SEMI-STRUCTURED DATA

• **ER, Relational, ODL data models are all based on schema**
  • Structure of data is rigid and known in advance
  • Efficient implementation and various storage and processing optimizations

• **Semistructured data is schemaless**
  • Flexible in representing data
  • Different objects may have different structure and properties
  • Self-describing (data is describing itself)
  • Harder to optimize and efficiently implement
## RELATIONAL MODEL FOR MOVIE DB

### Collection of records (tuples)

<table>
<thead>
<tr>
<th>title</th>
<th>year</th>
<th>studioName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>Fox</td>
</tr>
<tr>
<td>Gone With the Wind</td>
<td>1939</td>
<td>MGM</td>
</tr>
<tr>
<td>Wayne’s World</td>
<td>1992</td>
<td>Paramount</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carrie Fisher</td>
<td>123 Maple St., Hollywood</td>
</tr>
<tr>
<td>Mark Hamill</td>
<td>456 Oak Rd., Brentwood</td>
</tr>
<tr>
<td>Harrison Ford</td>
<td>789 Palm Dr., Beverly Hills</td>
</tr>
</tbody>
</table>

### Stars-in Relationship

<table>
<thead>
<tr>
<th>title</th>
<th>year</th>
<th>starName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>Carrie Fisher</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>Mark Hamill</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>Harrison Ford</td>
</tr>
<tr>
<td>Gone With the Wind</td>
<td>1939</td>
<td>Vivien Leigh</td>
</tr>
<tr>
<td>Wayne’s World</td>
<td>1992</td>
<td>Dana Carvey</td>
</tr>
<tr>
<td>Wayne’s World</td>
<td>1992</td>
<td>Mike Meyers</td>
</tr>
</tbody>
</table>
SEMI-STRUCTURED MODEL

Collection of nodes

- Leaf nodes contain data
- Internal nodes represent either objects or attributes
- Each link is either an attribute link or relationship link
XML

• XML: Extensible Markup Language

• XML is a tag-based notation (language) to describe data

• **XML has two modes**
  - **Well-formed XML** --- No Schema at all
  - **Valid XML** --- governed by DTD (Document Type Definition)
    • Allows validation and more optimizations and pre-processing

```xml
<Books>
  <Book ISBN="0553212419">
    <title>Sherlock Holmes: Complete Novels...</title>
    <author>Sir Arthur Conan Doyle</author>
  </Book>
  <Book ISBN="0743273567">
    <title>The Great Gatsby</title>
    <author>F. Scott Fitzgerald</author>
  </Book>
  <Book ISBN="0684826976">
    <title>Undaunted Courage</title>
    <author>Stephen E. Ambrose</author>
  </Book>
  <Book ISBN="0743203178">
    <title>Nothing Like It In the World</title>
    <author>Stephen E. Ambrose</author>
  </Book>
</Books>
```

XML document
HTML TAGS VS. XML TAGS

• HTML tags describe structure/presentation

<h1> Bibliography </h1>
<p> <i> Foundations of Databases </i> 
    Abiteboul, Hull, Vianu 
    <br> Addison Wesley, 1995 
</p>
<p> <i> Data on the Web </i> 
    Abiteboul, Buneman, Suciu 
    <br> Morgan Kaufmann, 1999 
</p>

Bibliography

Foundations of Databases, Abiteboul, Hull, Vianu
Addison Wesley, 1995

Data on the Web, Abiteboul, Buneman, Suciu
Morgan Kaufmann, 1999
HTML TAGS VS. XML TAGS (CONT’D)

- XML tags describe content (have semantics)

```xml
<bibliography>
  <book>
    <title> Foundations... </title>
    <author> Abiteboul </author>
    <author> Hull </author>
    <author> Vianu </author>
    <publisher> Addison Wesley </publisher>
    <year> 1995 </year>
  </book>
  ...
</bibliography>
```
XML TERMINOLOGY

• **tags**: book, title, author, ...
• **start tag**: `<book>`, **end tag**: `</book>`
• **elements**: `<book>...</book>,<author>...</author>`
• elements are nested
• empty element: `<red></red> abbrv. `<red/>`
• an XML document: single *root element*

*Well-formed* XML document: if it has matching tags
XML: ATTRIBUTES

Inside the start tag

<book price = “55” currency = “USD”>
  <title> Foundations of Databases </title>
  <author> Abiteboul </author>
  ...
  <year> 1995 </year>
</book>

Attributes are alternative ways to represent data
XML Example:

```xml
<?xml version = "1.0" encoding = "utf-8" standalone = "yes" ?>
<StarMovieData>
  <Star starID = "cf" starredIn = "sw">
    <Name>Carrie Fisher</Name>
    <Address>
      <Street>123 Maple St.</Street>
      <City>Hollywood</City>
    </Address>
  </Star>
  <Star starID = "mh" starredIn = "sw">
    <Name>Mark Hamill</Name>
    <Address>
      <Street>5 Locust Ln.</Street>
      <City>Malibu</City>
    </Address>
  </Star>
  <Movie movieID = "sw" starsOf = "cf", "mh">
    <Title>Star Wars</Title>
    <Year>1977</Year>
  </Movie>
</StarMovieData>
```

**Instructional tag** (the doc. Is XML)

**Root element**

**Sub elements**

**Attributes**

**Standalone means it does not follow a schema (well-formed)**
ATTRIBUTES VS. SUB-ELEMENTS

- Two alternative ways to describe the attributes of an object
- Attributes are also used to define IDs and references

```xml
<? xml version = "1.0" encoding = "utf-8" standalone = "yes" ?>
<StarMovieData>
  <Star>
    <Name>Carrie Fisher</Name>
    <Address>
      <Street>123 Maple St.</Street>
      <City>Hollywood</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>1977</Year>
  </Movie>
  <Movie>
    <Title>Star Wars</Title>
    <Year>1977</Year>
    <Movie year = 1977><Title>Star Wars</Title></Movie>
  </Movie>
</StarMovieData>
```
ATTRIBUTES VS. SUB-ELEMENTS

- **Elements**:
  - Basic building blocks of XML
  - Contain content which can be a structure

- **Attributes**
  - Specify additional information about an element.
  - Contain only simple type content

- Some data could be either an Element or an Attribute (so you need standards on how to decide which to use).
XML: ID AND IDREF

- In XML document they appear like any other attribute
- ID and IDREF are formally defined in DTD or XML Schema

```xml
<? xml version = "1.0" encoding = "utf-8" standalone = "yes" ?>
<StarMovieData>
  <Star starID = "cf" starredIn = "sw">
    <Name>Carrie Fisher</Name>
    <Address>
      <Street>123 Maple St.</Street>
      <City>Hollywood</City>
    </Address>
    <Address>
      <Street>5 Locust Ln.</Street>
      <City>Malibu</City>
    </Address>
  </Star>
  <Star starID = "mh" starredIn = "sw">
    <Name>Mark Hamill</Name>
    <Address>
      <Street>456 Oak Rd.</Street>
      <City>Brentwood</City>
    </Address>
  </Star>
  <Movie movieID = "sw" starsOf = "cf", "mh">
    <Title>Star Wars</Title>
    <Year>1977</Year>
  </Movie>
</StarMovieData>
```
XML NAMESPACES

• Tags may have namespaces
  • They define where the tag is defined (its format or structure)
• Namespace format ➔ xmlns:<name>=…

```
<book xmlns:isbn="www.isbn-org.org/def">
  <title> … </title>
  <number> 15 </number>
  <isbn:number> …. </isbn:number>
</book>
```
XML NAMESPACES

• syntactic: `<number>`, `<isbn:number>`
• semantic: provide URL for “shared” schema

```xml
<tag xmlns:mystyle = "http://…”>
    ...
    <mystyle:title> ... </mystyle:title>
    <mystyle:number> ... </mystyle:number>
</tag>
```
COVERED SO FAR…

• What are XML documents

• XML Structure
  • Tags, start and end tags, elements, attributes

• XML Types
  • Well-formed XML (No schema)
  • Valid XML (has a schema)
XML Schema
XML SCHEMA

• An XML document is usually (but not always) validated by an XML Schema.

• The XML Schema provides the information on whether the XML document “followed the rules” set up in the XML Schema.

• An XML Schema is an agreement between the sender and the receiver of a document as to the structure of that document.

Two mechanisms:

- Document Type Definition (DTD)
- XML Schema
XML SCHEMA

Element and Attribute declaration:

```
<xsd:element name = "DataTransmission">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref = "FirstName" minOccurs = "0"/>
         <xsd:element ref = "LastName" minOccurs = "0"/>
         <xsd:element ref = "Phone" minOccurs = "0"/>
         <xsd:element ref = "Birthdate" minOccurs = "0"/>
         <xsd:element ref = "Gender" minOccurs = "0"/>
         <xsd:element ref = "StreetAddress" minOccurs = "0"/>
         <xsd:element ref = "CityAddress" minOccurs = "0"/>
         <xsd:element ref = "StateCode" minOccurs = "0"/>
         <xsd:element ref = "ZipCode" minOccurs = "0"/>
         <xsd:element ref = "SSN" minOccurs = "0"/>
         <xsd:element name = "SafetyCapDate" type = "xsd:date"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```
EXAMPLE

  <FirstName>Maria</FirstName>
  <LastName>Montes</LastName>
  <Birthdate>1951-11-05</Birthdate>
  <Gender>F</Gender>
  <StreetAddress>1969 Ygnacio Valley Road</StreetAddress>
  <CityAddress>Walnut Creek</CityAddress>
  <StateCode>CA</StateCode>
  <ZipCode>94597</ZipCode>
  <SSN>561-88-9208</SSN>
  <SafetyCapDate>2001-05-22</SafetyCapDate>
</DataTransmission>
Data Types in XML Schema
SIMPLE DATA TYPES IN XML SCHEMA

- Comes with “atomic” simple data types
  - Integer, boolean, date, decimal, string, etc.
- You can build user-defined simple data types
  - Built on the included “atomic” data types
  - Allows declaration of
    - valid values, ranges, Patterns, Length, total digits
    - And more…
  - Attributes or Elements can be of a simple data type (either atomic or user-defined).
EXAMPLE: SIMPLE TYPES

```xml
<xsd:simpleType name = "SevenPlaceInteger">
  <xsd:restriction base = "xsd:integer">
    <xsd:totalDigits value = "7"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name = "GenderType">
  <xsd:restriction base = "xsd:string">
    <xsd:enumeration value = "M"/>
    <xsd:enumeration value = "F"/>
    <xsd:length value = "1"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name = "RelationshipCodeType">
  <xsd:restriction base = "xsd:string">
    <xsd:enumeration value = "self"/>
    <xsd:enumeration value = "spouse"/>
    <xsd:enumeration value = "dependent"/>
    <xsd:enumeration value = "other"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name = "SevenPlacePositiveInteger">
  <xsd:restriction base = "SevenPlaceInteger">
    <xsd:minInclusive value = "0"/>
  </xsd:restriction>
</xsd:simpleType>
```

builds on atomic simple data type
builds on custom simple data type
COMPLEX TYPES IN XML SCHEMA

- Builds a structure of Elements.
- Each subelement is either a simple data type or another structure of Elements.
- Only Elements can be of a complex data type.
- Can be named and reusable or anonymous and used only by a single Element.
- Can be an extension or restriction of another complex type.
EXAMPLE: COMPLEX DATA TYPES

<xsd:complexType name = "AddressType"> declaration of named complex data type
    <xsd:sequence>
        <xsd:element ref = "StreetAddress"/>
        <xsd:element ref = "CityAddress"/>
        <xsd:element ref = "StateCode"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name = "WorkAddress" type = "AddressType"/> association of Element with named complex data type

<xsd:complexType name = "AddressWithCountryType"> new complex data type extends existing complex data type
    <xsd:complexContent>
        <xsd:extension base = "AddressType">
            <xsd:sequence>
                <xsd:element name = "CountryCode" type = "xsd:string"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:element name = "PatientInsurance"> element with anonymous complex data type
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref = "Patient"/>
            <xsd:element ref = "TPMembership" minOccurs = "0" maxOccurs = "unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
MOVIES SCHEMA

1)  <? xml version = "1.0" encoding = "utf-8" ?>
2)  <xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">

3)    <xs:complexType name = "movieType">
4)      <xs:attribute name = "title" type = "xs:string"
         use = "required" />
5)      <xs:attribute name = "year" type = "xs:integer"
         use = "required" />
6)    </xs:complexType>

7)    <xs:element name = "Movies">
8)      <xs:complexType>
9)        <xs:sequence>
10)       <xs:element name = "Movie" type = "movieType"
          minOccurs = "0" maxOccurs = "unbounded" />
11)     </xs:sequence>
12)   </xs:complexType>
13)  </xs:element>

14)  </xs:schema>
<complexType name="Address">
  <sequence>
    <element name="street" type="string"/>
    <element name="city" type="string"/>
  </sequence>
</complexType>

<complexType name="USAddress">
  <complexContent>
    <extension base="Address">
      <sequence>
        <element name="state" type="string"/>
        <element name="zip" type="positiveInteger"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
Keys in XML Schema
KEYS IN XML SCHEMA

• Elements in XML can have keys (unique identifiers)
  • Keys can be attributes or subelements
  • A key can be a single field or multiple fields

• Key fields (attributes or subelements) cannot be missing

• Keys are defined in XML schema using special syntax

• Attributes do not have keys
KEYS IN XML SCHEMA

XML Schema for Key:

```xml
<key name="NumKey">
  <selector xpath="parts/part"/>
  <field xpath="@number"/>
</key>
```

- **Key**: give a name to the key
- **Selector**: following the selector xpath starting from the root, it will return a list of objects
- **Field**: in the returned objects, the xpath defined in `field' has to be unique
  - `@ symbol` refers to attributes
KEYS IN XML SCHEMA

• In general, the key syntax is:

```xml
<key name="someDummyNameHere">
    <selector xpath="p"/>
    <field xpath="p_1"/>
    <field xpath="p_2"/>
    ...
    <field xpath="p_k"/>
</key>
```

All these fields together form the key.
FOREIGN KEYS IN XML SCHEMA

• Foreign key syntax:

```
<keyref name="personRef" refer="fullName">
  <selector xpath="./personPointer"/>
  <field xpath="@first"/>
  <field xpath="@last"/>
</keyref>
```
EXAMPLE: MOVIE SCHEMA

1) <!-- xml version = "1.0" encoding = "utf-8" -->
2) <xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">

3) <xs:element name = "Stars">

4)   <xs:complexType>
5)     <xs:sequence>
6)       <xs:element name = "Star" minOccurs = "1"
7)           maxOccurs = "unbounded">
8)         <xs:complexType>
9)           <xs:sequence>
10)          <xs:element name = "Name"
11)             type = "xs:string" />
12)          <xs:element name = "Address"
13)             type = "xs:string" />
14)          <xs:element name = "StarredIn"
15)             minOccurs = "0"
16)             maxOccurs = "unbounded">
17)     </xs:sequence>
18) </xs:complexType>

19) <xs:complexType>
20) <xs:sequence>
21) </xs:sequence>

22) <xs:complexType>
23) <xs:attribute name = "title"
24)             type = "xs:string" />
25) <xs:attribute name = "year"
26)             type = "xs:integer" />
27) </xs:complexType>

28) </xs:element>
EXAMPLE: STARS SCHEMA

1) <? xml version = "1.0" encoding = "utf-8" ?>
2) <xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">

3)   <xs:element name = "Stars">
4)     
5)         <xs:complexType>
6)             <xs:sequence>
7)                 <xs:element name = "Star" minOccurs = "1"
8)                     maxOccurs = "unbounded">
9)                     
10)                     <xs:complexType>
11)                     <xs:sequence>
12)                     
13)                     <xs:element name = "Name"
14)                         type = "xs:string" />
15)                     
16)                     <xs:element name = "Address"
17)                         type = "xs:string" />
18)                     
19)                     <xs:element name = "StarredIn"
20)                         minOccurs = "0"
21)                         maxOccurs = "unbounded">
22)                         
23)                         <xs:complexType>
24)                         <xs:attribute name = "title"
25)                             type = "xs:string" />
26)                         
27)                         <xs:complexType>
28)                         <xs:attribute name = "year"
29)                             type = "xs:integer" />
30)                     
31)                     </xs:complexType>
32)                     </xs:sequence>
33)                     </xs:complexType>
34)                     </xs:sequence>
35)     </xs:complexType>
36)   </xs:element>
37) </xs:schema>
Using XML Schema
USING XML SCHEMA

- Source database
- Extract program
- XML Schema
- XML Document
- Target XML database
- Parse program
- XML Schema

Putting the data in XML documents following the given schema

Parsing the document and validating it against the schema
XML Schemas can build on each other to provide reusability.

Statecode.xsd

Base Definitions.xsd

Patient Search Request.xsd

Patient Search Response.xsd

Patient Update Request.xsd

<xsd:include schemaLocation = "StateCodes.xsd"/>

<xsd:include schemaLocation = "BaseDefinitions.xsd"/>

<xsd:include schemaLocation = "BaseDefinitions.xsd"/>
The Structure of an XML Schema

- Elements in an XML Schema are hierarchical.
- To expand the hierarchy with this tool (Tibco’s XML Authority), click here.
This is the result you get – you can now see the elements that make up the structure of the the `OtherDrugTaken` element.
XML Model vs. Relational Model
Database architecture is relational:
- Normalized to eliminate data redundancy
- Join on any two columns that have the same data type.
- Foreign keys can enforce data integrity
Relational metadata is stored in the database

- Database control tables fully define the structure of the database.
- Without the DBMS metadata the contents of the database are worthless.
- Completely self-contained (not reusable)
- Tables are structured, each column is a “bucket” for a specific kind of data
- In most databases, the metadata does not include descriptions, so a Data Dictionary is necessary.
XML METADATA – THE DOCUMENT

- Metadata built into the document
  - Every element has a tag to tell you where the data is stored in the document.
  - Descriptive tags give structure to the document and tell you what the data means (sort of).
  - Document cannot be parsed for storage on its own. What else is needed?…
An XML Schema (or DTD) is needed to:
- Provide standardization (basis of agreement)
- Allow meaningful parsing and data storage
- Specify agreement on document structure

A data dictionary is still necessary to provide definition for Elements and Attributes
**COMPARISON**

**RDBMS**
- Relationships among items is explicitly defined
- General-purpose storage and processing systems
- Good for general-purpose queries asking for different objects
- Easy to optimize for storage and querying
- Straightforward to export to XML

**XML**
- Relationships among items inferred by position
- Used for data exchange and with XSLT for web visualization
- Good for partitioned data and for retrieving objects with their all sub-components
- Harder to optimize for storage and querying
- Usually not straightforward