XML and Internet Databases

Chapter Outline

- Introduction
- Structured, Semi structured, and Unstructured Data.
- XML Hierarchical (Tree) Data Model.
- XML Documents, DTD, and XML Schema.
- XML Documents and Databases.
- XML Querying.
  - Xpath
  - XQuery

Introduction

- Although HTML is widely used for formatting and structuring Web *documents*, it is not suitable for specifying *structured data* that is extracted from databases.
- A new language—namely XML (eXtended Markup Language) has emerged as the standard for structuring and exchanging data over the Web. XML can be used to provide more information about the structure and meaning of the data in the Web pages rather than just specifying how the Web pages are formatted for display on the screen.
- The formatting aspects are specified separately—for example, by using a formatting language such as XSL (eXtended Stylesheet Language).

Structured, Semi Structured and Unstructured Data.

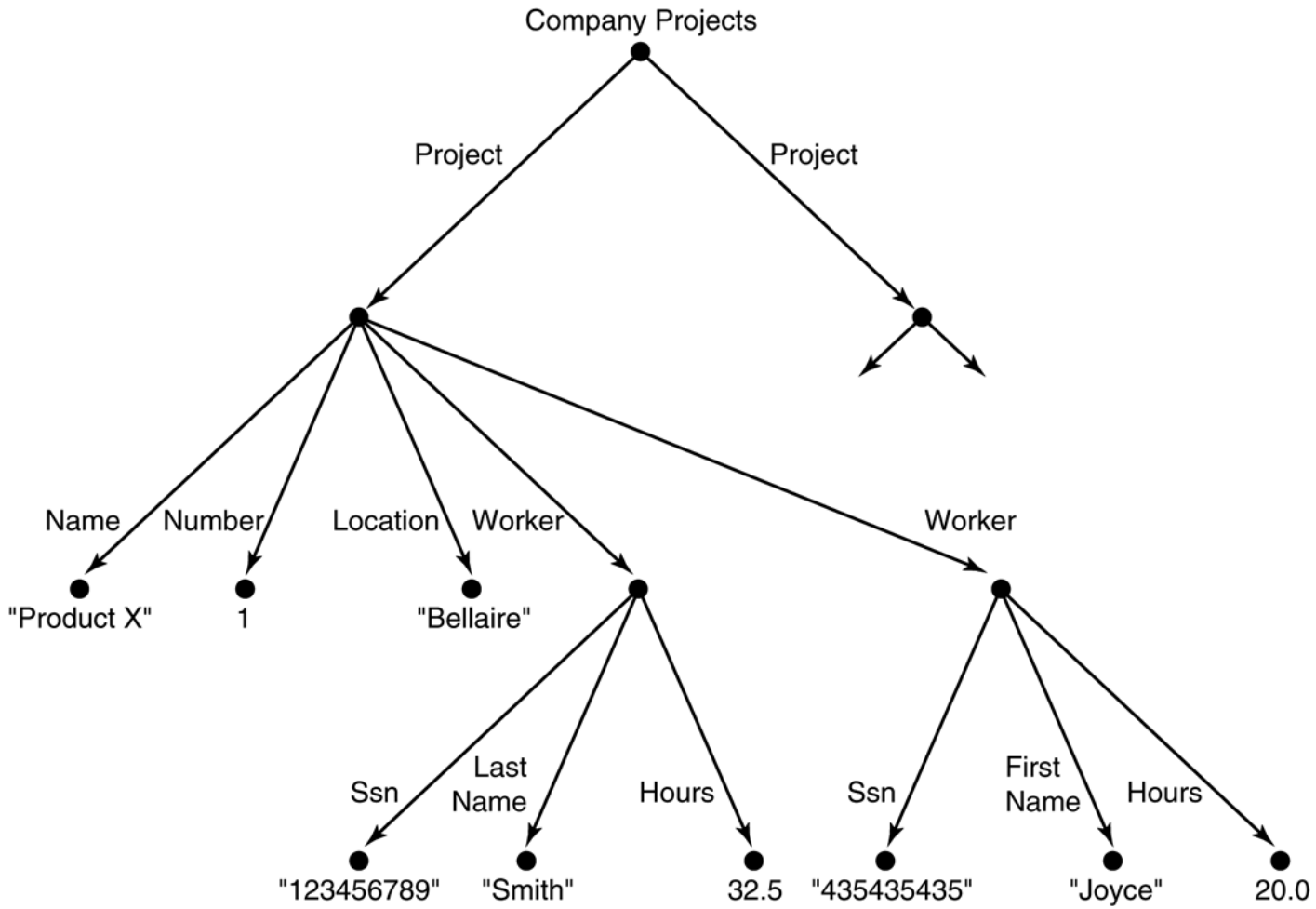
- Information stored in databases is known as **structured data** because it is represented in a strict format. The DBMS then checks to ensure that all data follows the structures and constraints specified in the schema.
- In some applications, data is collected in an ad-hoc manner before it is known how it will be stored and managed. This data may have a certain structure, but not all the information collected will have identical structure. This type of data is known as **semi-structured data.**
  - In semi-structured data, the schema information is *mixed in* with the data values, since each data object can have different attributes that are not known in advance. Hence, this type of data is sometimes referred to as **self-decribing data.**
- A third category is known as **unstructured data,** because there is very limited indication of the type of data. A typical example would be a text document that contains information embedded within it. Web pages in HTML that contain some data are considered as unstructured data.

Structured, Semi Structured and Unstructured Data (cont.)

- Semi-structured data may be displayed as a directed graph, as shown.
- The **labels** or **tags** on the directed edges represent the schema names—the names of attributes, object types (or entity types or classes), and relationships.
- The internal nodes represent individual objects or composite attributes.
- The leaf nodes represent actual data values of simple (atomic) attributes.

FIGURE 26.1

Representing semistructured data as a graph.



XML Hierarchical (Tree) Data Model

FIGURE 26.3
A complex XML element called <projects>.

```
<?xml version="1.0" standalone="yes"?>
<projects>

 <project>
  <Name>ProductX</Name>
  <Number>1</Number>
  <Location>Bellaire</Location>
  <DeptNo>5</DeptNo>
  <Worker>
   <SSN>123456789</SSN>
   <LastName>Smith</LastName>
   <hours>32.5</hours>
  </Worker>
  <Worker>
   <SSN>453453453</SSN>
   <FirstName>Joyce</FirstName>
   <hours>20.0</hours>
  </Worker>
 </project>
 </project>
  <Name>ProductY</Name>
  <Number>2</Number>
  <Location>Sugarland</Location>
  <DeptNo >5</DeptNo >
  <Worker>
   <SSN>123456789</SSN>
   <hours>7.5</hours>
  </Worker>
  <Worker>
   <SSN>453453453</SSN>
   <hours>20.0</hours>
  </Worker>
  <Worker>
   <SSN>333445555</SSN>
   <hours>10.0</hours>
  </Worker>
 </project>

...

</projects>
```

XML Hierarchical (Tree) Data Model (cont.)

- The basic object is XML is the **XML document**. There are two main structuring concepts that are used to construct an XML document: **elements** and **attributes**. Attributes in XML provide additional information that describe elements.
- As in HTML, elements are identified in a document by their **start tag** and **end tag.** The tag names are enclosed between angled brackets <…>, and end tags are further identified by a backslash </…>. **Complex elements** are constructed from other elements hierarchically, whereas **simple elements** contain data values.
- It is straightforward to see the correspondence between the XML textual representation and the tree structure. In the tree representation, internal nodes represent complex elements, whereas leaf nodes represent simple elements. That is why the XML model is called a **tree model** or a **hierarchical model.**

XML Hierarchical (Tree) Data Model (cont.)

It is possible to characterize three main types of XML documents:

1. ***Data-centric XML documents*:**

   These documents have many small data items that follow a specific structure, and hence may be extracted from a structured database. They are formatted as XML documents in order to exchange them or display them over the Web.

2. ***Document-centric XML documents*:**

   These are documents with large amounts of text, such as news articles or books. There is little or no structured data elements in these documents.

3. ***Hybrid XML documents*:**

   These documents may have parts that contains structured data and other parts that are predominantly textual or unstructured.

XML Documents, DTD, and XML Schema.

Well-Formed

- It must start with an XML declaration to indicate the version of XML being used—as well as any other relevant attributes.
- It must follow the syntactic guidelines of the tree model. This means that there should be a ***single root element***, and every element must include a matching pair of start tag and end tag within the start and end tags ***of the parent element***.
- A well-formed XML document is **syntactically correct.** This allows it to be processed by generic processors that traverse the document and create an internal tree representation.
  - **DOM (Document Object Model)** - Allows programs to manipulate the resulting tree representation corresponding to a well-formed XML document. The whole document must be parsed beforehand when using dom.
  - **SAX (Simple API for XML)** - Allows processing of XML documents on the fly by notifying the processing program whenever a start or end tag is encountered.

Valid

- A stronger criterion is for an XML document to be **valid.** In this case, the document must be well-formed, and in addition the element names used in the start and end tag pairs must follow the structure specified in a separate XML **DTD** (**Document Type Definition**) file or XML schema file.

XML Documents, DTD, and XML Schema (cont.)

FIGURE 26.4 An XML DTD file called projects.

```
<!DOCTYPE projects [
  <!ELEMENT projects (project+)>
  <!ELEMENT project (Name, Number, Location, DeptNo?, Workers)>
  <!ELEMENT Name (#PCDATA)>
  <!ELEMENT Number (#PCDATA)>
  <!ELEMENT Location (#PCDATA)>
  <!ELEMENT DeptNo (#PCDATA)>
  <!ELEMENT Workers (Worker*)>
  <!ELEMENT Worker (SSN, LastName?, FirstName?, hours)>
  <!ELEMENT SSN (#PCDATA)>
  <!ELEMENT LastName (#PCDATA)>
  <!ELEMENT FirstName (#PCDATA)>
  <!ELEMENT hours (#PCDATA)>
] >
```

XML Documents, DTD, and XML Schema (cont.)

XML DTD Notation

- A * following the element name means that the element can be repeated zero or more times in the document. This can be called an optional multivalued (repeating) element.
- A + following the element name means that the element can be repeated one or more times in the document. This can be called a required multivalued (repeating) element.
- A ? following the element name means that the element can be repeated zero or one times. This can be called an optional single-valued (non-repeating) element.
- An element appearing without any of the preceding three symbols must appear exactly once in the document. This can be called an required single-valued (non-repeating) element.
- The type of the element is specified via parentheses following the element. If the parentheses include names of other elements, these would be the *children* of the element in the tree structure. If the parentheses include the keyword #PCDATA or one of the other data types available in XML DTD, the element is a leaf node. PCDATA stands for *parsed character data*, which is roughly similar to a string data type.
- Parentheses can be nested when specifying elements.
- A bar symbol (*e1* | *e2*) specifies that either *e1* or *e2* can appear in the document.

XML Documents, DTD, and XML Schema (cont.)

Limitations of XML DTD

- First, the data types in DTD are not very general.
- DTD has its own special syntax and so it requires specialized processors. It would be advantageous to specify XML schema documents using the syntax rules of XML itself so that the same processors for XML documents can process XML schema descriptions.
- Third, all DTD elements are always forced to follow the specified ordering the document so unordered elements are not permitted.

XML Documents, DTD, and XML Schema (cont.)

FIGURE 26.5 An XML schema file called company.

```
<?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>
```

FIGURE 26.5
(continued)
An XML schema file called company.

```
<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>
```

FIGURE 26.5 (continued)
An XML schema file called company.

```
<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>
```

FIGURE 26.5 (continued)
An XML schema file called company.

```
    </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>
```

XML Documents, DTD, and XML Schema (cont.)

XML Schema

- ***Schema Descriptions and XML Namespaces*:**

  It is necessary to identify the specific set of XML schema language elements (tags) by a file stored at a Web site location. The second line in our example specifies the file used in this example, which is: "**http://www.w3.org/2001/XMLSchema**".

  Each such definition is called an **XML namespace.**

  The file name is assigned to the variable xsd using the attribute xmlns (XML namespace), and this variable is used as a prefix to all XML schema tags.

- ***Annotations, documentation, amd language used*:**

  The xsd:annotation and xsd:documentation are used for providing comments and other descriptions in the XML document. The attribute XML:lang of the xsd:documentation element specifies the language being used. Eg. “en”

- ***Elements and types*:**

  We specify the *root element* of our XML schema. In XML schema, the name attribute of the xsd:element tag specifies the element name, which is called company for the root element in our example. The structure of the company root element is a xsd:complexType.

XML Documents, DTD, and XML Schema (cont.)

XML Schema

- ***First-level elements in the company database*:**

  These elements are named **employee, department**, and **project,** and each is specified in an xsd:element tag. If a tag has only attributes and no further sub-elements or data within it, it can be ended with the back slash symbol (/>) and termed **Empty Element.**

- ***Specifying element type and minimum and maximum occurrences:***

  If we specify a type attribute in an xsd:element, this means that the structure of the element will be described separately, typically using the xsd:complexType element. The minOccurs and maxOccurs tags are used for specifying lower and upper bounds on the number of occurrences of an element. The default is exactly one occurrence.

- ***Specifying Keys:***

  For specifying **primary keys,** the tag xsd:key is used.

  For specifying **foreign keys,** the tag xsd:keyref is used. When specifying a foreign key, the attribute refer of the xsd:keyref tag specifies the referenced primary key whereas the tags xsd:selector and xsd:field specify the referencing element type and foreign key.

XML Documents, DTD, and XML Schema (cont.)

XML Schema

- ***Specifying the structures of complex elements via complex types:***

  Complex elements in our example are **Department, Employee, Project, and Dependent,** which use the tag **xsd:complexType**. We specify each of these as a sequence of subelements corresponding to the database attributes of each entity type by using the xsd:sequence and xsd:element tags of XML schema. Each element is given a name and type via the attributes **name** and **type** of xsd:element.

  We can also specify minOccurs and maxOccurs attributes if we need to change the default of exactly one occurrence. For (optional) database attributes where null is allowed, we need to specify minOccurs = 0, whereas for multivalued database attributes we need to specify maxOccurs = “unbounded” on the corresponding element.

- ***Composite (compound) attributes:***

  Composite attributes from ER Schema are also specified as complex types in the XML schema, as illustrated by the Address, Name, Worker, and WorkesOn complex types. These could have been directly embedded within their parent elements.

XML Documents and Databases.

Approaches to Storing XML Documents

- ***Using a dbms to store the documents as text*:**

  We can use a relational or object dbms to store whole XML documents as text fields within the dbms records or objects. This aproach can be used if the dbms has a special module for document processing, and would work for storing schemaless and document-centric XML documents.

- ***Using a dbms to store the document contents as data elements*:**

  This approach would work for storing a collection of documents that follow a specific XML DTD or XML schema. Since all the documents have the same structure, we can design a relational (or object) database to store the leaf-level data elements within the XML documents.

- ***Designing a specialized system for storing native XML data*:**

  A new type of database system based on the hierarchical (tree) model would be designed and implemented. The system would include specialized indexing and querying techniques, and would work for all types of XML documents.

- ***Creating or publishing customized XML documents from pre-existing relational databases*:**

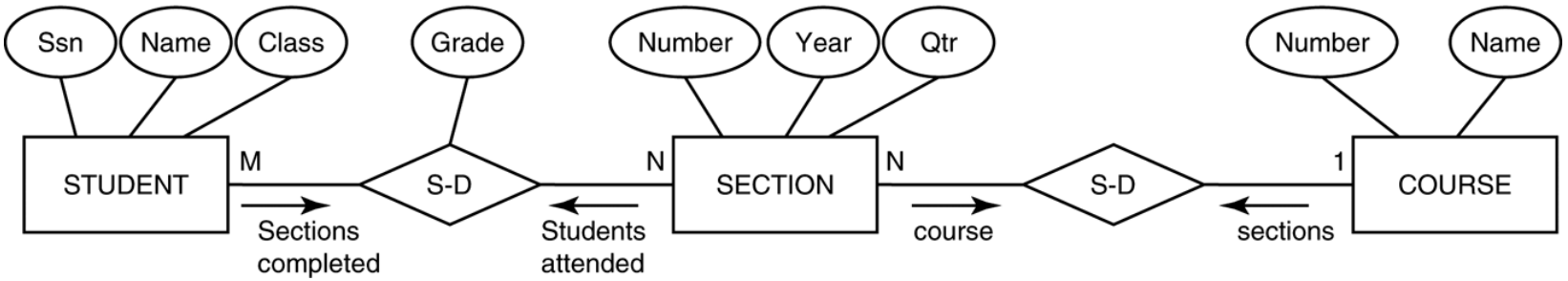
  Because there are enormous amounts of data already stored in relational databases, parts of these data may need to be formatted as documents for exchanging or displaying over the Web.

XML Documents, DTD, and XML Schema (cont.)

Extracting XML Documents from Relational Databases.

Suppose that an application needs to extract XML documents for student, course, and grade information from the university database. The data needed for these documents is contained in the database attributes of the entity types **course, section, and student** as shown below (part of the main ER), and the relationships s-s and c-s between them.

FIGURE 26.7
Subset of the UNIVERSITY database schema needed for XML document extraction.



XML Documents, DTD, and XML Schema (cont.)

Extracting XML Documents from Relational Databases.

One of the possible hierarchies that can be extracted from the database subset could choose **COURSE** as the root.

FIGURE 26.8
Hierarchical (tree) view with COURSE as the root.

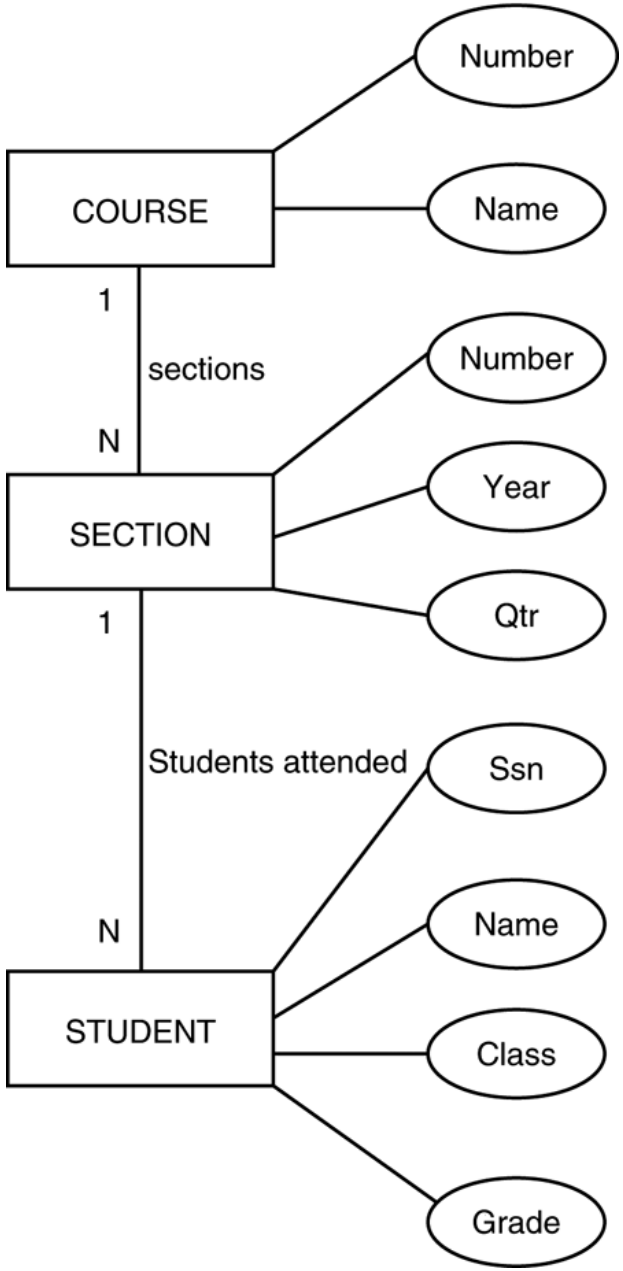


FIGURE 26.9
XML schema document with COURSE as the root.

```
<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>
```

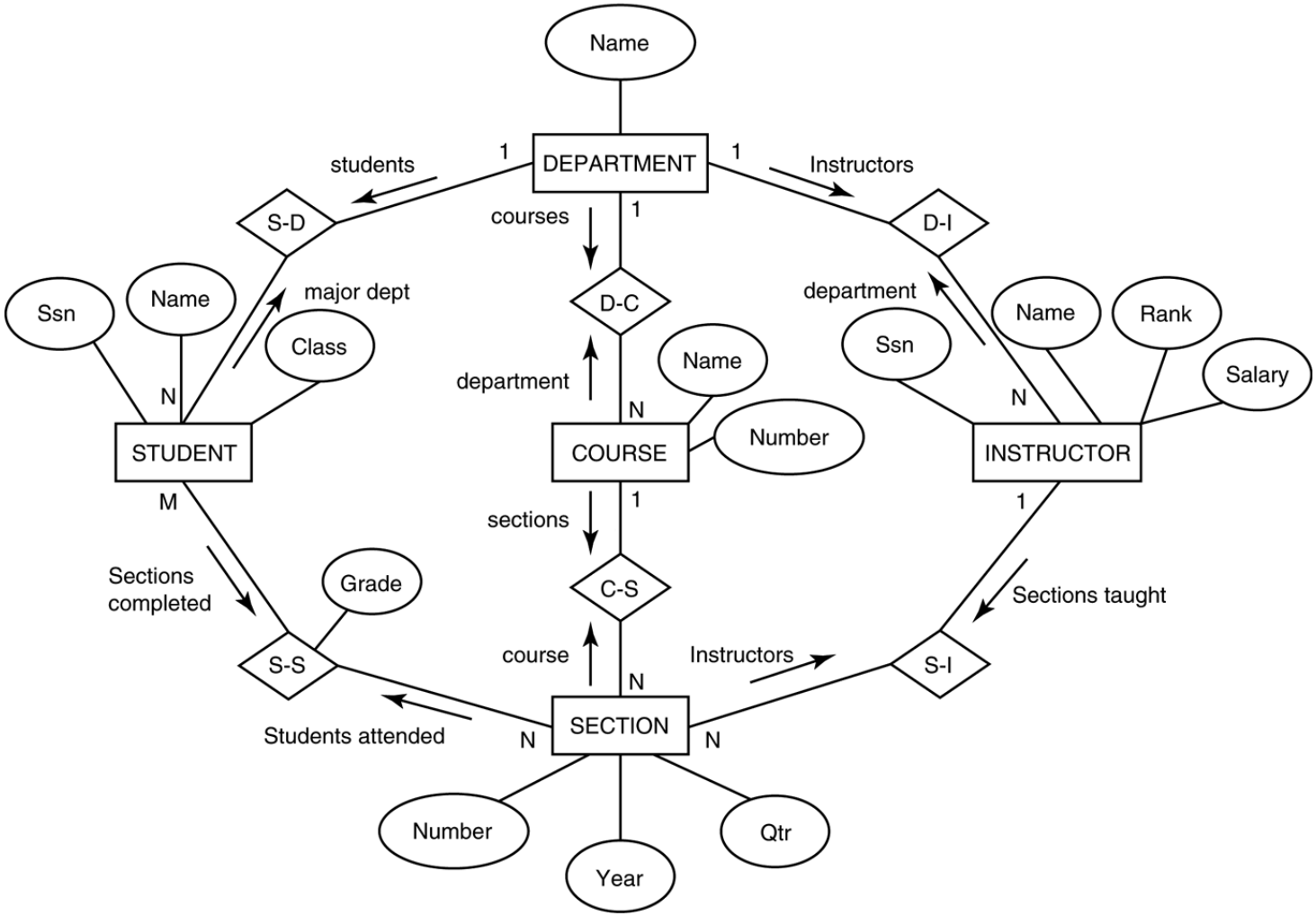
XML Documents, DTD, and XML Schema (cont.)

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.

Suppose that we need the information in all the entity types and relationships in figure below for a particular XML document, with **student** as the root element.

FIGURE 26.6
An ER schema diagram for a simplified UNIVERSITY database.



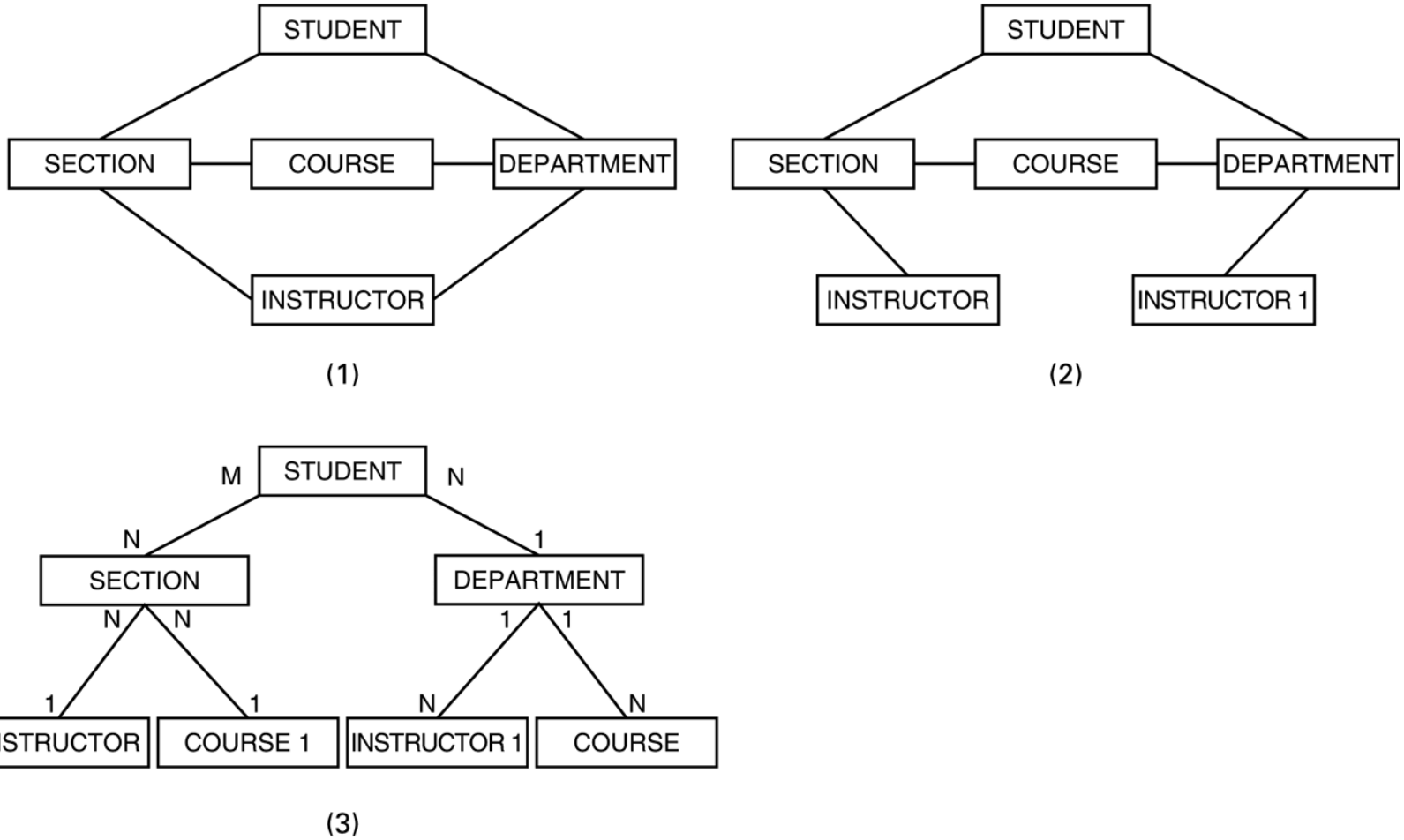
Breaking Cycles To convert Graphs into Trees

One way to break the cycles is to replicate the entity types involved in cycles.

- First, we replicate INSTRUCTOR as shown in part (2) of Figure, calling the replica to the right INSTRUCTOR1. The INSTRUCTOR replica on the left represents the relationship between instructors and the sections they teach, whereas the INSTRUCTOR1 replica on the right represents the relationship between instructors and the department each works in.

- We still have the cycle involving COURSE, so we can replicate COURSE in a similar manner, leading to the hierarchy shown in part (3) . The COURSE1 replica to the left represents the relationship between courses and their sections, whereas the COURSE replica to the right represents the relationship between courses and the department that offers each course.

FIGURE 26.13
Converting a graph with cycles into a hierarchical (tree) structure.



XML Querying

XPath

- An XPath expression returns a collection of element nodes that satisfy certain patterns specified in the expression.
- The names in the XPath expression are node names in the XML document tree that are either tag (element) names or attribute names, possibly with additional **qualifier conditions** to further restrict the nodes that satisfy the pattern.
- There are two main **separators** when specifying a path:

  single slash (/) and double slash (//).

  A single slash before a tag specifies that the tag must appear as a direct child of the previous (parent) tag, whereas a double slash specifies that the tag can appear as a descendant of the previous tag *at any level.*
- It is customary to include the file name in any XPath query allowing us to specify any local file name or path name that specifies the path.

  doc(www.company.com/info.XML)/company => COMPANY XML doc

XML Querying

1. Returns the COMPANY root node and all its descendant nodes, which means that it returns the whole XML document.
2. Returns all department nodes (elements) and their descendant subtrees.
3. Returns all employeeName nodes that are direct children of an employee node, such that the employee node has another child element employeeSalary whose value is greater than 70000.
4. This returns the same result as the previous one except that we specified the full path name in this example.
5. This returns all projectWorker nodes and their descendant nodes that are children under a **path /company/project** and that have a child node hours with value greater than 20.0 hours.

FIGURE 26.14
Some examples of XPath expressions on XML documents that follow the XML schema file COMPANY in Figure 26.5.

```
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]
```

XML Querying

XQuery

- XQuery uses XPath expressions, but has additional constructs.
- XQuery permits the specification of more general queries on one or more XML documents.
- The typical form of a query in XQuery is known as a **FLWR expression**, which stands for the four main clauses of XQuery and has the following form:

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

XML Querying

1. This query retrieves the first and last names of employees who earn more than 70000. The variable $x is bound to each employeeName element that is a child of an employee element, but only for employee elements that satisfy the qualifier that their **employeeSalary** is greater that 70000.
2. This is an alternative way of retrieving the same elements retrieved by the first query.
3. This query illustrates how a join operation can be performed by having more than one variable. Here, the $x variable is bound to each projectWorker element that is a child of project number 5, whereas the $y variable is bound to each employee element. The join condition matches ssn values in order to retrieve the employee names.

FIGURE 26.15

Some examples of XQuery queries on XML documents that follow the XML schema file COMPANY in Figure 26.5.

```
1.  FOR $x IN
        doc(www.company.com/info.xml)
        //employee [employeeSalary gt 70000]/employeeName
        RETURN <res> $x/firstName, $x/lastName </res>
2.  FOR $x IN
        doc(www.company.com/info.xml)/company/employee
        WHERE $x/employeeSalary gt 70000
        RETURN <res> $x/employeeName/firstName,
                $x/employeeName/lastName </res>
3.  FOR $x IN
        doc(www.company.com/info.xml)/company
                /project[projectNumber = 5]/projectWorker,
            $y IN
        doc(www.company.com/info.xml)/company/employee
        WHERE $x/hours gt 20.0 AND $y.ssn = $x.ssn
        RETURN <res> $y/employeeName/firstName,
                $y/employeeName/lastName, $x/hours </res>
```