XML

COMPILED BY
M ABDUL JAWAD

SEMI-STRUCTURED DATA

- Semi-structured data is a form of structured data that ***doesn't conform with the formal structure of data models associated with relational databases,*** but ***contains tags or other markers to separate semantic elements and enforce hierarchies of records*** and fields within the data.

- Un-Structured Data - "*Data that may be irregular or incomplete and have a structure that may change rapidly or unpredictably*".

SEMI-STRUCTURED DATA

- Semi-Structured data has gained importance for following reasons:
  - it may be desirable to treat web sources like a database , but we cannot constrain these sources with a schema.
  - it may be desirable to have flexibility format for data exchange between disparate databases.
  - the emergence of XML as the standard for data representation and exchange on the WEB and the similarity between XML and semi-structured data.
- Unfortunately relational, object oriented database management systems don't handle data of this nature particularly well.

SEMI-STRUCTURED DATA

- Information normally associated with Schema is contained within data itself.
- Some forms of semi-structured data have no separate schema, in others it exists but places loose constraints on data.
- Example
  - Data isn't regular. Some may hold First Name and Last Name, while others may hold First Name, Middle Name and Last Name.
  - Some may use a field "rent" for years or moths others may use it for a day, or a week.

SEMI-STRUCTURED DATA

```
<itemlist>
    <item>
        <identifier> RS1 </identifier>
        <description> Atom powered rocket sled </description>
        <quantity> 2 </quantity>
        <price> 199.95 </price>
    </item>
    <item>
        <identifier> SG2 </identifier>
        <description> Superb glue </description>
        <quantity> 1 </quantity>
        <unit-of-measure> liter </unit-of-measure>
        <price> 29.95 </price>
    </item>
</itemlist>
```

INTRODUCTION TO XML

- Database schema isn't appropriate for data exchange.
- We need a flexible data format for data exchange.
- SGML – Standard Generalized Markup Language, is a language for defining markup languages
- HTML and XML are the applications of SGML.
- Like HTML – Hyper Text Markup Language, XML has its roots in document management.
- HTML is used to format a document while XML is designed to represent data.
- XML were particularly used as a data format when an application must communicate with another application or integrate information from several other applications.

INTRODUCTION TO XML

- Anyways both HTML and XML are markup languages.
  - Markup refers to anything in a document that isn't intended to be part of printed output.
  - In document processing, a markup language is a formal description of what part of the document is content, what part is markup and what the markup means.
  - Just like database systems evolved from physical file processing to provide a separate logical view, markup languages evolved from specifying the function of a content.
    - Functional markup will specify the text representing section headings be marked up being a section heading instead of being marked up as text to be printed in large size, bold font.

COMPARISON

XML

- Extensible set of tags.
- Content orientated.
- Standard Data infrastructure.
- Allows multiple output forms.

HTML

- Fixed set of tags.
- Presentation oriented.
- No data validation capabilities.
- Single presentation.

XML

- XML is a meta-language – Enabling designers to create UNLIMITED customized tags to provide functionality not available with HTML.
- Unlike HTML, XML itself is intended for information management not display.
  - An XML element is made up of a start tag, an end tag, and data in between.
    - Example: <coordinator>Zahoor Ahmad </director>
    - Example of another element with the same value: <instructor> Zahoor Ahmad </instructor>
  - XML tags are case-sensitive: <CITY> <City> <city>
  - XML can abbreviate empty elements, for example:

<married> </married> can be abbreviated to <married/>

WHY USE XML

- Compared to storage of data in a relational database, the XML representation may be inefficient, since the tag names are repeated throughout the document.
- However, an XML representation has significant advantages when it is used to exchange data between organizations and for storing complex structured information in files.
  - Presence of tags, make the message self-documenting i.e Schema need not be consulted to understand meaning of text.
  - Format of document isn't rigid, different items of same item-list may have uneven or different measures of tags present – No harsh constraints.
    - Example..

STRUCTURE OF XML

- Fundamental construct of XML is an Element.
  - An element is simply a pair of matching START and END tags and all the text that appears between them. < >.
  - XML follows a tree structure; obviously a **ROOT** at the top will wrap everything.
  - Elements in XML document must nest properly.
    - **<Course> … <title> ….. </title> …. </Course>.**
  - Text is said to appear in the ***CONTEXT OF*** an element if it appears between the starting and end-tag of that element.
  - Tags are properly nested if every start-tag has unique matching end-tag that is in the context of same parent element.

STRUCTURE OF XML

- Text may also be mixed with the sub-elements of an element.

```
...
    <course>
        This course is being offered for the first time in 2009.
        <course_id> BIO-399 </course_id>
        <title> Computational Biology </title>
        <dept_name> Biology </dept_name>
        <credits> 3 </credits>
    </course>
...
```

- Nested Structures although being natural in XML, may lead to redundant storage of data.
- Nested representations are widely used in XML data interchange to avoid Joins.

STRUCTURE OF XML

- XML specifies the notion of ***Attributes.***
  - ***Attributes of an element appear as 'name-value' pairs before the closing >.***
  - ***It is important to distinct Sub-element and an attribute in the context of document construction.***
  - ***An attribute is implicitly text that doesn't appear in the printed or displayed document.***
  - ***In general we use, attributes only to represent identifiers and store all other data as sub-elements.***

IDENTIFIERS FOR XML TAGS

- Since XML documents are designed to be exchanged between applications, a ***'namespace'*** mechanism has been introduced to allow organizations to specify globally unique names to be used as element tags in documents.
  - ***Idea of namespace is to prepend each tag or attribute with universal resource identifier.***
  - ***If Central University K****ashmir needs to ensure that XML documents it created would not duplicate tags used by any other university, it would prepend a unique identifier with : to each tag name.*
    - ***If www.cuk.edu.in is a unique identifier.***

```
<university xmlns:yale="http://www.yale.edu">
    . . .
    <yale:course>
        <yale:course_id> CS-101 </yale:course_id>
        <yale:title> Intro. to Computer Science</yale:title>
        <yale:dept_name> Comp. Sci. </yale:dept_name>
        <yale:credits> 4 </yale:credits>
    </yale:course>
    . . .
</university>
```

XML DOCUMENT SCHEMA

- Databases have schemas, which are used to ***constrain what information can be stored*** in the database and to ***constrain the data types*** of the stored information.
- XML documents can be created without any associated schema.
  - An element may have any number of sub-elements or attributes.
  - Well this is only possible given the self-describing nature of the data format; but sometimes it becomes mandatory to enforce some constraints.
- The first attempt to describe Schema of XML is the ***Document Type Definition***.
- With a DTD, independent groups of people can agree on a standard DTD for interchanging data.
- An application can use a DTD to verify that XML data is valid.

DOCUMENT TYPE DEFINITION - DTD

- DTD is an optional part of an XML document which serves the main purpose of Schema.
- DTD, however constraints only the appearance of Sub-elements and attributes within an element.
  - DTD is primarily a list of rules for what pattern of sub-elements may appear within an element.
- Building Blocks of XML Documents
  - ***Elements***
  - ***Attributes***
  - ***Entities***
  - ***PCDATA***
  - ***CDATA***

BUILDING BLOCKS OF XML

- **Elements:**
  - Elements are the **main building blocks** of both XML and HTML documents.
  - <message>some text</message>
  - Elements can contain text, other elements, or be empty. Examples of empty HTML elements are "hr", "br".
- **Attributes:**
  - Attributes provide **extra information about elements**.
  - Attributes are always placed inside the opening tag of an element. Attributes always come in name/value pairs.

BUILDING BLOCKS OF XML

- **Entities:**
  - Some characters have a special meaning in XML, like the less than sign (<) that defines the start of an XML tag.
- **PCDATA**:
  - PCDATA means parsed character data - the text found between the start tag and the end tag of an XML element. *PCDATA is text that WILL be parsed by a parser. The text will be examined by the parser for entities and markup.*
- **CDATA:**
  - CDATA means character data.
  - **CDATA is text that will NOT be parsed by a parser**. Tags inside the text will NOT be treated as markup and entities will not be expanded.

CDATA VS PCDATA

CDATA

- **<?xml** version="1.0"**?>**

<!DOCTYPE employee SYSTEM "student.dtd"**>**

<student>

<![CDATA[

<firstname>Irfan</firstname>

<lastname>Rasool</lastname>

<email>da.irfanrasool@gmail.com</email>

]]>

</student>

- **Output:**

<firstname>Irfan</firstname><lastname>Rasool</lastname><email>da.irfanrasool@gmail.com </email>

PCDATA

<?xml version="1.0"**?>**

<!DOCTYPE employee SYSTEM "student.dtd"

<student>

<firstname>Irfan</firstname>

<lastname>Rasool</lastname>

<email>da.irfanrasool@gmailcom</email>

</student>

- ***Output:***
- *Irfan Rasool da.irfanrasool@gmail.com*

DTD EXAMPLE

- Example : University Database
  - Department(***dept_name***, building, budget)
  - Course (***course_id, title,*** **dept_name**, credits)
  - Instructor (*IID,* name, **dept_name**, salary)
  - Teaches (**IID, course_id)**
- **Required DTD ->**
- the + operator specifies "one or more."
- the ∗operator is used to specify "zero or more,"
- while the ? operator is used to specify an optional element (that is, "zero or one").

```
<!DOCTYPE university [
    <!ELEMENT university ( (department|course|instructor|teaches)+)>
    <!ELEMENT department ( dept_name, building, budget)>
    <!ELEMENT course ( course_id, title, dept_name, credits)>
    <!ELEMENT instructor (IID, name, dept_name, salary)>
    <!ELEMENT teaches (IID, course_id)>
    <!ELEMENT dept_name( #PCDATA )>
    <!ELEMENT building( #PCDATA )>
    <!ELEMENT budget( #PCDATA )>
    <!ELEMENT course_id ( #PCDATA )>
    <!ELEMENT title ( #PCDATA )>
    <!ELEMENT credits( #PCDATA )>
    <!ELEMENT IID( #PCDATA )>
    <!ELEMENT name( #PCDATA )>
    <!ELEMENT salary( #PCDATA )>
]>
```

UNIVERSITY DTD WITH ATTRIBUTES

- University DTD with ID and IDREF attribute types.

```
<!DOCTYPE university-3 [
    <!ELEMENT university ( (department|course|instructor)+)>
    <!ELEMENT department ( building, budget )>
    <!ATTLIST department
        dept_name ID #REQUIRED >
    <!ELEMENT course (title, credits )>
    <!ATTLIST course
        course_id ID #REQUIRED
        dept_name IDREF #REQUIRED
        instructors IDREFS #IMPLIED >
    <!ELEMENT instructor ( name, salary )>
    <!ATTLIST instructor
        IID ID #REQUIRED
        dept_name IDREF #REQUIRED >
    · · · declarations for title, credits, building,
        budget, name and salary · · ·
]>
```

DTD EXAMPLE

- Attribute of type **ID** provides a unique identifier for the element.
  - At most one attribute of an element is permitted to be of type ID.
- An attribute of type **IDREF** is reference to an element., while attribute of type **IDREFS** is reference to list of attributes.
  - The attribute must contain a value that appears in the id attribute of some element in the document.

- Attribute Values:
  - #REQUIRED - *a value must be specified for the attribute in each element.*
    - <!ATTLIST element-name attribute-name attribute-type #REQUIRED>
  - #IMPLIED – *when we don't want to force the author to include an attribute*.
    - <!ATTLIST element-name attribute-name attribute-type #IMPLIED>
  - #FIXED - *keyword when you want an attribute to have a fixed value without allowing the author to change it.*
    - <!ATTLIST element-name attribute-name attribute-type #FIXED "value">
  - #DEFAULT – *If no value is specified for an attribute, a default value is inserted.*
    - <!ATTLIST element-name attribute-name "value">

XML DATA WITH ID AND IDREF ATTRIBUTES

```
<university-3>
    <department dept name="Comp. Sci.">
        <building> Taylor </building>
        <budget> 100000 </budget>
    </department>
    <department dept name="Biology">
        <building> Watson </building>
        <budget> 90000 </budget>
    </department>
    <course course id="CS-101" dept name="Comp. Sci"
            instructors="10101 83821">
        <title> Intro. to Computer Science </title>
        <credits> 4 </credits>
    </course>
    ....
    <instructor IID="10101" dept name="Comp. Sci.">
        <name> Srinivasan </name>
        <salary> 65000 </salary>
    </instructor>
    ....
</university-3>
```

DTD - LIMITATIONS

- Some of the limitations of DTD schema are:
  - Individual text elements and attributes can`t be ***typed*** further. For instance we have an element
    - <balance>…..</balance> This element can`t be constrained to be a positive number. Lack of such constrains is problematic for data processing and exchange.
  - There is lack typing in IDs and IDREFSs.
    - No way to specify the type of element to which an IDREF or IDREFS attribute should refer.

XML SCHEMA

- An effort to address the deficiencies of the DTD and develop a more sophisticated schema language – ***XML Schema***.
- XML Schema defines a number of built-in types *string, integer, decimal, date* and *Boolean.*
- In order to avoid conflicts with user-defined, we prefix the XML Schema with the namespace prefix "xs"; this prefix is associated with the XML Schema namespace by the xmlns:xs specification in the root element.
  - ***<xs:schema xmlns:xs=http://www.cukashmir.ac.in/XMLSchema>***
  - ***Let us compile our first XML Schema document.***

XML SCHEMA VERSION OF UNIV. DTD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="university" type="universityType" />
<xs:element name="department">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="dept name" type="xs:string"/>
            <xs:element name="building" type="xs:string"/>
            <xs:element name="budget" type="xs:decimal"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
....
<xs:element name="instructor">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="IID" type="xs:string"/>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="dept name" type="xs:string"/>
            <xs:element name="salary" type="xs:decimal"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
... Contd.
```

XML SCHEMA VERSION OF UNIV. DTD

```
….
<xs:complexType name=“UniversityType”>
   <xs:sequence>
      <xs:element ref=“department” minOccurs=“0” maxOccurs=“unbounded”/>
      <xs:element ref=“course” minOccurs=“0” maxOccurs=“unbounded”/>
      <xs:element ref=“instructor” minOccurs=“0” maxOccurs=“unbounded”/>
      <xs:element ref=“teaches” minOccurs=“0” maxOccurs=“unbounded”/>
   </xs:sequence>
</xs:complexType>
</xs:schema>
```

- Choice of “xs:” was ours -- any other namespace prefix could be chosen