COSC 304
Introduction to Database Systems

XML

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

XML

Extensible Markup Language (***XML***) is a markup language that allows for the description of data semantics.

◆ XML is a markup language for describing any type of data because the markup terms can be user-defined.

◆ XML is **case-sensitive** unlike HTML.

XML is a standard by the World Wide Web Consortium (W3C).

Page 2

Advantages of XML

Some advantages of XML:

◆ Simplicity
  ⇨ The XML standard is relatively short and simple.

◆ Open standard
  ⇨ Standardized by W3C to be vendor and platform independent.

◆ Extensibility
  ⇨ XML allows users to define their own tags.

◆ Separation of data and presentation
  ⇨ XML data may be presented to the user in multiple ways.

◆ Interoperability
  ⇨ By standardizing on tags, interoperation and integration of systems is simplified.

Page 3

XML Components

An XML document is a text document that contains markup in the form of ***tags***.

An XML document consists of:

◆ An XML *declaration line* indicating the XML version.

◆ *Elements* (or tags) called *markup*. Each element may contain free-text, attributes, or other nested elements.
  ⇨ Every XML document has a single root element.
  ⇨ Tags, as in HTML, are matched pairs, as <foo> … </foo>.
  ⇨ Closing tags are not needed if the element contains no data:
  ⇨ Tags may be nested.

◆ An *attribute* is a name-value pair declared inside an element.

◆ Comments

XML data is ordered by nature.

Page 4

XML Example

```
<?xml version = "1.0" encoding="UTF-8" standalone="no"?>
<?xml:stylesheet type="text/xsl" href="dept.xsl"?>
<!DOCTYPE root SYSTEM "dept.dtd">

<!-- Department/Employee data formatted in XML -->

<root>
  <Dept dno = "D1">
      <Emp eno="E7"><name>R. Davis</name></Emp>
      <Emp eno="E8"><name>J. Jones</name></Emp>
  </Dept>
  <Dept dno = "D2" mgr = "E7">
      <Emp eno="E6"><name>L. Chu</name></Emp>
      <Emp eno="E3"><name>A. Lee</name></Emp>
      <budget>350000</budget>
  </Dept>
</root>
```
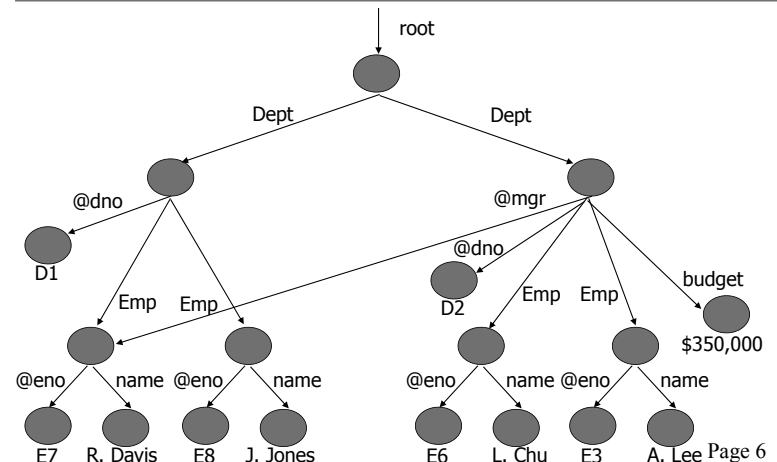
XML declaration
Stylesheet for presentation
External DTD for validation
Comment
Attribute
Element reference

Page 5

XML (tree view)

Page 6

Well-Formed and Valid XML Documents

An XML document is **well-formed** if it obeys the syntax of the XML standard. This includes:
- ◆ Having a single root element
- ◆ All elements must be properly closed and nested.

An XML document is **valid** if it is well-formed and it conforms to a Document Type Definition (DTD) or an XML Schema Definition (XSD).
- ◆ A document can be well-formed without being valid if it contains tags or nesting structures that are not allowed in its DTD/XSD.
- ◆ The DTD/XSD are schema definitions for an XML document.

XML Well-Formed Question

Question: How many of these two documents are well-formed?

1: `<x><a>Test</a><bT></bt></x>`

2: `<x>abc</x><y>def</y>`

A) 0
B) 1
C) 2

Namespaces

Namespaces allow tag names to be qualified to avoid naming conflicts. A naming conflict would occur when the same name is used by two different domains or vocabularies.

A namespace consists of two components:
- ◆ 1) A declaration of the namespace and its abbreviation.
- ◆ 2) Prefixing tag names with the namespace name to exactly define the tag's origin.

Namespaces Example

```
<?xml version = "1.0" encoding="UTF-8" standalone="no"?>
<root xmlns = "http://www.foo.com"          ←——— Default namespace
      xmlns:n1 = "http://www.abc.com">←— n1 namespace
  <Dept dno = "D1">
      <Emp eno="E7"><name>R. Davis</name></Emp>
      <Emp eno="E8"><name>J. Jones</name></Emp>
  </Dept>
  <Dept dno = "D2" mgr = "E7">
      <Emp eno="E6"><name>L. Chu</name></Emp>
      <Emp eno="E3"><name>A. Lee</name></Emp>
      <n1:budget>350000</budget>
  </Dept>                    budget is a XML tag in the
</root>                        n1 namespace.
```

Schemas for XML

Although an unrestricted XML format is useful to some applications, database data normally has some structure, even though that structure may not be as rigid as relational schemas.

It is valuable to define schemas for XML documents that restrict the format of those documents.

There are two main ways of specifying a schema for XML:
- ◆ Document Type Definition (DTD) (original, older)
- ◆ XML Schema

Document Type Definitions (DTDs)

A **Document Type Definition** (**DTD**) defines the grammatical rules for the document. It is not required for an XML document but provides a mechanism for checking a document's validity.

General DTD form:

```
<!DOCTYPE myroot [ <elements> ]>
```

name of root element in XML document

contents of DTD declares elements and attributes

Essentially, a DTD is a set of document rules expressed using EBNF (Extended Backus-Naur Form) grammar. The rules limit:
- ◆ the set of allowable element names, how elements can be nested, and the attributes of an element among other things

DTD Example

```
<!DOCTYPE root [
  <!ELEMENT root(Dept+)>
  <!ELEMENT Dept(Emp*, budget?)>
      <!ATTLIST Dept dno ID #REQUIRED>
      <!ATTLIST Dept mgr IDREF #IMPLIED>
  <!ELEMENT budget (#PCDATA)>
  <!ELEMENT Emp (name)>
      <!ATTLIST Emp eno ID #REQUIRED>
  <!ELEMENT name (#PCDATA)>
]>
```

+ means 1 or more times

* means 0 or more times

? means 0 or 1 time

Element reference (like a foreign key)

ID is a unique value that identifies the element

Parsed Character Data (atomic value)

DTD Elements

Each element declaration has the form:

```
<!ELEMENT <name> ( <components> )>
```

Notes:

◆ Each `!ELEMENT` declaration specifies that an element is being created.

◆ Components is either a list of one or more subelements or `#PCDATA`, or `EMPTY` (has no content).

◆ Each subelement may occur exactly once, zero or one time (`?`), zero or more (`*`), or one or more (`+`).

◆ An element that is a composite element (contains other elements) specifies the other elements in parentheses. These elements must appear in the document in the order specified.

DTD Attributes

Attributes are declared using:

```
<!ATTLIST <eltName> <attrName> <type> <value>)
```

Notes:

◆ Attribute names must be unique within an element.

◆ The type of attribute may be (among others):
  ⇨ `CDATA` - non-parsed string
  ⇨ `ID` - element identifier (`ID` valued attributes must start with a letter)
  ⇨ `IDREF` or `IDREFS` - one or more element references

◆ The value source may be:
  ⇨ `#IMPLIED` - attribute is optional
  ⇨ `#REQUIRED` - always present
  ⇨ `#FIXED "default value"`- declared default value

DTD Attribute Example

Values for an attribute may be specified:

```
<!ELEMENT Emp (name,salary)>
<!ATTLIST Emp title ("EE"|"SA"|"PR"|"ME") "EE">
<!ATTLIST Emp eno ID #REQUIRED>
<!ATTLIST Emp bdate CDATA #IMPLIED>
```

DTD `ID` and `IDREF`

An `ID` attribute is used to uniquely identify an element within a document. It is used to model keys.

◆ Note however that the scope of the uniqueness is the entire document not elements of the given type.

◆ An `ID` is like declaring named anchors in HTML.

◆ An ID must be a string starting with a letter and must be unique throughout the *whole* document.

`IDREF`/`IDREFS` are attributes that allow an element to refer to another element in the document.

◆ An `IDREF` attribute contains a single value that references a named location in the document (an existing `ID` value).

◆ An `IDREFS` attribute contains a list of `ID` references.

◆ Using `IDREF` allows the structure of an XML document to be a graph, rather than just a tree.

DTD Choice

The symbol "`|`" can connect alternative sequences of tags.

For example, a name may have an optional title, a first name, and a last name, in that order, or is a full name:

```
<!ELEMENT NAME (
  (TITLE?, FIRST, LAST) | FULLNAME
)>
```

Using DTDs in an XML Document

A DTD can be either embedded in the XML document itself or specified as an external file.

Embedding:

◆Set STANDALONE = "yes" in XML declaration.

◆Put DTD right after XML declaration line.

Separate file:

◆Set STANDALONE = "no" in XML declaration.

◆Put DTD in another file say myfile.dtd.

◆Put the line in the XML document:

```
<!DOCTYPE myroot SYSTEM "myfile.dtd">
```

name of root element in XML document   private DTD   DTD file location

DTD Question

Build a DTD for our relational database:

◆Emp (eno, ename, title, salary, dno)

◆Project (pno, pname, budget, dno)

◆WorksOn (eno, pno, resp, dur)

◆Department (dno, dname, mgr)

XML Schema

While DTDs are sufficient for many uses of XML, they are lacking compared to other forms of database schema. Specifically, they do not have good support for:

◆data types

◆constraints

◆explicit primary keys and foreign key references

Further, DTDs are written in a non-XML syntax.

XML Schema was defined by the W3C to provide a standard XML schema language which itself is written in XML and has better support for data modeling.

Types in XML Schema

Elements that contain other elements are called *complex types*:

```
<xsd:element name = "Dept">
  <xsd:complexType>
      <xsd:sequence>
            // Children defined here
      </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Elements that have no subelements are *simple types*:

```
<xsd:element name = "name" type = "xsd:string" />
<xsd:element name = "budget" type = "xsd:decimal" />
```

Cardinality in XML Schema

The number of occurrences of elements can be controlled by cardinality constraints minOccurs and maxOccurs:

```
<xsd:element name = "Emp"
          minOccurs="0" maxOccurs="unbounded">

<xsd:element name = "budget" type = "xsd:decimal"
  minOccurs="1" maxOccurs="1" />
```

Constraints in XML Schema

Uniqueness constraints dictate that the value of an element (with a specified path) must be unique:

```
<xsd:unique name = "UniqueDno">
  <xsd:selector xpath = "Dept" />
  <xsd:field xpath = "@dno" />
</xsd:unique>
```

Key constraints are uniqueness constraints where the value cannot be null.

```
<xsd:key name = "EmpKey">
  <xsd:selector xpath = "Dept/Emp" />
  <xsd:field xpath = "@eno" />
</xsd:key>
```

Constraints in XML Schema (2)

Reference constraints forces the value of a reference to be constrained to specified keys:

```
<xsd:keyref name = "DeptMgrFK" refer="EmpKey">
  <xsd:selector xpath = "Dept" />
  <xsd:field xpath = "@mgr" />
</xsd:keyref>
```

Other Features of XML Schema

XML Schema also allows you to:
◆ Define your own types
◆ Define schema references to simplify schema maintenance
◆ Define groups of elements or attributes
◆ Use groups to allow for schema variations and choices
◆ Construct elements that are lists or unions of values

XML Schema Example

```
<?xml version = "1.0">
<xsd:schema xmlns:xsd = "http://www.w3.org/2000/10/XMLSchema">
<xsd:element name = "root">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name = "Dept" minOccurs="1" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:attribute name = "dno" type = "xsd:string" />
          <xsd:attribute name = "mgr" type = "xsd:string" />
          <xsd:element name = "Emp" minOccurs="0" maxOccurs="unbounded">
            <xsd:complexType>
              <xsd:element name = "name" type = "xsd:string" />
              <xsd:attribute name = "eno" type = "xsd:string" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="budget" minOccurs="0" type ="xsd:decimal" />
        </xsd:complexType>
      </xsd:element>
```

XML Schema Example

```
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:key name = "DeptKey">
  <xsd:selector xpath = "Dept" />
  <xsd:field xpath = "@dno" />
</xsd:key>
<xsd:key name = "EmpKey">
  <xsd:selector xpath = "Dept/Emp" />
  <xsd:field xpath = "@eno" />
</xsd:key>
<xsd:keyref name = "DeptMgrFK" refer = "EmpKey">
  <xsd:selector xpath = "Dept" />
  <xsd:field xpath = "@mgr" />
</xsd:keyref>
</xsd:schema>
```

XML Parsers

An *XML parser* processes the content and structure of an XML document and may use its schema (if one is present).
◆ Example: xmllint

XML parsers read in the XML document and determine if the document is well-formed and valid (if a schema is provided).

Once a document is parsed, programs may manipulate the document using one of two common interfaces: DOM and SAX.
◆ Note that you can write and parse XML documents without using a parser as the document itself is just a text file.
  ⇨ DOM is a tree-based API that parses the entire document into an in-memory tree and provides methods for traversing the tree.
  ⇨ SAX is an event-based serial access method. As a parser reads an XML file, it generates events when a new element is seen, closed, etc.

XSL and XSLT

XSL (eXtensible Stylesheet Language) is a W3C recommendation that defines how XML data is displayed.
◆ It is similar but more powerful than Cascading Stylesheet Specification (CSS) used with HTML.

XSLT (eXtensible Stylesheet Language for Transformations) is a subset of XSL that provides a method for transforming XML (or other text documents) into other documents (XML, HTML).

XSLT Overview

XSLT works by defining transformation rules (templates) that match regions of the document specified by XPath expressions and then produce output for each matched region.

◆This template matching is often done recursively.

Key syntax:

◆Anything in **{ }** is evaluated as an XPath expression.

◆Template match (and recursively apply template):

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

◆Extract value:

```
<xsl:value-of select="name"/>
```

Page 31

XSLT Example

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
           version="2.0">
<xsl:output method="xml" indent="yes"/>          XML output (indent)
  <xsl:template match="root">          action when match root element
      <Employees>
            <xsl:apply-templates select="Dept"/>
      </Employees>
  </xsl:template>
  <xsl:template match="Dept">          match Dept under root
      <xsl:apply-templates select="Emp"/>
  </xsl:template>
  <xsl:template match="Emp">          How to output attribute
      <Emp number="{@eno}">          in HTML or XML output.
            <xsl:value-of select="name"/>
      </Emp>
  </xsl:template>          Outputs value of name
</xsl:stylesheet>          tag as text in new Emp tag.
```

Page 32

XSLT Example (2)

```
<root>
  <Dept dno = "D1">
      <Emp eno="E7"><name>R. Davis</name></Emp>
      <Emp eno="E8"><name>J. Jones</name></Emp>
  </Dept>
  <Dept dno = "D2" mgr = "E7">
      <Emp eno="E6"><name>L. Chu</name></Emp>
      <Emp eno="E3"><name>A. Lee</name></Emp>
      <budget>350000</budget>
  </Dept>
</root>
```

Output:
```
<Employees>
  <Emp number="E7">R. Davis</Emp>
  <Emp number="E8">J. Jones</Emp>
  <Emp number="E6">L. Chu</Emp>
  <Emp number="E3">A. Lee</Emp>
</Employees>
```

Page 33

Conclusion

Extensible Markup Language (***XML***) is a markup language that allows for the description of data semantics.

An XML document does not need a schema to be *well-formed*. An XML document is *valid* if it conforms to its DTD schema. XML Schemas can define schemas with more precision.

XML may be parsed (DOM/SAX), queried (XPath/XQuery) and displayed/transformed (XSL,XSLT).

Page 34

Objectives

◆List some advantages of XML.

◆Given an XML document, determine if it is well-formed.

◆Given an XML document and a DTD, determine if it is valid.

◆Explain the difference between #PCDATA and CDATA.

◆Know the symbols (?,*,+) for cardinality constraints in DTDs.

◆Compare and contrast ID/IDREFs in DTDs with keys and foreign keys in the relational model.

◆List some advantages that XML Schema has over DTDs.

◆Compare/contrast the two XML parser APIs: DOM and SAX.

◆Explain why and when namespaces are used.

◆Explain what XSL and XSLT are used for.

◆Be able to write a XSLT document to transform an XML file.

Page 35