Semi-structured Data

Facts about the Web
- Growing fast
- Popular
- Semi-structured data
  - Data is presented for ‘human’-processing
  - Data is often ‘self-describing’ (including name of attributes within the data fields)

Vision for Web data
- *Object-like* – it can be represented as a collection of objects of the form described by the conceptual data model
- *Schemaless* – not conformed to any type structure
- *Self-describing* – necessary for machine readable data

Facts about database systems
- Integration of databases with different schemas is often needed
- Sharing information between different databases on the World Wide Web becomes more and more important for business
Semi-structured data

- Bridging different data models (relational, object-oriented)

Semi-structured data representation

- A database of semi-structured data is a graph with
  - A set of nodes, each is either a leaf or an interior node;
  - Each interior node has a set of arcs coming out from it, connecting it with another node; each arc has a label; and
  - A root that does not have an arc entering it.
  Every node must be reachable from the root.

Example of semi-structured data representing a movie and stars

Information integration via semi-structured data

- Simple
- Semi-structured data as interface between users of different databases (with different schemas)
XML – Overview

- Simplifying the data exchange between software agents
- Popular thanks to the involvement of W3C (World Wide Web Consortium – independent organization www.w3c.org)

XML – Characteristics

- Simple, open, widely accepted
- HTML-like (tags) but extensible by users (no fixed set of tags)
- No predefined semantics for the tags (because XML is developed not for the displaying purpose)
- Semantics is defined by stylesheet (later)

XML Documents

- User-defined tags: 
  `<tag> info </tag>`
- Properly nested: `<tag1>.. <tag2>… </tag1> </tag2>` is not valid
- Root element: an element contains all other elements
- Processing instructions `<?command ....?>`
- Comments `<!-- comment -->`
- CDATA type
- DTD

XML element

- Begin with an opening tag of the form `<XML_element_name>`
- End with a closing tag `</XML_element_name>`
- The text between the beginning tag and the closing tag is called the content of the element
<Star-Movie-Data>
  <Star>
    <Name>Carrie Fisher</Name>
    <Address><Street>123 Maple St.</Street><City>Hollywood</City></Address>
    <Address><Street>5 Locus Ln.</Street><City>Malibu</City></Address>
  </Star>
  <Star>
    <Name>Mark Hamill</Name>
    <Address><Street>456 Oak Rd.</Street><City>Brentwood</City></Address>
  </Star>
  <Movie>
    <Title>Star Wars</Title><Year>1997</Year>
  </Movie>
</Star-Movie-Data>

**Relationship between XML elements**

- **Child-parent relationship**
  - Elements nested directly in an element are the children of this element (*Student* is a child of *PersonList*, *Name* is a child of *Student*, etc.)

- **Ancestor/descendant relationship**: important for querying XML documents (extending the child/parent relationship)

**XML elements & Database Objects**

- XML elements can be converted into objects by
  - considering the tag’s names of the children as attributes of the objects
  - Recursive process

```
<Student StudentID="123">
  <Name>XYZ PQR</Name>
  <CrsTaken>
    <CrsName>CIS82</CrsName>
    <Grade>A</Grade>
  </CrsTaken>
</Student>
```
XML elements & Database Objects

• Differences: Additional text within XML elements

```xml
<Student StudentID="123">
  <Name> "XYZ PQR" </Name>
  has taken the following course
  <CrsTaken>
    Database management system II
    <CrsName>CS582</CrsName>
    with the grade
    <Grade>"A"</Grade>  </CrsTaken>
</Student>
```

XML elements & Database Objects

• Differences: XML elements are ordered

```xml
<CrsTaken>
  <CrsName>"CS582"</CrsName>
  <Grade>"A"</Grade>
</CrsTaken>

<#901, Grade: "A", CrsName: "CS582">1
```

XML Attributes

• Can occur within an element (arbitrary many attributes, order unimportant, same attribute only one)
• Allow a more concise representation
• Could be replaced by elements
• Less powerful than elements (only string value, no children)
• Can be declared to have unique value, good for integrity constraint enforcement (next slide)

XML Attributes

• Can be declared to be the type of ID, IDREF, or IDREFS
• ID: unique value throughout the document
• IDREF: refer to a valid ID declared in the same document
• IDREFS: space-separated list of strings of references to valid IDs
Well-formed XML Document

- It has a root element
- Every opening tag is followed by a matching closing tag, elements are properly nested
- Any attribute can occur at most once in a given opening tag, its value must be provided, quoted

Document Type Definition

- Set of rules (by the user) for structuring an XML document
- Can be part of the document itself, or can be specified via a URL where the DTD can be found
- A document that conforms to a DTD is said to be valid
- Viewed as a grammar that specifies a legal XML document, based on the tags used in the document

DTD Components

- A name – must coincide with the tag of the root element of the document conforming to the DTD
- A set of ELEMENTs – one ELEMENT for each allowed tag, including the root tag
- ATTLIST statements – specifies the allow attributes and their type for each tag
- *, +, ? – like in grammar definition
  - * : zero or finitely many number
  - + : at least one
  - ? : zero or one

DTD Components – Element

```xml
<!ELEMENT Name definition>
```

Name of the element
definition can be: EMPTY, (#PCDATA), or element list (e1,e2,…,en) where the list (e1,e2,…,en) can be shorted using grammar like notation
DTD Components – Element

<!ELEMENT Name(e1,..,en)> 
  n\textsuperscript{th} – element 

<!ELEMENT PersonList (Title,Contents)> 
<!ELEMENT Contents(Person *)>

DTD Components – Element

<!ELEMENT Name EMPTY> 
no child for the element Name
<!ELEMENT Name (#PCDATA)> 
value of Name is a character string

<!ELEMENT Title EMPTY> 
<!ELEMENT Id (#PCDATA)> 

DTD Components – Attribute List

<!ATTLIST EName Att {Type} Property> 
where 
- Ename – name of an element defined in the DTD 
- Att – attribute name allowed to occur in the 
  opening tag of Ename 
- {type} – might/might not be there; specify the type 
  of the attribute (CDATA, ID, IDREF, IDREFS) 
- Property – either #REQUIRED or #IMPLIED

<!DOCTYPE Stars [ 
  <!ELEMENT STARS (STAR*)> 
  <!ELEMENT STAR(NAME,ADDRESS+,MOVIES)> 
  <!ELEMENT NAME (#PCDATA)> 
  <!ELEMENT ADDRESS (STREET, CITY)> 
  <!ELEMENT STREET (#PCDATA)> 
  <!ELEMENT CITY (#PCDATA)> 
  <!ELEMENT MOVIES (MOVIE*)> 
  <!ELEMENT MOVIE (TITLE, YEAR)> 
  <!ELEMENT TITLE (#PCDATA)> 
  <!ELEMENT YEAR (#PCDATA)> 
]>
A simple DTD for the movie and star database (no integrity constraints)
A DTD for the movie and star database with attributes and integrity constraints