Chapter 11
The semi-structured data model

Structured data
XML (http://www.w3.org/XML/)
Document Type Definitions
XML Schema
Graphs of Semistructured Data

- Nodes = objects.
- Labels on arcs (like attribute names).
- Atomic values at leaf nodes (nodes with no arcs out).

Flexibility: no restriction on:
- Labels out of a node.
- Number of successors with a given label.
The bar object for Joe’s Bar

The beer object for Bud

Notice a new kind of data.
XML

- XML = *Extensible Markup Language*.
- HTML uses tags for formatting (e.g., “italic”), XML uses tags for *semantics* (e.g., “this is an address”).
- **Key idea**: create tag sets for a domain, and translate all data into properly tagged XML documents.
XML: Motivation

- Data interchange is critical in today’s networked world
  - Examples:
    - Banking: funds transfer
    - Order processing (especially inter-company orders)
    - Scientific data
  - Paper flow of information between organizations is being replaced by electronic flow of information
- Each application area has its own set of standards for representing information.
- XML has become the basis for all new generation data interchange formats.
## Comparison with Relational Data

<table>
<thead>
<tr>
<th></th>
<th>Relational</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structure</strong></td>
<td>Table</td>
<td>Tree, graph</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Non rigid format</td>
</tr>
<tr>
<td><strong>Schema</strong></td>
<td>fixed</td>
<td>Flexible</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tags self describing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Allows nested structures</td>
</tr>
<tr>
<td><strong>Queries</strong></td>
<td>Simple, high level</td>
<td>complex</td>
</tr>
<tr>
<td></td>
<td>language</td>
<td></td>
</tr>
<tr>
<td><strong>Ordering</strong></td>
<td>none</td>
<td>implied</td>
</tr>
</tbody>
</table>
Well-Formed and Valid XML

- **Well-Formed XML** allows you to invent your own tags.
- **Valid XML** conforms to a certain **DTD**, or **XML schema**.

Relational database → Valid XML → Well-formed XML

from strict Structure to loose Structure
Well-Formed XML

keep basic structural requirements
• Single root element
• Matched tags, proper nesting
• Unique attributes within elements

XML Document → XML Parser → Parsed XML

“Not well-formed”
Well-Formed XML (cont.)

- Start the document with a *declaration*, surrounded by `<?xml ... ?>`.

- Normal declaration is:
  ```xml
  <?xml version = "1.0" standalone = "yes" ?>
  ```
  - "standalone" = "no DTD provided."

- Balance of document is a *root tag* surrounding nested tags.
Well-Formed XML (cont.)

- Tags are normally matched pairs, as `<FOO> ... </FOO>`.
- Unmatched tags also allowed, as `<FOO/>`
- Tags may be nested arbitrarily.
- XML tags are case-sensitive.
Example: Well-Formed XML

```xml
<?xml version = "1.0" standalone = "yes" ?>
<BARS>
  <BAR>
    <NAME>Joe’s Bar</NAME>
    <BEER>
      <NAME>Bud</NAME>
      <PRICE>2.50</PRICE>
    </BEER>
    <BEER>
      <NAME>Miller</NAME>
      <PRICE>3.00</PRICE>
    </BEER>
  </BAR>
  ...
</BARS>
```

- Single root element
- Matched tags, proper nesting
- Unique attributes within elements

A NAME subelement
A BEER subelement
Tags surrounding a BEER element
Valid XML

- Each XML needs a standard to define what are valid elements, using
  - XML type specification languages to specify the syntax
    - DTD (Document Type Descriptors)
    - XML Schema
  - Plus textual descriptions of the semantics

- DTD or XML schema

- XML Document
- Validating XML Parser
- Parsed XML

“Not valid”
DTD Structure

<!DOCTYPE <root tag> [ 
  <!ELEMENT <name> (<components>) >
  . . . more elements . . .
] >
Example: DTD

<!DOCTYPE BARS [ 
<!ELEMENT BARS (BAR*)>
<!ELEMENT BAR (NAME, BEER+)>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT BEER (NAME, PRICE)>
<!ELEMENT PRICE (#PCDATA)>
]> 

A BARS object has zero or more BAR’s nested within.

A BAR has one NAME and one or more BEER subobjects.

A BEER has a NAME and a PRICE.

NAME and PRICE are text.
Element Descriptions

- Subtags must appear in order shown.
- A tag may be followed by a symbol to indicate its multiplicity.
  - * = zero or more.
  - + = one or more.
  - ? = zero or one.
- Symbol | can connect alternative sequences of tags.
Example: Element Description

- A name is an optional title (e.g., “Prof.”), a first name, and a last name, in that order, or it is an IP address:

```xml
<!ELEMENT NAME (TITLE?, FIRST, LAST) | IPADDR>
```
Use of DTD’s

1. Set standalone = “no”.

2. Either:
   a) Include the DTD as a preamble of the XML document, or
   b) Follow DOCTYPE and the <root tag> by SYSTEM and a path to the file where the DTD can be found.
Example: (a)

<?xml version = "1.0" standalone = "no" ?>

<!DOCTYPE BARS [ 
  <!ELEMENT BARS (BAR*)> 
  <!ELEMENT BAR (NAME, BEER+)> 
  <!ELEMENT NAME (#PCDATA)> 
  <!ELEMENT BEER (NAME, PRICE)> 
  <!ELEMENT PRICE (#PCDATA)> 
 ]>

<BARS>
  <BAR>
    <NAME>Joe’s Bar</NAME>
    <BEER>
      <NAME>Bud</NAME> <PRICE>2.50</PRICE>
    </BEER>
    <BEER>
      <NAME>Miller</NAME> <PRICE>3.00</PRICE>
    </BEER>
  </BAR>
  <BAR> ...
</BARS>
Example: (b)

- Assume the BARS DTD is in file bar.dtd.

```xml
<?xml version = "1.0" standalone = "no" ?>
<!DOCTYPE BARS SYSTEM "bar.dtd">
<BARS>
   <BAR><NAME>Joe’s Bar</NAME>
   <BEER><NAME>Bud</NAME>
        <PRICE>2.50</PRICE></BEER>
   <BEER><NAME>Miller</NAME>
        <PRICE>3.00</PRICE></BEER>
   </BAR>
   <BAR> ...
</BARS>
```

Get the DTD from the file bar.dtd
Attributes

- Opening tags in XML can have **attributes**.

- In a DTD,
  ```xml
  <!ATTLIST E . . . >
  ``
  declares attributes for element $E$, along with its datatype.

The declaration of the form:
```xml
<!ATTLIST element-name, attribute-name, type>
```
Attribute Declaration

<!ATTLIST element-name, attribute-name, type>

- Name
- **Type of attribute**
  - CDATA
  - ID (identifier) or IDREF (ID reference) or IDREFS (multiple IDREFs)
- Whether
  - mandatory (#REQUIRED)
  - has a default value (value),
  - or neither (#IMPLIED)
Example: Attributes

- Bars can have an attribute `TYPE`, a character string describing the bar.

```xml
<!ELEMENT BAR (NAME BEER*)>
<!ATTLIST BAR TYPE CDATA #IMPLIED>
```

Attribute is optional opposite: `#REQUIRED`
Example: Attributes (Cont.)

Example

```xml
<!ELEMENT BAR (NAME BEER*)>
<!ATTLIST BAR
    TYPE (sushi | sports | other)>

• Bar objects can have a type, and the value of that type is limited to the three strings shown.
Example: Attribute Use

In a document that allows BAR tags, we might see:

```xml
<BAR TYPE = "sushi">
  <NAME>Homma’s</NAME>
  <BEER><NAME>Sapporo</NAME>
    <PRICE>5.00</PRICE></BEER>
  ...
</BAR>
```
ID’s and IDREF’s

- Attributes can be pointers (IDREF) from one object to another (ID).
- Allows the structure of an XML document to be a general graph, rather than just a tree.
Example: ID’s and IDREF’s

A new BARS DTD includes both BAR and BEER subelements.

BARS and BEERS have ID attributes name.

BARS have SELLS subelements, consisting of a number (the price of one beer) and an IDREF theBeer leading to that beer.

BEERS have attribute soldBy, which is an IDREFS leading to all the bars that sell it.
The DTD

```xml
<!DOCTYPE BARS [
  <!ELEMENT BARS (BAR*, BEER*)>
  <!ELEMENT BAR (SELLS+)>
    <!ATTLIST BAR name ID #REQUIRED>
  <!ELEMENT SELLS (#PCDATA)>
    <!ATTLIST SELLS theBeer IDREF #REQUIRED>
  <!ELEMENT BEER EMPTY>
    <!ATTLIST BEER name ID #REQUIRED>
    <!ATTLIST BEER soldBy IDREFS #IMPLIED>]
```

Bar elements have name as an ID attribute and have one or more SELLS subelements.

SELLS elements have a number (the price) and one reference to a beer.

Beer elements have an ID attribute called name, and a soldBy attribute that is a set of Bar names.

No matched closing tag, No subelements, Nor have text as a value
Example: A Document

<BARS>
  <BAR name = "JoesBar">
    <SELLS theBeer = "Bud">2.50</SELLS>
    <SELLS theBeer = "Miller">3.00</SELLS>
  </BAR> ...
  <BEER name = "Bud" soldBy = "JoesBar SuesBar ..." />
</BARS>

No matched closing tag, No subelements, Nor have text as a value
Example DTD for bookstore data

```xml
<!ELEMENT Bookstore (Book*, Author*)>
<!ELEMENT Book (Title, Remark?)>
<!ATTLIST Book ISBN ID #REQUIRED Price CDATA #REQUIRED Edition CDATA #IMPLIED Authors IDREFS #REQUIRED>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Remark (#PCDATA | BookRef)*>
<!ELEMENT BookRef EMPTY>
<!ATTLIST BookRef book IDREF #REQUIRED>
<!ELEMENT Author (First_Name, Last_Name)>
<!ATTLIST Author Ident ID #REQUIRED>
<!ELEMENT First_Name (#PCDATA)>
<!ELEMENT Last_Name (#PCDATA)>
```
EMPTY (no subelements) or ANY (anything can be a subelement)
Limitations of DTDs

- No **typing** of text elements and attributes
  - All values are strings, no integers, reals, etc.
- Difficult to specify unordered sets of subelements
  - Order is usually irrelevant in databases (unlike in the document-layout environment from which XML evolved)
- IDs and IDREFs are **untyped**
XML Schema

- A more powerful way to describe the structure of XML documents.

- XML-Schema declarations are themselves XML documents.

  They describe “elements” and the things doing the describing are also “elements.”
Structure of an XML-Schema Document

```xml
<?xml version = "..." ?>
<xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">
  ...
</xs:schema>
```

So uses of "xs" within the schema element refer to tags from this namespace.

Defines "xs" to be the namespace described in the URL shown. Any string in place of "xs" is OK.
The `xs:element` Element

- Has attributes:
  1. `name` = the `tag-name` of the element being defined.
  2. `type` = the `type` of the element.
     - Could be an XML-Schema type, e.g., `xs:string`.
     - Or the name of a type defined in the document itself.
Example: xs:element

```
<xs:element name = "NAME"
    type = "xs:string" />
```

- Describes elements such as

```
<NAME>Joe’s Bar</NAME>
```
XML Schema: Simple Types

- Elements that do not contain other elements or attributes are of type simpleType.

- Attributes must be defined last:
Complex Types

- To describe elements that consist of subelements, we use `xs:complexType`.
  - Attribute `name` gives a name to the type.
- Typical subelement of a complex type is `xs:sequence`, which itself has a sequence of `xs:element` subelements.
  - Use `minOccurs` and `maxOccurs` attributes to control the number of occurrences of an `xs:element`. 
Example: a Type for Beers

```xml
<xs:complexType name="beerType">
  <xs:sequence>
    <xs:element name="NAME" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="PRICE" type="xs:float" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>
```

Exactly one occurrence

Like ? in a DTD
Construct a complex type

- In place of **xs:sequence**, use:
  - **xs:all**: each of the elements between the opening tag and its matched closing tag must occur, in any order, exactly once each.
  - **xs:choice**: one of the elements found between the opening tag and closing tag.
An Element of Type beerType

We don’t know the name of the element of this type.
**xs:attribute**

- **xs:attribute** elements can be used within a complex type to indicate attributes of elements of that type.
- **attributes of xs:attribute:**
  - name and type as for xs.element.
  - use = “required” or “optional”.
Example: xs:attribute

```xml
<xs:complexType name="beerType">
  <xs:attribute name="name" type="xs:string" use="required" />
  <xs:attribute name="price" type="xs:float" use="optional" />
</xs:complexType>
```
An Element of This New Type

beerType

We still don’t know the element name.

The element is empty, since there are no declared subelements.
Restricted Simple Types

- `xs:simpleType` can describe enumerations and range-restricted base types.
- `name` is an attribute
- `xs:restriction` is a subelement.
Restrictions

- Attribute `base` gives the simple type to be restricted, e.g., `xs:integer`.

- `xs:{min, max}{Inclusive, Exclusive}` are four attributes that can give a lower or upper bound on a numerical range.

- `xs:enumeration` is a subelement with attribute `value` that allows enumerated types.
Example: `license` Attribute for BAR

```xml
<xs:simpleType name = "license">
  <xs:restriction base = "xs:string">
    <xs:enumeration value = "Full" />
    <xs:enumeration value = "Beer only" />
    <xs:enumeration value = "Sushi" />
  </xs:restriction>
</xs:simpleType>
```
Example: Prices in Range [1,5)

```xml
<xs:simpleType name = "price">
  <xs:restriction
      base = "xs:float"
      minInclusive = "1.00"
      maxExclusive = "5.00" />
</xs:simpleType>
```
Keys in XML Schema

- An `xs:element` can have an `xs:key` subelement.

- **Meaning**: within this element, all subelements reached by a certain *selector* path will have unique values for a certain combination of *fields*.

- **Example**: within one BAR element, the name attribute of a BEER element is unique.
Example: Key

```xml
<xs:element name = "BAR" ... >

  ...

  <xs:key name = "barKey">
    <xs:selector xpath = "BEER" />
    <xs:field xpath = "@name" />
  </xs:key>

</xs:element>
```

XPath is a query language for XML. All we need to know here is that a path is a sequence of tags separated by `/`. And `@` indicates an attribute rather than a tag.
Foreign Keys

- An `xs:keyref` subelement within an `xs:element` says that within this element, certain values (defined by selector and field(s), as for keys) must appear as values of a certain key.

- An element has a field or fields that serve as a reference to the key for some other element.
Example: Foreign Key

- Suppose that we have declared that subelement **NAME** of BAR is a key for BARS.
  - The name of the key is **barKey**.
- We wish to declare DRINKER elements that have FREQ subelements. An attribute **bar** of FREQ is a foreign key, referring to the **NAME** of a BAR.
Example: Foreign Key in XML Schema

```xml
<xs:element name = ""DRINKERS"
    ...
    <xs:keyref name = ""barRef"
        refers = ""barKey"
        <xs:selector xpath = ""DRINKER/FREQ"" />
        <xs:field xpath = ""@bar"" />
    </xs:keyref>
</xs:element>
```
XML document (with a XML schema)

```xml
<?xml version="1.0"?>
<XXX
  xmlns="http://www.w3school.com.cn"
  xmlns:xsi="http://www.s3.org/2001/xml_Schema-instance"
  xsi:schemaLocation="http://www.w3school.com.cn XXX.xsd">
  <!--...-->
</XXX>
```
<?xml version="1.0" encoding="ISO-8859-1"?>

<shiporder orderid="889923"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="shiporder.xsd">
  
  <orderperson>John Smith</orderperson>
  <shipto>
    <name>Ola Nordmann</name>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  </shipto>
  
  <item>
    <title>Empire Burlesque</title>
    <note>Special Edition</note>
    <quantity>1</quantity>
    <price>10.90</price>
  </item>
  
  <item>
    <title>Hide your heart</title>
    <quantity>1</quantity>
    <price>9.90</price>
  </item>

</shiporder>
The xml schema (example)

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="shiporder">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="orderperson" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="shipto">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="address" type="xs:string"/>
        <xs:element name="city" type="xs:string"/>
        <xs:element name="country" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
The xml schema (example)

<xs:element name="item" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="note" type="xs:string" minOccurs="0"/>
      <xs:element name="quantity" type="xs:positiveInteger"/>
      <xs:element name="price" type="xs:decimal"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:attribute name="orderid" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>
</xs:schema>
XML Schema (Summary)

- XML Schema is a more sophisticated schema language which addresses the drawbacks of DTDs. Supports
  - Typing of values
    - E.g. integer, string, etc
    - Also, constraints on min/max values
  - User-defined, complex types
  - Many more features, including
    - uniqueness and foreign key constraints, inheritance
- XML Schema is itself specified in XML syntax, unlike DTDs
  - More-standard representation, but verbose
- XML Scheme is integrated with namespaces
- BUT: XML Schema is significantly more complicated than DTDs.
Summarization

- XML and its application
  - XML Elements and attributes
- DTD
  - Identifiers and references in DTD’s
- XML schema
  - Simple types, complex types in XML schema, key and foreign key declaration.
Classroom Exercises:
1) what are the minimum and maximum possible number of Mayor elements?
2) what are the minimum and maximum possible number of Library elements?

<!DOCTYPE CityInfo [ 
  <!ELEMENT CityInfo (Government, Neighborhood+)>
  <!ATTLIST CityInfo Name CDATA #REQUIRED>
  <!ELEMENT Government (Mayor, Assistant)>
  <!ELEMENT Mayor (#PCDATA)>
  <!ELEMENT Assistant (#PCDATA)>
  <!ELEMENT Neighborhood (Library | Bookshop)>?
  <!ATTLIST Neighborhood Name CDATA #REQUIRED>
  <!ELEMENT Library (#PCDATA)>
  <!ELEMENT Bookshop (#PCDATA)> ]>
Classroom Exercises

1) what are the minimum and maximum possible number of Name elements?
2) what are the minimum and maximum possible number of Snack elements?

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="PassengerInfo">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Name" type="xs:string" maxOccurs="2"/>
        <xs:element name="Seat" type="xs:string"/>
        <xs:choice>
          <xs:element name="Meal" type="xs:string"/>
          <xs:element name="Snack" type="xs:string" maxOccurs="2"/>
        </xs:choice>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
Create table students
(Sid char(8) primary key, 
Name varchar(30), 
Dept char(4)
Hobbies XMLTYPE)  --- table created

Insert into students values (‘10403050’,’LI Hong’,’CS’,
’<Hobbies><English level>6</English level><piano>8</piano><basketball>well</basketball></Hobbies>’);
XML → SQL (cont.)

Select * from “students”;

<table>
<thead>
<tr>
<th>SID</th>
<th>name</th>
<th>dept</th>
<th>Hobbies</th>
</tr>
</thead>
<tbody>
<tr>
<td>10403050</td>
<td>LI Hong</td>
<td>CS</td>
<td>&lt;hobbies&gt;&lt;English level&gt;6&lt;/English level&gt;&lt;piano&gt;8&lt;/piano&gt;&lt;basketball&gt;well&lt;/basketball&gt;&lt;/hobbies&gt;</td>
</tr>
</tbody>
</table>

Select XMLELEMENT(“students”, XMLFOREST(name as “StudentName, dept as “Department”)) from “students”;

XMLELEMENT(“students”, XMLFOREST(name as “StudentName, dept as “Department”

<students><StudentName>LI Hong</StudentName><Department/cs</Department></students>