CS3492 DATABASE MANAGEMENT SYSTEMS

UNIT I RELATIONAL DATABASES **10**

Purpose of Database System – Views of data – Data Models – Database System Architecture – Introduction to relational databases – Relational Model – Keys – Relational Algebra – SQL fundamentals – Advanced SQL features – Embedded SQL– Dynamic SQL

UNIT II DATABASE DESIGN **8**

Entity-Relationship model – E-R Diagrams – Enhanced-ER Model – ER-to-Relational Mapping – Functional Dependencies – Non-loss Decomposition – First, Second, Third Normal Forms, Dependency Preservation – Boyce/Codd Normal Form – Multi-valued Dependencies and Fourth Normal Form – Join Dependencies and Fifth Normal Form

UNIT III TRANSACTIONS **9**

Transaction Concepts – ACID Properties – Schedules – Serializability – Concurrency Control – Need for Concurrency – Locking Protocols – Two Phase Locking – Deadlock – Transaction Recovery - Save Points – Isolation Levels – SQL Facilities for Concurrency and Recovery.

UNIT IV IMPLEMENTATION TECHNIQUES **9**

RAID – File Organization – Organization of Records in Files – Indexing and Hashing –Ordered Indices – B+ tree Index Files – B tree Index Files – Static Hashing – Dynamic Hashing – Query Processing Overview – Algorithms for SELECT and JOIN operations – Query optimization using Heuristics and Cost Estimation.

UNIT V ADVANCED TOPICS **9**

Distributed Databases: Architecture, Data Storage, Transaction Processing – Object-based Databases: Object Database Concepts, Object-Relational features, ODMG Object Model, ODL, OQL - XML Databases: XML Hierarchical Model, DTD, XML Schema, XQuery – Information Retrieval: IR Concepts, Retrieval Models, Queries in IR systems.

TOTAL: 45 PERIODS

TEXT BOOKS:

1. Abraham Silberschatz, Henry F. Korth, S. Sudharshan, —Database System Concepts||,Sixth Edition, Tata McGraw Hill, 2011
2. Ramez Elmasri, Shamkant B. Navathe, —Fundamentals of Database Systems||, Sixth Edition, Pearson, 2011.

REFERENCES:

1. C. J. Date, A.Kannan, S. Swamynathan, —An Introduction to Database Systems||, Eighth Edition, Pearson Education, 2006.
2. Raghu Ramakrishnan, —Database Management Systems||, Fourth Edition, McGraw-Hill College Publications, 2015.
3. G.K.Gupta,"Database Management Systems||, Tata McGraw Hill, 2011.

UNIT V ADVANCED TOPICS

Distributed Databases: Architecture, Data Storage, Transaction Processing – Object-based Databases: Object Database Concepts, Object-Relational features, ODMG Object Model, ODL, OQL - XML Databases: XML Hierarchical Model, DTD, XML Schema, XQuery – Information Retrieval: IR Concepts, Retrieval Models, Queries in IR systems

DISTRIBUTED DATABASES

- Distributed databases (DDBs), distributed database management systems (DDBMSs), and how the client-server architecture is used as a platform for database application development. Distributed databases bring the advantages of distributed computing to the database management domain.
- A **distributed computing system** consists of a number of processing elements, not necessarily homogeneous, that are interconnected by a computer network, and that cooperate in performing certain assigned tasks.
- As a general goal, distributed computing systems partition a big, unmanageable problem into smaller pieces and solve it efficiently ina coordinated manner.
- The economic viability of this approach stems from two reasons: more computing power is harnessed to solve a complex task, and each autonomous processing element can be managed independently to develop its own applications.
- DDB technology resulted from a merger of two technologies: database technology, and network and data communication technology. Computer networks allow distributed processing of data.
- Traditional databases, on the other hand, focus on providing centralized, controlled access to data. Distributed databases allow an integration of information and its processing by applications that may themselves be centralized or distributed.

5.5.1 Distributed Database Concepts

We can define a **distributed database (DDB)** as a collection of multiple logically interrelated databases distributed over a computer network, and a **distributed database management system (DDBMS)** as a software system that manages a distributed database while making the distribution transparent to the user.

Distributed databases are different from Internet Web files. Web pages are basically a very large collection of files stored on different nodes in a network—the Internet—with interrelationships among the files represented via hyperlinks.

The common functions of database management, including uniform query processing and transaction processing, *do not* apply to this scenario yet.

The technology is, however, moving in a direction such that distributed World Wide Web (WWW) databases will become a reality in the future. We have discussed some of the issues of

5.5.2 Differences between DDB and Multiprocessor Systems

We need to distinguish distributed databases from multiprocessor systems that use shared storage (primary memory or disk). For a database to be called distributed, the following minimum conditions should be satisfied:

- **Connection of database nodes over a computer network.**

  There are multiple computers, called **sites** or **nodes**. These sites must be connected by an underlying **communication network** to transmit data and commands among sites.

- **Logical interrelation of the connected databases.**

  It is essential that the information in the databases be logically related.

- **Absence of homogeneity constraint among connected nodes.**

It is not necessarythat all nodes be identical in terms of data, hardware, and software.

5.5.3 Transparency

The concept of transparency extends the general idea of hiding implementation details from end users. A highly transparent system offers a lot of flexibility to the end user/application developer since it requires little or no awareness of underlying details on their part.

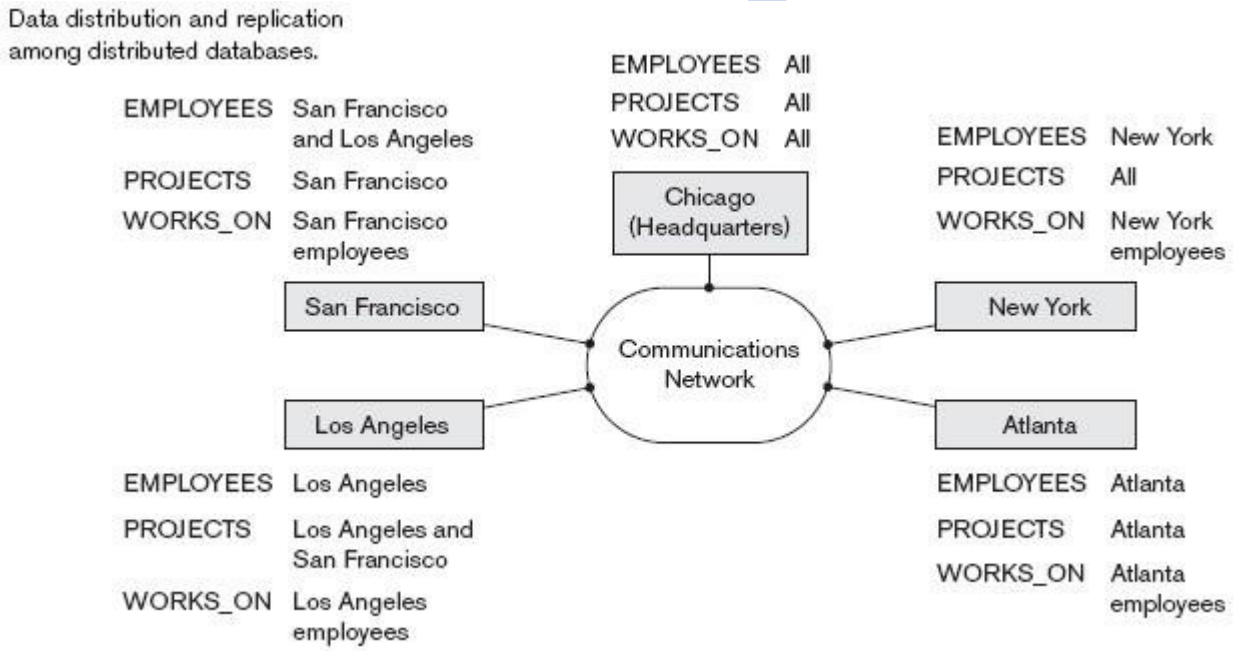
In the case of a traditional centralized database, transparency simply pertains to logical and physical data independence for application developers. However, in a DDB scenario, the data and software are distributed over multiple sites connected by a computer network, so additional types of transparencies are introduced.

- **Data organization transparency (also known as *distribution* or *networktransparency*).**

This refers to freedom for the user from the operational details of the network and the placement of the data in the distributed system. It may be divided into location transparency and naming transparency.

- **Location transparency** refers to the fact that the command used to perform a task is independent of the location of the data and the location of the node where the command was issued.

- **Naming transparency** implies that once a name is associated with an object, the named objects can be accessed unambiguously without additional specification as to where the data is located.
- **Replication transparency.** As we show in Figure 25.1, copies of the same data objects may be stored at multiple sites for better availability, performance, and reliability. Replication transparency makes the user unaware of the existence of these copies.
- **Fragmentation transparency.** Two types of fragmentation are possible.
- **Horizontal fragmentation** distributes a relation (table) into sub relations



5.5.4

a

ed

n,

and execution.

- **Design autonomy** refersto independence of data model usage and transaction management techniques among nodes.
- **Communication autonomy** determines the extent to which each node can decide on sharing of information with other nodes.
- **Execution autonomy** refers to independence of users to act as they please.

5.5.5 *Reliability and Availability*

- Reliability and availability are two of the most common potential advantages cited for distributed databases.
- **Reliability** is broadly defined as the probability that a system is running (not down) at a certain time point, whereas **availability** is the probability that the system is continuously available during a time interval. We can directly relate reliability and availability of the database to the faults, errors, and failures associated with it.
- A failure can be described as a deviation of a system's behavior from that which is specified in order to ensure correct execution of operations.
- **Errors** constitute that subset of system states that causes the failure. **Fault** is the cause of an error.

5.5.6 Advantages of Distributed Databases

Organizations resort to distributed database management for various reasons. Some important advantages are listed

1. **Improved ease and flexibility of application development**.
2. **Increased reliability and availability.**
3. **Improved performance.**
4. **Easier expansion**.

5.5.7 Additional Functions of Distributed Databases

■ **Keeping track of data distribution.** The ability to keep track of the data distribution, fragmentation, and replication by expanding the DDBMS catalog.

■ **Distributed query processing.** The ability to access remote sites and transmit queries and data among the various sites via a communication network.

■ **Distributed transaction management.** The ability to devise execution strategies for queries and transactions that access data from more than one site and to synchronize the access to distributed data and maintain the integrity of the overall database.

■ **Replicated data management.** The ability to decide which copy of a replicated data item to access and to maintain the consistency of copies of a replicated data item.

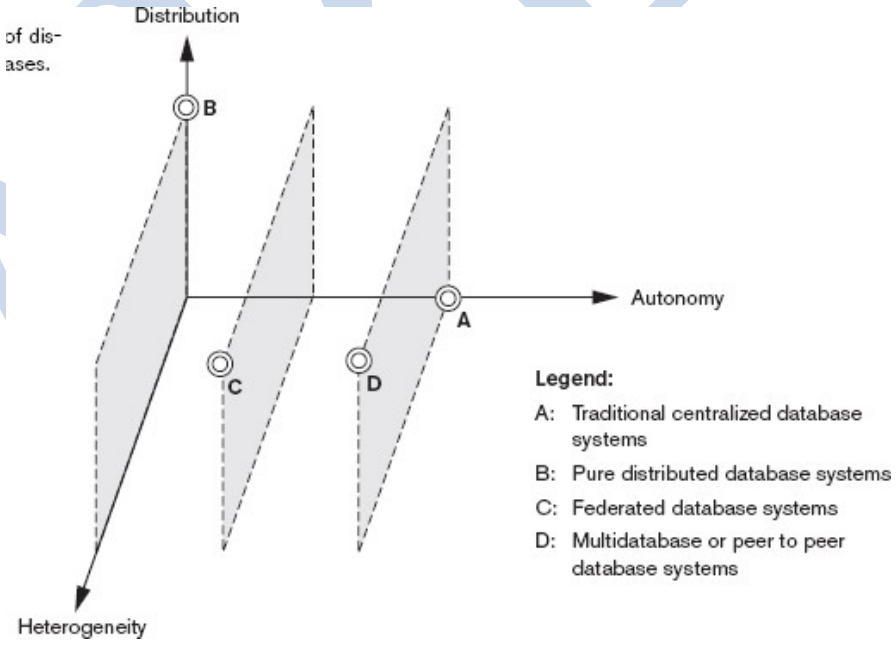
■ **Distributed database recovery.** The ability to recover from individual site crashes and from new types of failures, such as the failure of communication links.

■ **Security.** Distributed transactions must be executed with the proper management of the security of the data and the authorization/access privileges of users.

■ **Distributed directory (catalog) management.** A directory contains information (metadata) about data in the database. The directory may be global for the entire DDB, or local for each site. The placement and distribution of the directory are design and policy issues

5.5.8 Types of Distributed Database Systems

The term *distributed database management system* can describe various systems that differ from one another in many respects. The main thing that all such systems have in common is the fact that data and software are distributed over multiple sites connected by some form of communication network. In this section we discuss a number of types of DDBMSs and the criteria and factors that make some of these systems different.



XML DATABASES

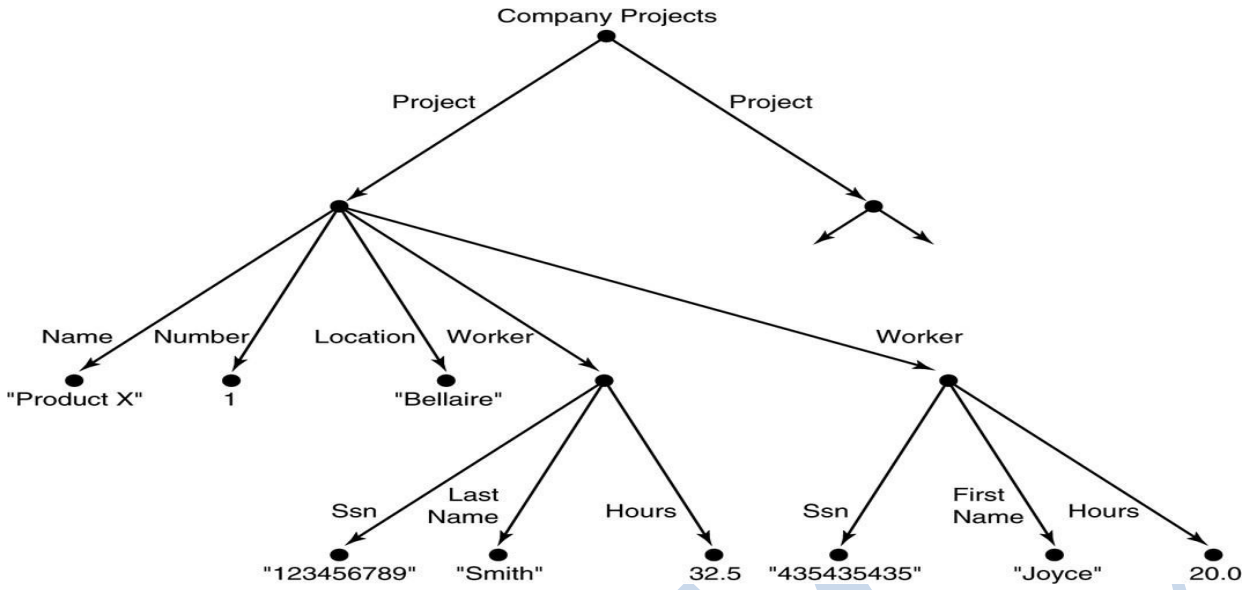
5.12.1 Introduction

- 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).

5.12.2 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.
- 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.

5.12.3 Representing semi structured data as a graph.



5.12.4 XML Hierarchical (Tree) Data Model

```
<?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>
```

- 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.**

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.

5.12.5 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** - 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.

5.12.6 XML Documents, DTD, and XML Schema:

```
<!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)>
] >
```

5.12.7 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.

5.12.8 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 perm.

5.12.9 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>
```

5.12.10 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.

- ***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.

- ***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.

5.12.11 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.

5.12. 12 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.