If you want an element to contain a mixture of text and other elements, do so by specifying the element types along with #PCDATA in a 0-or-more list, e.g., (#PCDATA | Author | Editor)*.

* **Q:** What are the benefits of using a DTD?

* **Q:** Is there a benefit of not using a DTD?
  - Specifying keys and references in DTD

* **Q:** Any concern/issue/problem with the above XML data?

* **NOTE:**
  - redundancy problem for Author sub-element
  - Can we separate author out and add pointers to the authors?
* Q: How can we specify that Ident is a unique key and Authors are references to the key?

* ID and IDREF(S) Attributes
  • Element pointers: assign a special ID attribute to an element, then point to that element with a special IDREF or IDREFS attribute in another element.
  
  • IDREFS: each IDREF is separated by whitespace

* DTD for the above data:

```xml
<!ELEMENT Bookstore (Book*, Author*)>
<!ELEMENT Book (Title, Remark?)>
<!ATTLIST Book ISBN ID #REQUIRED
  Price CDATA #REQUIRED
  Edition CDATA #IMPLIED
  Authors IDREFS #REQUIRED>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Remark (#PCDATA)>
<!ELEMENT Author (#PCDATA | (First_Name, Last_Name))>
<!ATTLIST Author Ident ID #REQUIRED>
<!ELEMENT First_Name (#PCDATA)>
<!ELEMENT Last_Name (#PCDATA)>
```

* Q: What is a possible implication of ID and IDREFS on data model?

* Q: What should be an attribute vs. an element?
XML Schema

- **Schema definition written in XML** (often used with extension .xsd (XML Schema Definition))
- Enclosed in `<schema> ... </schema>` under namespace [http://www.w3.org/2001/XMLSchema](http://www.w3.org/2001/XMLSchema) for both xml elements and types

- Simple type: element with no children or attributes
  `<element name="..." type="..."/>`, where type can be string, integer, decimal, float, boolean, ...

  e.g., Simple XML schema example
  
  - XML -
    ```xml
    <?xml version="1.0" ?>
    <Book>Web Applications</Book>
    ```
  
  - XML schema -
    ```xml
    <?xml version="1.0" ?>
    <schema xmlns="http://www.w3.org/2001/XMLSchema"
            xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <element name="Book" type="xs:string" />
    </schema>
    ```

- Complex type: element with child elements or attributes

  ```xml
  <element name="...">
    <complexType>
      <sequence>
        <element ... />
        ...
      </sequence>
      <attribute ... />
    </complexType>
  </element>
  ```

  e.g., DTD and equivalent XML schema
  ```xml
  <!ELEMENT Book (Title, Author+, Remark?)>
  <!ELEMENT Title (#PCDATA)>
  <!ELEMENT Remark (#PCDATA)>
  <!ELEMENT Author (#PCDATA)>
  <!ATTLIST Book ISBN CDATA #REQUIRED
            Edition CDATA #IMPLIED>
  ```

  ```xml
  <?xml version="1.0"?>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://oak.cs.ucla.edu/cs144">
    <xs:element name="Book">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Title" type="xs:string"/>
          <xs:element name="Author" type="xs:string" minOccurs="1" maxOccurs="unbounded"/>
          <xs:element name="Remark" type="xs:string" minOccurs="0" maxOccurs="1"/>
        </xs:sequence>
        <xs:attribute name="ISBN" type="xs:string" use="required"/>
        <xs:attribute name="Edition" type="xs:string"/>
      </xs:complexType>
    </xs:element>
  </xs:schema>
  ```
Note:
- `<sequence>` honors ordering, `<all>` or `<choice>` does not
- minOccurs and maxOccurs are occurrence indicators
- “use” attribute has default value “optional”
- targetNamespace specifies the namespace of defined elements

- For key specification, ID and IDREF(S) possible for attribute types, but key/keyref should be preferred

e.g.,

```
<Bookstore>
  <Review isbn="103">Great</Review>
</Bookstore>
```

ISBN is unique among all books
Review’s isbn attribute is a foreign key to Book ISBN

```
<xs:key name="key-isbn">
  <xs:selector xpath="/Bookstore/Book"/>
  <xs:field xpath="ISBN"/>
</xs:key>
<xs:keyref name="foreignkey-isbn" refer="key-isbn">
  <xs:selector xpath="/Bookstore/Review"/>
  <xs:field xpath="@key-isbn"/>
</xs:keyref>
```

- Q: DTD vs. XML Schema. Which one is more expressive?
  - Q: Can everything specified in DTD be specified in XML Schema?

  - Q: Can everything specified in XML Schema be specified in DTD?

- Q: Why use DTD?
XPath

Example

```xml
<AAA>
  <BBB aaa="111" bbb="222">
    <CCC/>
    <CCC aaa="333" bbb="444" ccc="555" />
  </BBB>
  <BBB aaa="666">
    <CCC aaa="777"/>
    <DDD aaa="888">
      <CCC aaa="999">35</CCC>
    </DDD>
  </BBB>
  <BBB aaa="AAA">
    <DDD aaa="BBB"/>
  </BBB>
</AAA>
```

- XPath: simple "path expression" that matches XML data by navigating down (and occasionally up or across) the tree and possibly evaluating conditions over data in the tree.

- NOTE:
  - XPath tree is not identical to XML DOM
  - The value of an element node is the concatenation of its all descendent values

- XPath examples at XPath Lab [http://oak.cs.ucla.edu/cs144/examples/xpath.html](http://oak.cs.ucla.edu/cs144/examples/xpath.html)

  - `/AAA` : root element AAA
    * `/` at the beginning means starting from the root
  - `/AAA/BBB` : all BBB elements that are children of root element AAA
    * Q: How many elements are selected for `/AAA/BBB`?

  - `//CCC` : all CCC elements regardless of the path
    * `//` means any descendant
    * Q: How many elements are selected by `//CCC`?

    * Q: `/AAA//CCC` : how many elements?

  - `/AAA/BBB/*` : all elements that are children of `/AAA/BBB`
    * Q: How many elements?

  - `/AAA/BBB/@aaa` : attributes aaa of elements `/AAA/BBB`
    * `@` means attribute
    * Q: How many attributes?
selection conditions can be specified in []
  * /AAA/BBB[1]: first element /AAA/BBB
    • Note: index starts at 1 not 0.

  * /AAA/BBB[last()]: last element /AAA/BBB

  * /AAA/BBB[@aaa]: all elements /AAA/BBB that have attribute aaa
    • Q: How many elements?

  * /AAA/BBB[CCC]: all /AAA/BBB elements that have CCC as a child
    • Q: How many elements?

  * /AAA/BBB[@xxx='111']: BBB elements with attribute value xxx='111'

  * Q: //*[CCC > 20]: what does it mean?

  * Q: //*[. > 20]: what does it mean?

  * Q: //DDD[./CCC > 20]/EEE: what does it mean?

  * //CCC | //BBB: All CCC or BBB elements

    • | means "union"

Again, XPath tree is not identical to XML DOM
  * Mostly, assume that a text node is not a child of its element node, but a value of the element
    • except for child::node() which returns text node as a child
    • /AAA/BBB/DDD/CCC/* won't match any node in the above example
  * The official standard XPath specification is confusing regarding parent-child relationship of nodes, so for practical purpose, ignore this part of the standard
  * Source of great confusion and surprises. Do not try to be smart with XPath. use straightforward XPath expressions
XML to Relation Mapping

- **How to store XML data to a relational database?**
  
  - There really is no "right" answer. Still an active area of research

**Example 1**

```
<!ELEMENT Book (Title, Author, Remark)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (First_Name, Last_Name, Bio)>
<!ELEMENT First_Name (#PCDATA)>
<!ELEMENT Last_Name (#PCDATA)>
<!ELEMENT Bio (#PCDATA)>
<!ELEMENT Remark (#PCDATA)>
<!ATTLIST Book ISBN CDATA #REQUIRED
    Price CDATA #REQUIRED>
```

[tree diagram of the above DTD]

- Q: How should we convert the XML to a relation?

  * Choice 1: Store each Book element in a single column as a text
    - Q: Any potential problem with this approach?

  * Choice 2: Capture the exact XML DOM tree structure.
    1. unique id to each node.
    2. table for node id, name, type, and value
    3. table for parent-child relation
    4. table for element-attribute relation

    - Q: Potential benefit compared to Choice 1?

    - Q: Potential problem?

  * Choice 3: Try to capture domain data as opposed to XML structure
    - (Title, ISBN, Price, Remark, A_FN, A_LN, Bio)?
• Q: Potential benefit compared to the previous two choices?

• Q: Potential problem?
  
  - Q: Can we support XPath query on the original XML after this transformation?

• Q: Better table design? Is it in BCNF?
  
  - Note: nested child elements may force table split, particularly when the child elements have non-key attributes

**Example 2**

```xml
<!ELEMENT Book (Title, Author+, Remark*)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (First_Name, Last_Name, Bio)>
<!ELEMENT First_Name (#PCDATA)>
<!ELEMENT Last_Name (#PCDATA)>
<!ELEMENT Bio (#PCDATA)>
<!ELEMENT Remark (#PCDATA)>
<!ATTLIST Book ISBN CDATA #REQUIRED
  Price CDATA #OPTIONAL>
```

- Q: How to convert it under Choice 3? What to do with Authors?

- Q: What to do with Remark and Price?

- Remarks
  
  * ?, | may force table split or use NULL
  * *, + may also force table split