XML Databases

- HTML is adequate to represent the structure of documents for display purposes
- HTML is inadequate to represent the structure of data for database purposes
- An application cannot distinguish first and last names, based on HTML tags
- XML (eXtensible Markup Language) was developed
- HTML has a **fixed** set of tags.
- XML allows the user to define **new collections** of tags
- These new tags can be used to structure any type of data that we need to transmit (i.e. over the Web)

- XML was developed by a W3C WG
- It provides a **bridge** between the document-oriented view of data (HTML) and the schema-oriented view of data (DBMS)
- How do we encode the display (in a Web browser) of the new XML user-defined tags?
- **XSL** (eXtensible Style Language) is a way of describing how an XML document should be displayed

Introduction to XML

- **Elements**: (alias **tags**) are the primary building blocks of an XML document.
- The start of the content of an element ELM is marked with <ELM> (**start tag**)
- The end of the content of an element ELM is marked with </ELM> (**end tag**)
- XML elements must be properly nested and are case sensitive
- **Example:**

```
<BOOK>
<AUTHOR>
<FNAME> John </FNAME>
<LNAME> Smith </LNAME>
</AUTHOR>
</BOOK>
```

Introduction to XML

- **Attributes:** An element can have descriptive attributes that provide additional information about it
- The values of attributes (enclosed in quotes) are set inside the start tag of an element <ELM attrib="value">
- **Entity references:** shortcuts for portions of common text or the content of external files. start: & end: ;
- Whenever they appear in XML documents, they are **textually replaced** by their content
- 5 predefined XML entity references:
  < & > " '
- They are also used to insert arbitrary Unicode characters into the text.

Introduction to XML

- **Comments:** start with <!- end with ->
- **DTDs** (Document Type Declarations) sets of rules that allows the user to specify their own sets of elements, attributes and entities.
- A DTD is a **grammar** that indicates which tags are allowed, in what order they can appear and how they can be nested.
- We distinguish **two types** of XML documents:
- An XML document is called **valid**, if there is a DTD associated with it and the document is structured according to the rules of the DTD.

Introduction to XML

- An XML document is called **well-formed**, if it does not have a DTD, but follows 3 structural guidelines:

1) Starts with an XML declaration

2) There is a root element that contains all other elements

3) All elements are properly nested

```
<?XML version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE BOOKLIST SYSTEM "books.dtd">
<BOOKLIST>
<BOOK genre="Science" format="Hardcover">
  <AUTHOR>
    <FIRSTNAME>Richard</FIRSTNAME><LASTNAME>Feynman</LASTNAME>
  </AUTHOR>
  <TITLE>The Character of Physical Law</TITLE>
  <PUBLISHED>1980</PUBLISHED>
</BOOK>
<BOOK genre="Fiction">
  <AUTHOR>
    <FIRSTNAME>R.K.</FIRSTNAME><LASTNAME>Narayan</LASTNAME>
  </AUTHOR>
  <TITLE>Waiting for the Mahatma</TITLE>
  <PUBLISHED>1981</PUBLISHED>
</BOOK>
<BOOK genre="Fiction">
  <AUTHOR>
    <FIRSTNAME>R.K.</FIRSTNAME><LASTNAME>Narayan</LASTNAME>
  </AUTHOR>
  <TITLE>The English Teacher</TITLE>
  <PUBLISHED>1980</PUBLISHED>
</BOOK>
</BOOKLIST>
```

XML DTDs

- DTD format:
  <!DOCTYPE name [DTDdeclaration]>
  DTDdeclaration is a description of the rules

```
<!DOCTYPE BOOKLIST [
  <!ELEMENT BOOKLIST (BOOK)*>
  <!ELEMENT BOOK (AUTHOR,TITLE,PUBLISHED?)>
    <!ELEMENT AUTHOR (FIRSTNAME,LASTNAME)>
      <!ELEMENT FIRSTNAME (#PCDATA)>
      <!ELEMENT LASTNAME (#PCDATA)>
    <!ELEMENT TITLE (#PCDATA)>
    <!ELEMENT PUBLISHED (#PCDATA)>
  <!ATTLIST BOOK genre (Science|Fiction) #REQUIRED>
  <!ATTLIST BOOK format (Paperback|Hardcover) "Paperback">
]>
```

XML DTDs

- <!ELEMENT BOOKLIST (BOOK)*>
  the elm BOOKLIST consists of zero or more BOOK elements
- <!ELEMENT BOOKLIST (BOOK)+>

  the elm BOOKLIST consists of at least one BOOK elements
- <!ELEMENT BOOK (AUTHOR,TITLE,PUBLISHED?)>
  the elm BOOK contains 3 elms
  (? → optional elm, 0 or 1 occurrence)
- <!ELEMENT LASTNAME (#PCDATA)>
  the elm LASTNAME does not contain other elements, but contains text. PCDATA == **P**arsed **C**haracter **D**ata (leaf node)

XML DTDs

- Element type declaration syntax:
  <!ELEMENT (contentType)>
- 5 possible content types:
  (1) other elements
  (2) #PCDATA (parsed character data)
  (3) EMPTY
  (4) ANY
  (5) a **regular expression**:
  (list of, exp*, exp?, exp+, exp1|exp2)

XML DTDs

- Attributes of elements are declared **outside** of the element
- <!ATTLIST BOOK genre (Science|Fiction) #REQUIRED>
- genre is an attribute of the elm BOOK
- genre can take two values
- genre is a required attribute
- Attribute declaration syntax:
  <!ATTLIST elmName (attName attType default)+>

XML DTDs

- XML defines several possible attribute types: string types, enumerated types …
- <!ATTLIST BOOK edition CDATA "1">
- For enum. types, we list all possible values
- For enum. types, we can also have a **default** value (attribute value is set automatically)
- #REQUIRED is a default specification
- <!ATTLIST BOOK genre (Science|Fiction) "Science">

XML-QL: Querying XML data

- XML docs come with a lot of structure
- We can use a high-level language to exploit this structure to query XML data and retrieve the results.
- Informal examples of XML-QL:

```
WHERE <BOOK>
          <NAME>
              <LASTNAME> $l </LASTNAME>
          </NAME>
      </BOOK> IN "www.server.com/books.xml"
CONSTRUCT <RESNAME> $l </RESNAME>
```

XML-QL: Querying XML data

- XML-QL queries extract data from an XML document by specifying a pattern of markups
- We are interested in data nested inside a BOOK/NAME/LASTNAME elements
- For each part of the XML document that matches the structure specified by the query, the variable l is bound to the contents of the element LASTNAME
- Variable names as prefixed by the $ sign
- The result is an XML document:

```
<RESNAME> name1 </RESNAME>
<RESNAME> name2 </RESNAME>
```

XML-QL: Querying XML data

- Find the lnames/fnames of all authors who wrote a book that was published in 1980.

```
WHERE <BOOK> <NAME>
                <LASTNAME> $l </LASTNAME>
                <FIRSTNAME> $f </FIRSTNAME>
              </NAME>
              <PUBLISHED>1980</PUBLISHED>
            </BOOK> IN "www.server.com/books.xml"
CONSTRUCT  <RESNAME>
             <FIRST> $f </FIRST>
             <LAST> $l </LAST>
           </RESNAME>
```

XML-QL: Querying XML data

- For each year, find the last names of authors who wrote a book published in that year.

```
WHERE   <BOOK> $e </BOOK> IN "www.server.com/books.xml",
        <AUTHOR> $n </AUTHOR>,
        <PUBLISHED> $p </PUBLISHED> IN $e
CONSTRUCT <RESNAME>
            <PUBLISHED> $p </PUBLISHED>
            WHERE <LASTNAME> $l </LASTNAME>
                IN $n CONSTRUCT
                        <LASTNAME> $l </LASTNAME>
            </RESNAME>
```