DTD and XML Schema

© Leonidas Fegaras
University of Texas at Arlington

DTD: Document Type Definition

- A DTD imposes a structure on an XML document
- Not quite a typing system
  - it is purely syntactic
  - now replaced by XML Schema
- Uses regular expressions to specify structure
  - firstname — an element with tag name firstname
  - book* — zero or more books
  - year? — an optional year
  - firstname, lastname — a firstname followed by lastname
  - book | journal — either a book or a journal

Example of an XML Document

```
<?xml version=”1.0” encoding=”UTF-8”?>
<bib>
  <vendor id="id0_1">
    <name>Amazon</name>
    <email>webmaster@amazon.com</email>
    <book>
        <title>Unix Network Programming</title>
        <publisher>Addison Wesley</publisher>
        <year>1995</year>
        <author>
            <firstname>Richard</firstname>
            <lastname>Stevens</lastname>
        </author>
        <price>38.68</price>
    </book>
    <book>
        <title>An Introduction to Object-Oriented Design</title>
        <publisher>Addison Wesley</publisher>
        <year>1996</year>
        <author>
            <firstname>Jo</firstname>
            <lastname>Levin</lastname>
        </author>
        <author>
            <firstname>Harold</firstname>
            <lastname>Perry</lastname>
        </author>
        <price>11.55</price>
    </book>
  </vendor>
</bib>
```

The DTD of the XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT bib (vendor)*>
<!ELEMENT vendor (name, email, book*)>
<!ATTLIST vendor id ID #REQUIRED>
<!ELEMENT book (title, publisher?, year?, author+, price)>
<!ELEMENT author (firstname?, lastname)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

Summary of the DTD Syntax

- A tagged element in a DTD is defined by

  <!ELEMENT name e>

  where e is a DTD expression
- If e, e1, e2 are DTD expressions, then so are:
  - EMPTY — empty content
  - ANY — any well-formed XML element
  - #PCDATA — any text
  - *tagname* — an element with tag name *tagname*
  - e1,e2 — e1 followed by e2
  - e1 | e2 — either e1 or e2
  - e* — zero or more occurrences of e
  - e+ — one or more occurrences of e
  - e? — optional e (zero or one occurrence)
  - (e)
- Note: tagged elements are global
  - can only be defined once in a DTD

DTD Syntax (cont.)

- Attribute specification:

  <!ATTLIST name (attribute-name type accuracy?)+>

  type is:

  - ID — must be unique within the document
  - IDREF — a reference to an existing ID
  - IDREFS — multiple IDREFs
  - CDATA — any string
  - NMTOKEN — an XML tag name

  accuracy is #REQUIRED, #IMPLIED, #FIXED 'value', value 'v1 ... vn'

- Note: ID, IDref, and IDrefs attributes are not typed!
- Example:

```
<!ELEMENT person (#PCDATA)>
<!ATTLIST person
        id   ID   #REQUIRED
        children   IDrefs   #IMPLIED >
```

  the id attribute is required while the children attribute is optional

Connecting an XML document to a DTD

- In-line the DTD into the XML file:

```
<?xml version="1.0"?>
<!DOCTYPE db [
   <!ELEMENT person ...>
   ...
]>
<db>
   <person> ... </person>
   ...
</db>
```

DTD

XML data

- Better: put the DTD in a separate file and reference it by URL:

```
<?xml version="1.0"?>
<!DOCTYPE db SYSTEM "http://lambda.uta.edu/person.dtd">
<db> ...
```

- Documents are validated against their DTDs during parsing

Recursive DTDs

We want to capture a person with a mother and a father

- First attempt:

```
<!ELEMENT person (name, address, person, person)>
```

where the first person is the mother while the second is the father

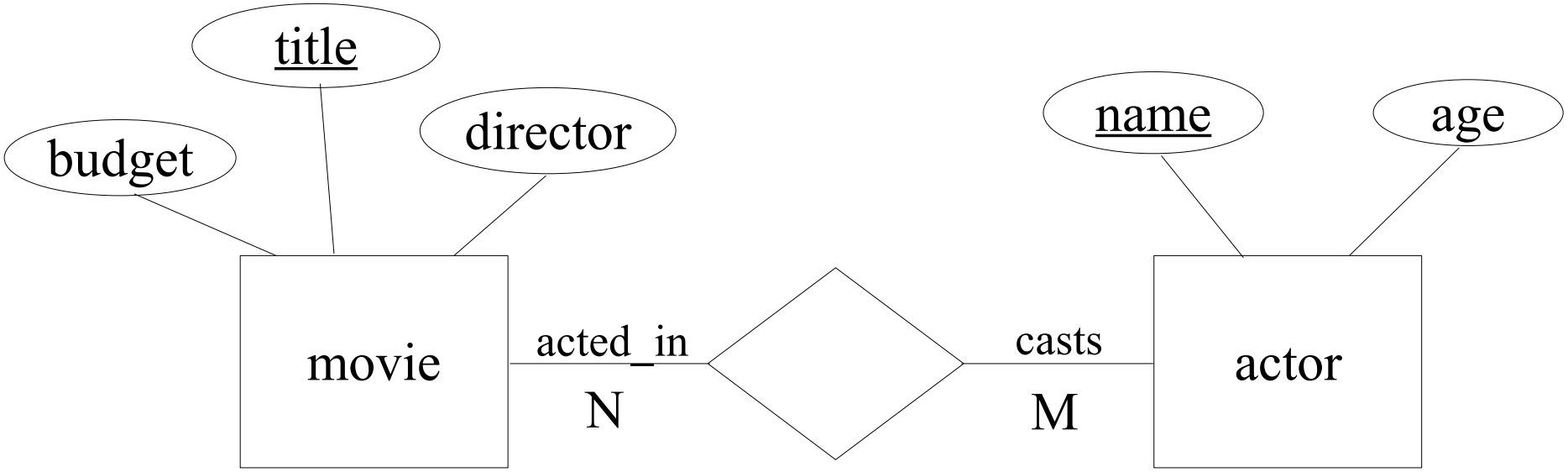
- Second attempt:

```
<!ELEMENT person (name, address, person?, person?)>
```

- Third attempt:

```
<!ELEMENT person (name, address)>
<!ATTLIST person
    id          ID      #REQUIRED
    mother      IDREF   #IMPLIED
    father      IDREF   #IMPLIED>
```

An Entity-Relationship Diagram



The XML Document

```
<db>
  <movie id="m1" casts="a1 a3">
      <title>Waking Ned Divine</title>
      <director>Kirk Jones III</director>
      <budget>100,000</budget>
  </movie>
  <movie id="m2" casts="a2 a9 a21">
      <title>Dragonheart</title>
      <director>Rob Cohen</director>
      <budget>110,000</budget>
  </movie>
  <movie id="m3" casts="a1 a8">
      <title>Moondance</title>
      <director>Dagmar Hirtz</director>
      <budget>90,000</budget>
  </movie>
  <actor id="a1" acted_in="m1 m3 m78">
      <name>David Kelly</name>
      <age>55</age>
  </actor>
  <actor id="a2" acted_in="m2 m9 m11">
      <name>Sean Connery</name>
      <age>68</age>
  </actor>
  <actor id="a3" acted_in="m1 m35">
      <name>Ian Bannen</name>
      <age>45</age>
  </actor>
   :
</db>
```

The DTD

```
<!ELEMENT db (movie+, actor+)>
<!ELEMENT movie (title, director, budget)>
   <!ATTLIST movie id ID #REQUIRED
                   casts IDREFS #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT director (#PCDATA)>
<!ELEMENT budget (#PCDATA)>
<!ELEMENT actor (name, age, directed*)>
  <!ATTLIST actor id ID #REQUIRED
                  acted_in IDREFS #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT age (#PCDATA)>
<!ELEMENT directed (#PCDATA)>
```

XML Namespaces

- When merging multiple docs together, name collisions may occur
- A namespace is a mechanism for uniquely naming tagnames and attribute names to avoid name conflicts
- Tag/attribute names are now *qualified names* (QNames)

  (namespace ':')? localname

  example: bib:author
- A document may use multiple namespaces
- A DTD has its own namespace in which all names are unique
- A namespace in an XML doc is defined as an attribute:

  xmlns:bib="http://lambda.uta.edu/biblio"

  where bib is the namespace name and the URI is a unique identifier
- The default namespace is defined as

  xmlns="URI"

  If not defined, it is the global namespace
- The URI should not be an actual file

Example

```
<item xmlns="http://www.acme.com/supplies"
        xmlns:toy="http://www.acme.com/toys">
<name>backpack</name>
<feature>
  <toy:item>
    <toy:name>cyberpet</toy:name>
  </toy:item>
</feature>
</item>
```

- Alternatively:

```
<item xmlns="http://www.acme.com/supplies">
<name>backpack</name>
<feature>
  <item xmlns="http://www.acme.com/toys">
    <name>cyberpet</name>
  </item>
</feature>
</item>
```

XML Schema

- Replaced DTD
  - less cryptic but more verbose than DTDs
- Specifies Object-Oriented schemas
- Supports a richer set of basic types
- Provides better ways of deriving new type declarations from old ones
  - type extensions and restrictions
- Database-style key concept
- Uses namespaces
- XML Schema primer: http://www.w3.org/TR/xmlschema-0/
- XML Schema document:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
...
</xs:schema>
```

Example

- XML Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="description" type="xs:string" />
</xs:schema>
```

- Matches the XML doc:

```
<?xml version="1.0" encoding="UTF-8"?>
<description>This is an XML doc</description>
```

- xs:string is an *atomic type*

Simple Types Built into XML Schema

- string, int, short, long, float, double, decimal, byte, Boolean
- date

  2008-02-27
- time

  12:20:35
- DateTime

  1999-05-31T13:20:00.000-05:00
- duration

  P10Y1M5T1H20M5S

  10years+1month+5days+1hour+20'+5"
- binary
- anyURI
- QName (see namespaces)
- Name and NCName

xsd:simpleType

- You can define your own simple types using xsd:simpleType

```
<xsd:simpleType name="myInteger">
  <xsd:restriction base="xsd:int">
      <xsd:minInclusive value="10000" />
      <xsd:maxInclusive value="99999" />
  </xsd:restriction>
</xsd:simpleType>
```

- Another example:

```
<xsd:simpleType name="ssn">
  <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d{3}-\d{2}-\d{4}" />
  </xsd:restriction>
</xsd:simpleType>
```

Simple Type Restrictions

- Syntax:

```
<xsd:simpleType name="a-new-simple-type">
  <xsd:restriction base="some-defined-simple-type">
     <xsd:restriction value="some-value" />
  </xsd:restriction>
</xsd:simpleType>
```

- Common restrictions:
  - length — the length of the string or the list
  - minLength
  - maxLength
  - minInclusive — up to and including the given number
  - maxInclusive — down to and including the given number
  - minExclusive — up to but not including the given number
  - maxExclusive — down to but not including the given number
  - choice — the string must be one of the listed choices
  - pattern — the string must match the given pattern
  - encoding — for binary (eg, base64)

Other Simple Type Restrictions

- Enumeration:

```
<xsd:simpleType name="animal">
  <xsd:restriction base="xsd:string">
      <xsd:enumeration value="dog" />
      <xsd:enumeration value="cat" />
  </xsd:restriction>
</xsd:simpleType>
```

- List:

```
<xsd:simpleType name="animals">
   <xsd:list  itemType="animal" />
</xsd:simpleType>
```

- Union:

```
<xs:simpleType name="animalOrPerson">
      <xs:union  memberTypes="animal person" />
</xs:simpleType>
```

Complex Types

- Syntax:

  <xsd:complexType name="*a-new-complex-type*">

  ...

  </xsd:complexType>

- Inside xsd:complexType:
  - Any content

    <xsd:any>
  - Concatenation (A1, A2, ..., An)

    <xsd:sequence> A1 A2 ... An </xsd:sequence>
  - Alternation (A1 | A2 | ... | An)

    <xsd:choice> A1 A2 ... An </xsd:choice>
  - All, but in any order (A1 & A2 & ... & An)

    <xsd:all> A1 A2 ... An </xsd:all>

- Alternative syntax:

  <xsd:complexType name="*a-new-complex-type*" type="*some-type*" />

Elements

- Example:

```
<xsd:element name='employee'>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name='name' type='xsd:string'/>
      <xsd:element name='address' type='xsd:string'/>
      <xsd:element name='ssn' type='xsd:string'/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

- It corresponds to the DTD:

```
<!ELEMENT employee (name,address,ssn)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT ssn (#PCDATA)>
```

Element Attributes

- Element attributes can only appear inside xsd:complexType in an xsd:element

```
<xsd:attribute name="ssn" type="ssn" use="required"/>
<xsd:attribute name="father" type="xsd:string" use="optional" />
```

- Example:

```
<xsd:complexType name="USAddress" >
  <xsd:sequence>
    <xsd:element  name="name"  type="xsd:string"/>
    <xsd:element  name="street"  type="xsd:string"/>
    <xsd:element  name="city"  type="xsd:string"/>
    <xsd:element  name="state"  type="xsd:string"/>
    <xsd:element  name="zip"  type="xsd:decimal"/>
  </xsd:sequence>
  <xsd:attribute  name="country"  type="xsd:NMTOKEN"  fixed="US"/>
</xsd:complexType>
```

Keys

- Must be defined inside a complexType
- Keys can be defined using XPath expressions

```
<xsd:key name="ssn">
    <xsd:selector xpath="dept/person" />
    <xsd:field xpath="@ssn" />
</xsd:key>
```

  - the selector defines the applicable elements
  - the field defines the data that are unique within the applicable elements
  - xsd:unique may be used to define unique combination of values
- Foreign keys are like IDRefs but are well-typed

```
<xsd:keyref name="mother" refer="ssn">
        <xsd:selector xpath="dept/person" />
        <xsd:field xpath="@mother" />
<xsd:keyref>
```

Other Specifiers

- Other attributes of *xsd:element*
  - minOccurs: minimum number of occurrences
    default = “1”
  - maxOccurs: maximum number of occurrences
    default = “1” if minOccurs is not given
    default = “unbounded” otherwise
  - An element is optional when minOccurs=“0” and maxOccurs=“1”
- Unconstrained element content

```
<xsd:element name="anything" type="xsd:anyType"/>
```

- Other attributes of *xsd:complexType*
  - ref = “type-name” (external reference)
  - mixed = “true” (can be mixed content)

Using *xsd:complexType*

```
<xsd:complexType name="Items">
  <xsd:sequence>
    <xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="productName" type="xsd:string"/>
          <xsd:element name="quantity">
            <xsd:simpleType>
              <xsd:restriction base="xsd:positiveInteger">
                <xsd:maxExclusive value="100"/>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:element>
          <xsd:element name="USPrice"  type="xsd:decimal"/>
          <xsd:element ref="comment"   minOccurs="0"/>
          <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
        </xsd:sequence>
        <xsd:attribute name="partNum" type="SKU" use="required"/>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
```

Example from Elmasri-Navathe Textbook

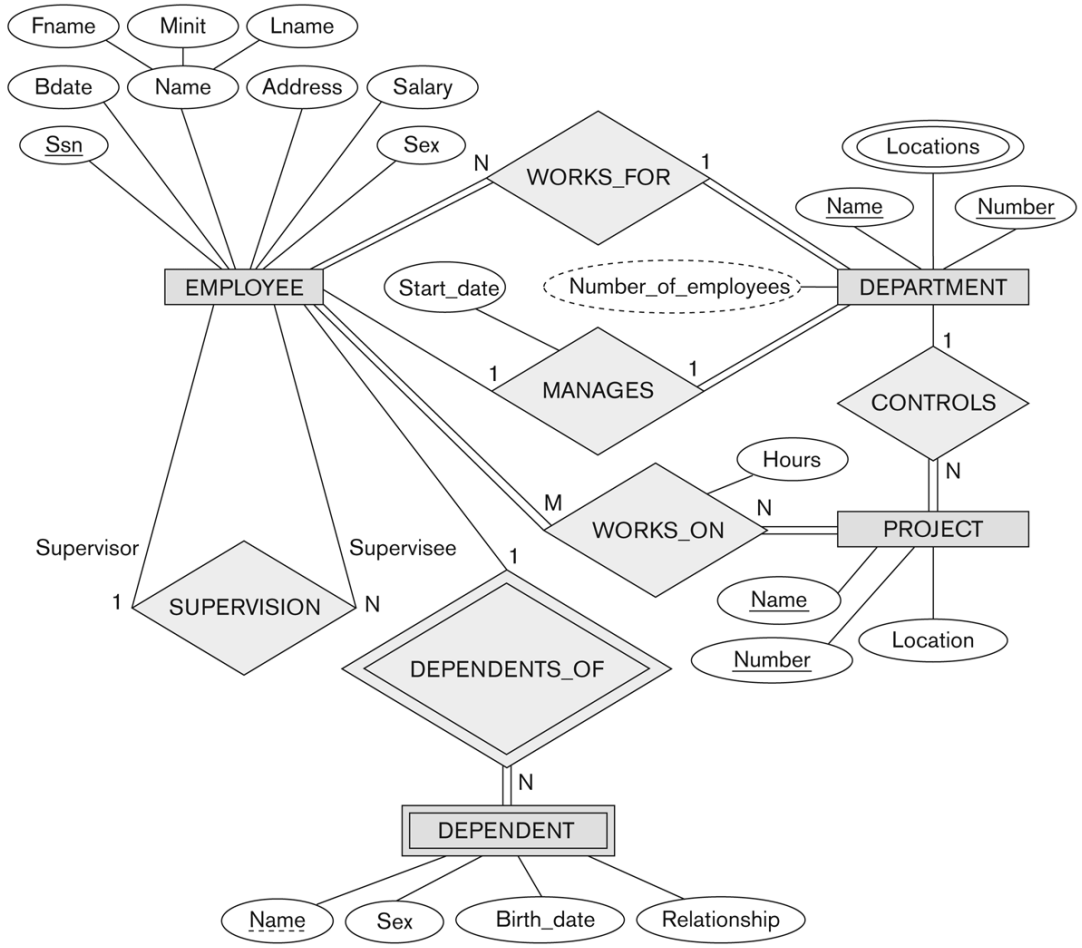


Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

Example

```
<?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>
```

Example (cont.)

```
    <xsd:unique name="departmentNameUnique">
        <xsd:selector xpath="department" />
        <xsd:field xpath="departmentName" />
    </xsd:unique>
    <xsd:unique name="projectNameUnique">
        <xsd:selector xpath="project" />
        <xsd:field xpath="projectName" />
    </xsd:unique>
    <xsd:key name="projectNumberKey">
        <xsd:selector xpath="project" />
        <xsd:field xpath="projectNumber" />
    </xsd:key>
    <xsd:key name="departmentNumberKey">
        <xsd:selector xpath="department" />
        <xsd:field xpath="departmentNumber" />
    </xsd:key>
    <xsd:key name="employeeSSNKey">
        <xsd:selector xpath="employee" />
        <xsd:field xpath="employeeSSN" />
    </xsd:key>
    <xsd:keyref name="departmentManagerSSNKeyRef" refer="employeeSSNKey">
        <xsd:selector xpath="department" />
        <xsd:field xpath="departmentManagerSSN" />
    </xsd:keyref>
    <xsd:keyref name="employeeDepartmentNumberKeyRef"
        refer="departmentNumberKey">
        <xsd:selector xpath="employee" />
        <xsd:field xpath="employeeDepartmentNumber" />
    </xsd:keyref>
    <xsd:keyref name="employeeSupervisorSSNKeyRef" refer="employeeSSNKey">
        <xsd:selector xpath="employee" />
        <xsd:field xpath="employeeSupervisorSSN" />
    </xsd:keyref>
    <xsd:keyref name="projectDepartmentNumberKeyRef"
        refer="departmentNumberKey">
        <xsd:selector xpath="project" />
        <xsd:field xpath="projectDepartmentNumber" />
    </xsd:keyref>
    <xsd:keyref name="projectWorkerSSNKeyRef" refer="employeeSSNKey">
        <xsd:selector xpath="project/projectWorker" />
        <xsd:field xpath="SSN" />
    </xsd:keyref>
    <xsd:keyref name="employeeWorksOnProjectNumberKeyRef"
        refer="projectNumberKey">
        <xsd:selector xpath="employee/employeeWorksOn" />
        <xsd:field xpath="projectNumber" />
    </xsd:keyref>
</xsd:element>
```

```
<xsd:complexType name="Department">
  <xsd:sequence>
    <xsd:element name="departmentName" type="xsd:string" />
    <xsd:element name="departmentNumber" type="xsd:string" />
    <xsd:element name="departmentManagerSSN" type="xsd:string" />
    <xsd:element name="departmentManagerStartDate" type="xsd:date" />
    <xsd:element name="departmentLocation" type="xsd:string"
        minOccurs="0" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Employee">
  <xsd:sequence>
    <xsd:element name="employeeName" type="Name" />
    <xsd:element name="employeeSSN" type="xsd:string" />
    <xsd:element name="employeeSex" type="xsd:string" />
    <xsd:element name="employeeSalary" type="xsd:unsignedInt" />
    <xsd:element name="employeeBirthDate" type="xsd:date" />
    <xsd:element name="employeeDepartmentNumber" type="xsd:string" />
    <xsd:element name="employeeSupervisorSSN" type="xsd:string" />
    <xsd:element name="employeeAddress" type="Address" />
    <xsd:element name="employeeWorksOn" type="WorksOn" minOccurs="1"
        maxOccurs="unbounded" />
    <xsd:element name="employeeDependent" type="Dependent" minOccurs="0"
        maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Project">
  <xsd:sequence>
    <xsd:element name="projectName" type="xsd:string" />
    <xsd:element name="projectNumber" type="xsd:string" />
    <xsd:element name="projectLocation" type="xsd:string" />
    <xsd:element name="projectDepartmentNumber" type="xsd:string" />
    <xsd:element name="projectWorker" type="Worker" minOccurs="1"
        maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Dependent">
  <xsd:sequence>
    <xsd:element name="dependentName" type="xsd:string" />
    <xsd:element name="dependentSex" type="xsd:string" />
    <xsd:element name="dependentBirthDate" type="xsd:date" />
    <xsd:element name="dependentRelationship" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Address">
  <xsd:sequence>
    <xsd:element name="number" type="xsd:string" />
    <xsd:element name="street" type="xsd:string" />
    <xsd:element name="city" type="xsd:string" />
    <xsd:element name="state" type="xsd:string" />
  </xsd:sequence>
```

Example (cont.)

```
    </xsd:complexType>
    <xsd:complexType name="Name">
       <xsd:sequence>
          <xsd:element name="firstName" type="xsd:string" />
          <xsd:element name="middleName" type="xsd:string" />
          <xsd:element name="lastName" type="xsd:string" />
       </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Worker">
       <xsd:sequence>
          <xsd:element name="SSN" type="xsd:string" />
          <xsd:element name="hours" type="xsd:float" />
       </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="WorksOn">
       <xsd:sequence>
          <xsd:element name="projectNumber" type="xsd:string" />
          <xsd:element name="hours" type="xsd:float" />
       </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Using Schemas

- Using *targetNamespace*:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns="http://lambda.uta.edu/PO"
            targetNamespace="http://lambda.uta.edu/PO">
  <xsd:element name="PurchaseOrder">
    <xsd:complexType>
    ...
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

- XML data that conforms to this XML Schema:

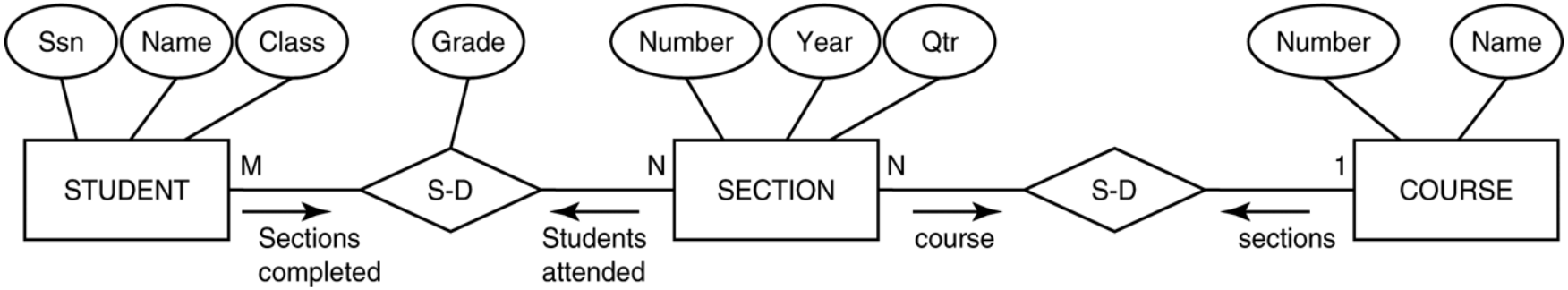
```
<?xml version="1.0"?>
<po:PurchaseOrder xmlns:po="http://lambda.uta.edu/PO">
...
```

Validating an XML Document

```
import javax.xml.validation.SchemaFactory;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.XMLReader;
import java.io.File;
class Validate {
    public static void main ( String args[] ) throws Exception {
        SAXParserFactory factory = SAXParserFactory.newInstance();
        factory.setValidating(false);
        factory.setNamespaceAware(true);
        SchemaFactory schemaFactory =
                SchemaFactory.newInstance("http://www.w3.org/2001/XMLSchema");
        factory.setSchema(schemaFactory.newSchema("mySchema.xsd"));
        XMLReader xmlReader = factory.newSAXParser().getXMLReader();
        xmlReader.parse("myDoc.xml");
    }
}
```

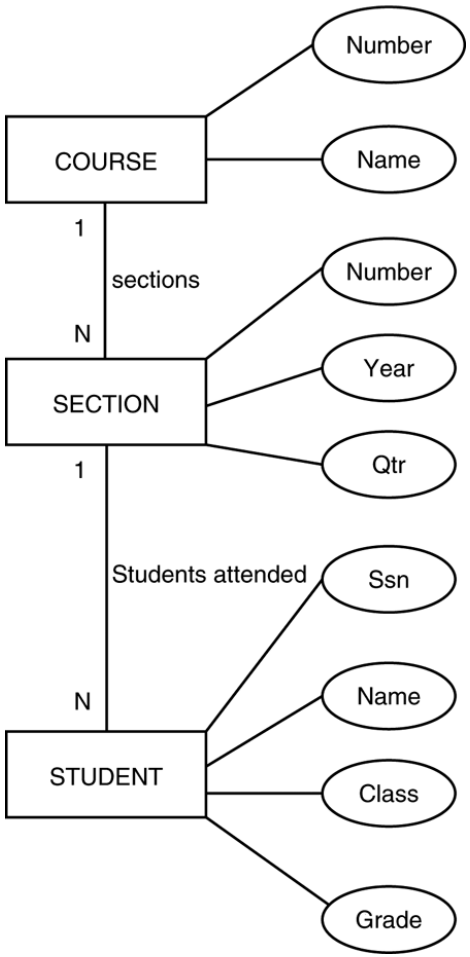
Mapping an XML Design to XML Schema

- Need to convert an Entity-Relationship (ER) diagram into a hierarchical diagram
  - The ER diagram is in general a graph (entities connected with relationships)
  - The XML schema is a forest (one tree per XML document)
- Breaking cycles to convert graphs into trees
  - It is possible to have a more complex subset with one or more cycles, indicating multiple relationships among the entities
  - One way to break the cycles is to replicate the entity types involved in cycles
- Simple example:



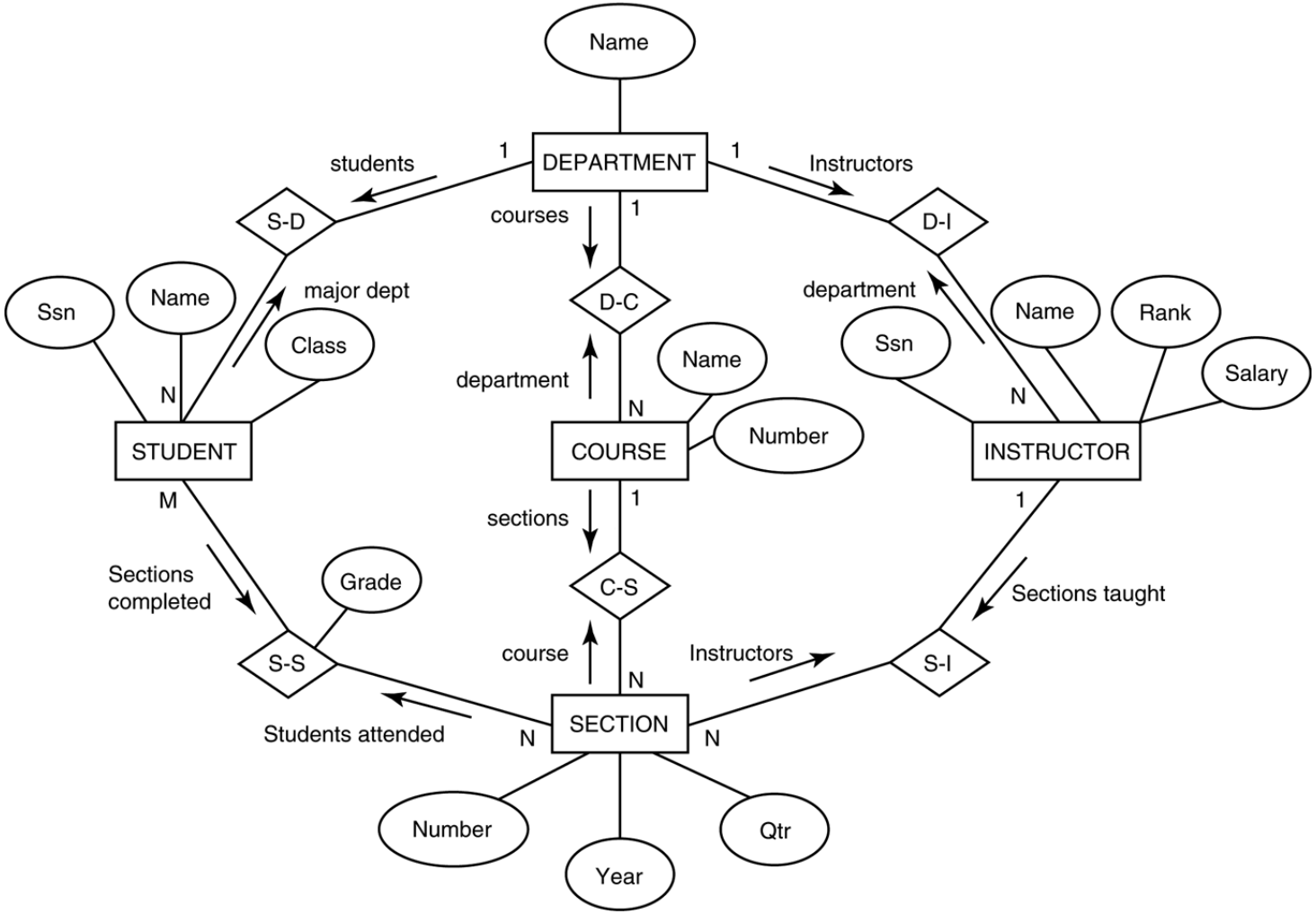
Simple Example

- For this, we only need to choose the root

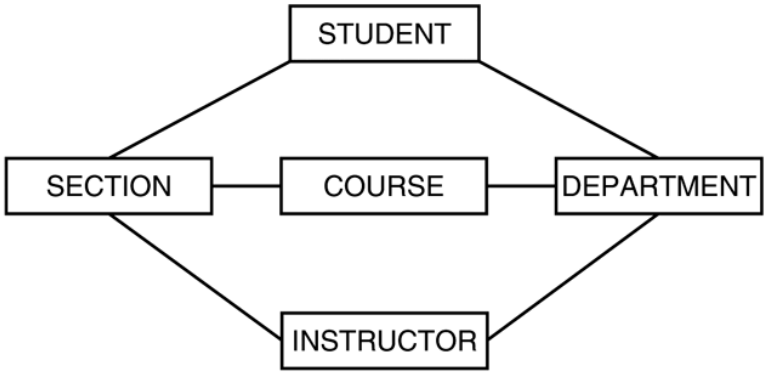



```
<xsd:element name="root">
<xsd:sequence>
<xsd:element name="course" minOccurs="0" maxOccurs="unbounded">
  <xsd:sequence>
    <xsd:element name="cname" type="xsd:string" />
    <xsd:element name="cnumber" type="xsd:unsignedInt" />
    <xsd:element name="section" minOccurs="0" maxOccurs="unbounded">
      <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>
```

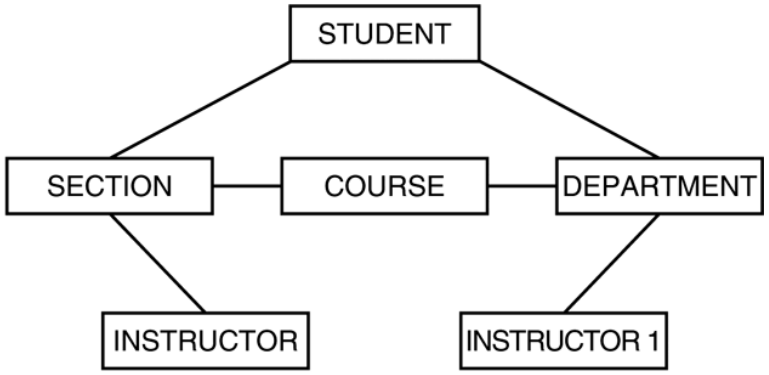
A Complex Example



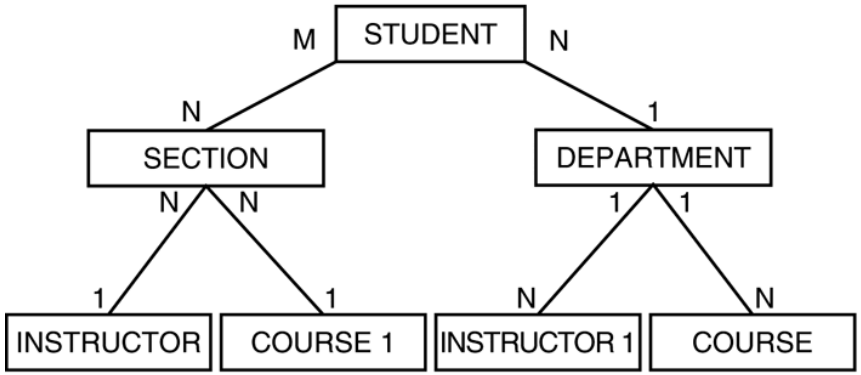
A Complex Example (cont.)



(1)



(2)



(3)