COS221
L20 - Extensible Markup Language
(Chapter 12 in Edition 6 and Chapter 13 in Edition 7)

Linda Marshall

17 April 2023

EXtensible Markup Language (XML)

- XML was designed to store and transport data. It is a standard for structuring and exchanging data over the web.
- Types of data:
    - **Structured** data: Stored in a strict format and constraints that are maintained a program, such as by the DBMS.
    - **Unstructured** data: Has its own internal structure, but does not conform neatly into a spreadsheet or database, for example a pdf file, word processing documents, Web pages in HTML that contain some data are considered as unstructured data.
    - **Semi-structured** data: Data that doesn't reside in a relational database but that does have some organisational properties that make it easier to analyse, for example XML documents and NoSQL databases.

Representing semi-structured data as a graph

- *Labels* (or tags) represent names of attributes, object types and relationships
- *Internal nodes* represent individual or composite attributes
- *Leaf nodes* represent data values of simple (atomic) attributes.

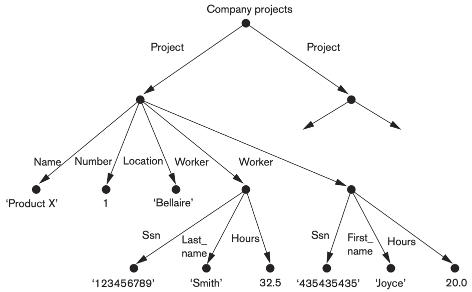


Figure 13.1
Representing semistructured data as a graph.

XML Hierarchical (Tree) Data Model

- The basic object in XML is the XML document.
- Two main structuring concepts that are used to construct an XML document:
  - Elements
  - Attributes
- Attributes in XML provide additional information that describe elements
- Additional concepts such such as entities, identifiers, and references exist in XML
- An XML document that conforms to a structure or DTD is *structured*, an XML document that does not conform to a schema is called *semi-structured* (or schemaless).

XML Documents, DTD, and XML Schema

An XML document is said to be *wellformed* if it complies with the following conditions:

- begins with a *XML declaration*
- follow the syntactic guidelines of the tree data model. That is, must have a single root element and every element must include a matching start and end tag.

If an XML document is wellformed, it is syntactically correct. It is therefore possible to traverse the tree structure with API's such as DOM (Document Object Model) and SAX (Simple API for XML).

XML Documents, DTD, and XML Schema - *cont.*

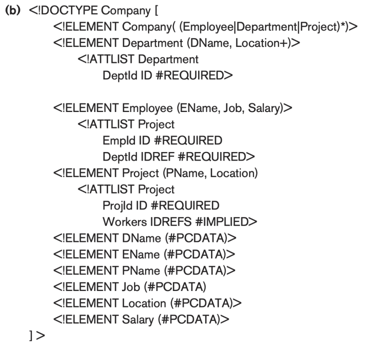
A stronger criterion for XML documents is it must be *valid*.

- Valid XML documents are wellformed.
- The XML document tags must follow the format as set out in the DTD or XML Schema file.
- To specify that the XML schema must be checked against the DTD, the XML file must include in its declaration the following lines:

```
<?xml version = "1.0" standalone = "no"?>
<!DOCTYPE Projects SYSTEM "proj.dtd">
```

XML Documents, DTD, and XML Schema - *cont.*

An example of a XML DTD is given by:



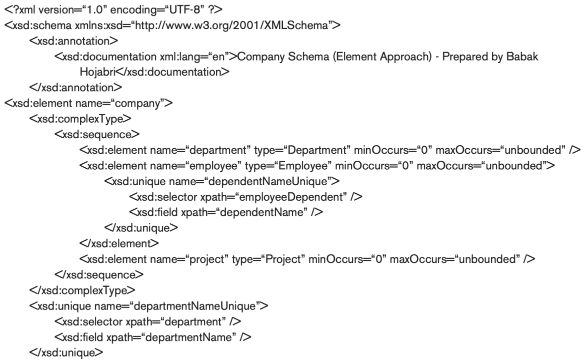
```
(b) <!DOCTYPE Company [
        <!ELEMENT Company( (Employee|Department|Project)*)>
        <!ELEMENT Department (DName, Location+)>
            <!ATTLIST Department
                DeptId ID #REQUIRED>

        <!ELEMENT Employee (EName, Job, Salary)>
            <!ATTLIST Project
                EmpId ID #REQUIRED
                DeptId IDREF #REQUIRED>
        <!ELEMENT Project (PName, Location)
            <!ATTLIST Project
                ProjId ID #REQUIRED
                Workers IDREFS #IMPLIED>
        <!ELEMENT DName (#PCDATA)>
        <!ELEMENT EName (#PCDATA)>
        <!ELEMENT PName (#PCDATA)>
        <!ELEMENT Job (#PCDATA)
        <!ELEMENT Location (#PCDATA)>
        <!ELEMENT Salary (#PCDATA)>
    ]>
```

Figure 13.4
(a) An XML DTD file called *Projects*.
(b) An XML DTD file called *Company*.

XML Documents, DTD, and XML Schema - *cont.*

- The XML Schema language makes use of the syntax of standard XML documents so that only one parser is required for the structural definition.
- Distinction is made between two types of document:
  - XML instance document - contains both tags and data values
  - XML schema document - contains tags, tree structure elements , constraints etc, but not data values



```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">Company Schema (Element Approach) - Prepared by Babak
            Hojabri</xsd:documentation>
    </xsd:annotation>
<xsd:element name="company">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="department" type="Department" minOccurs="0" maxOccurs="unbounded" />
            <xsd:element name="employee" type="Employee" minOccurs="0" maxOccurs="unbounded">
                <xsd:unique name="dependentNameUnique">
                    <xsd:selector xpath="employeeDependent" />
                    <xsd:field xpath="dependentName" />
                </xsd:unique>
            </xsd:element>
            <xsd:element name="project" type="Project" minOccurs="0" maxOccurs="unbounded" />
        </xsd:sequence>
    </xsd:complexType>
    <xsd:unique name="departmentNameUnique">
        <xsd:selector xpath="department" />
        <xsd:field xpath="departmentName" />
    </xsd:unique>
```

Storing and Extracting XML Documents from Databases

Approaches for storing XML documents for querying and retrieval:

- Using a file system or a DBMS to store the documents as text.
  Good for storing schemaless and document-centric XML documents
- Using a DBMS to store the document contents as data elements.
  Works well with XML documents were there is a strict structure, that is XML schemas with either a DTD or XML Schema
- Designing a specialised system for storing native XML data.
  Specific XML-based hierarchical database
- **Creating or publishing customised XML documents from pre-existing relational databases.**
  Middleware layer (application) to map the database schema onto the presentation schema in XML.

XML Query Languages

Two query language standards have emerged:

- XPath - makes use of identified path expression to match patterns allowing one to write queries to select items from a tree-structured XML documemt

Figure 13.6
Some examples of XPath expressions on XML documents that follow the XML schema file *company* in Figure 13.5.

1. /company
2. /company/department
3. //employee [employeeSalary gt 70000]/employeeName
4. /company/employee [employeeSalary gt 70000]/employeeName
5. /company/project/projectWorker [hours ge 20.0]

- XQuery - allows for general queries on one or more XML documents

```
FOR <variable bindings to individual nodes (elements)>
LET <variable bindings to collections of nodes (elements)>
WHERE <qualifier conditions>
ORDER BY <ordering specifications>
RETURN <query result specification>
```

```
LET $d : = doc(www.company.com/info.xml)
FOR $x IN $d/company/project[projectNumber = 5]/projectWorker,
      $y IN $d/company/employee
WHERE $x/hours gt 20.0 AND $y.ssn = $x.ssn
ORDER BY $x/hours
RETURN <res> $y/employeeName/firstName, $y/employeeName/lastName,
                   $x/hours </res>
```

Extracting XML Documents from Relational Databases

- Most database systems follow a flat relational model. Adding referential integrity, results in the database schema being modelled as a graph structure.
- (E)ER models are inherently graph structures.
- We have a mapping between the (E)ER and relational models.
- As XML is hierarchical in nature, it is necessary to create hierarchical views of the relational schema.
- This can be done from either the (E)ER-diagram or the relational schema with referential integrity, by translating subgraphs into trees.

Extracting XML Documents from Relational Databases

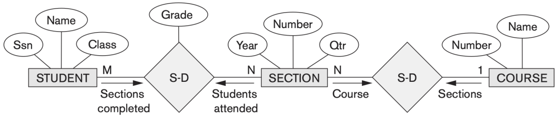
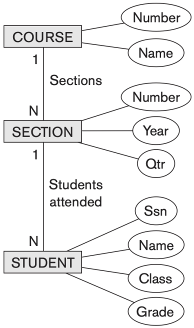


Figure 13.9
Subset of the UNIVERSITY database schema needed for XML document extraction.

Hierarchical tree view with COURSE chosen as the root. Either SECTION or STUDENT could also have been considered.



```
<xsd:element name="root">
    <xsd:sequence>
    <xsd:element name="course" minOccurs="0" maxOccurs="unbounded">
        <xsd:sequence>
            <xsd:element name="cname" type="xsd:string" />
            <xsd:element name="cnumber" type="xsd:unsignedInt" />
            <xsd:element name="section" minOccurs="0" maxOccurs="unbounded">
                <xsd:sequence>
                    <xsd:element name="secnumber" type="xsd:unsignedInt" />
                    <xsd:element name="year" type="xsd:string" />
                    <xsd:element name="quarter" type="xsd:string" />
                    <xsd:element name="student" minOccurs="0" maxOccurs="unbounded">
                        <xsd:sequence>
                            <xsd:element name="ssn" type="xsd:string" />
                            <xsd:element name="sname" type="xsd:string" />
                            <xsd:element name="class" type="xsd:string" />
                            <xsd:element name="grade" type="xsd:string" />
                        </xsd:sequence>
                    </xsd:element>
                </xsd:sequence>
            </xsd:element>
        </xsd:sequence>
    </xsd:element>
    </xsd:sequence>
    </xsd:element>
```

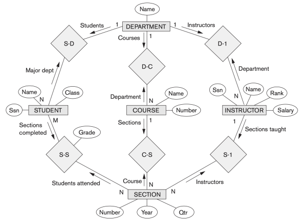
Figure 13.11
XML schema document with *course* as the root.

Extracting XML Documents from Relational Databases

When converting graphs into trees, it is necessary to break cycles. This may result in some entities being duplicated.

- Create a lattice with an entity at the root (STUDENT) - (a)
- Replicate the entity/entities involved in cycles - (b) and (c)

Figure 13.8
An ER schema diagram for a simplified UNIVERSITY database.



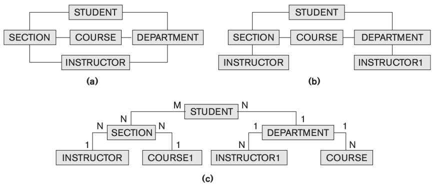


Figure 13.15
Converting a graph with cycles into a hierarchical (tree) structure.

XML/SQL: SQL Functions for Creating XML Data

Extensions have been added to SQL to aid in the translation of relational databases into XML.

- XMLELEMENT. - specifies an element (tag) name to appear in the XML
- XMLFOREST - enables the creation of several elements rather than using XMLELEMENT individually
- XMLAGG - aggregates several elements together
- XMLROOT - allows the selected element to form the root of the XML
- XMLATTRIBUTES - creates attributes for the elements

XML/SQL: SQL Functions for Creating XML Data - Example 1

```
SELECT  XMLELEMENT(NAME "employee",
                      XMLFOREST (
                        E.Lname AS "ln",
                        E.Fname AS "fn",
                        E.Salary AS "sal" ) )
FROM EMPLOYEE AS E
WHERE E.Ssn = "123456789" ;
```

Which results in:

```
<employee>
    <ln>Smith</ln>
    <fn>John</fn>
    <sal>30000</sal>
</employee>
```

XML/SQL: SQL Functions for Creating XML Data - Example 2

```
SELECT XMLROOT(
               XMLELEMENT (NAME "dept4emps",
               XMLAGG (
                       XMLELEMENT (NAME "emp"
                       XMLFOREST (Lname, Fname, Salary)
                       ORDER BY Lname ) ) )
FROM EMPLOYEE
WHERE Dno = 4 ;
```

Which results in:

```
<dept4emps>
    <emp><Lname>Jabbar</Lname><Fname>Ahmad</Fname><Salary>25000</Salary></emp>
    <emp><Lname>Wallace</Lname><Fname>Jennifer</Fname><Salary>43000</Salary></emp>
    <emp><Lname>Zelaya</Lname><Fname>Alicia</Fname><Salary>25000</Salary></emp>
</dept4emps>
```

Creating XML data with a program

Examples to do this can be found at:

- PHP - https://programmerblog.net/how-to-generate-xml-files-using-php/
- Java - https://stackoverflow.com/questions/10959924/covert-database-table-result-into-xml-how-to-generate-
- ...

Google is your friend