

# UNIT – I

## SYLLABUS

**Databases and Database Users:** Introduction - An Example - Characteristics of the Database Approach - Actors on the Scene - Workers behind the Scene - Advantages of Using the DBMS Approach - A Brief History of Database Applications - When Not to Use a DBMS

**Database System Concepts and Architecture:** Data Models, Schemas, and Instances - Three-Schema Architecture and Data Independence - Database Languages and Interfaces - The Database System Environment - Centralized and Client/Server Architectures for DBMSs - Classification of Database Management Systems

**Data Modeling Using the Entity-Relationship (ER) Model:** Using High-Level Conceptual Data Models for Database Design - An Example Database Application - Entity Types, Entity Sets, Attributes, and Keys - Relationship Types, Relationship Sets, Roles, and Structural Constraints - Weak Entity Types - Refining the ER Design for the COMPANY Database - ER Diagrams, Naming Conventions, and Design Issues

## 1. Databases and Database Users

### 1.1) Introduction

- **Data** mean real facts that can be recorded and that have implicit meaning.
- **Database** is a collection of related data.
- A database has the following implicit properties:
  - ✓ A database represents some real world aspect, sometimes called the **mini-world** or the **universe of discourse (UoD)**. Changes to the mini-world are reflected in the database.
  - ✓ A database is a logically coherent collection of data with some inherent meaning. A random assortment of data cannot correctly be referred to as a database.
  - ✓ A database is designed, built, and populated with data for a specific purpose.
- A **database management system (DBMS)** is a collection of programs that enables users to create and maintain a database.
- The DBMS facilitates the processes of *defining*, *constructing*, *manipulating*, and *sharing* databases among various **users** and **applications**.
  - **Defining** a database involves specifying the data types, structures, and constraints of the data to be stored in the database. The database definition or descriptive information is also stored in the form of a **database catalog** or **dictionary**; it is called **meta-data**.
  - **Constructing** the database is the process of storing the data on some storage medium that is controlled by the DBMS.
  - **Manipulating** a database includes functions such as querying the database to retrieve specific data, updating the database to reflect changes in the **mini-world**, and generating reports from the data.
  - **Sharing** a database allows multiple users and programs to access the database simultaneously.
- An **application program** accesses the database by sending queries or requests for data to the DBMS.
- A **query** typically causes some data to be retrieved.
- A **transaction** may cause some data to be read and some data to be written into the database.

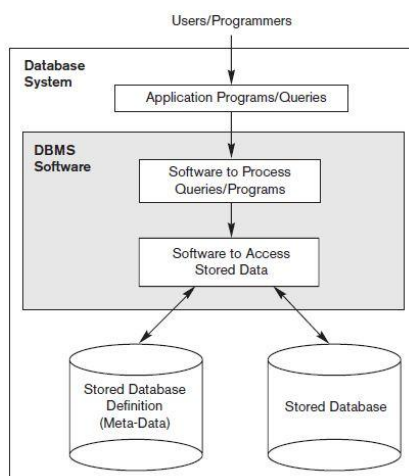


Figure 1.1A simplified database system environment.

# UNIT – I

## 1.2) An Example

UNIVERSITY database for maintaining information concerning **students, courses,**and **grades** in a university environment. Figure 1.2 shows the database structure and a few sample data for such a database.

### STUDENT

Name	Student_number	Class	Department
Smith	17	1	CS
Brown	8	2	CS

### COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

### SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

### GRADE\_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

### PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

**Figure 1.2** A database that stores student and course information.

To *define* this database

- We must specify the structure of the records of each file by specifying the different types of **data elements** to be stored in each record.
- We must also specify a **data type** for each data element within a record.

To *construct* the UNIVERSITY database,

- We store data to represent each student, course, section, grade report, and prerequisite as a record in the appropriate file.
- Notice that records in the various files may be related.

Database *manipulation* involves querying and updating. Examples of queries are as follows:

- Retrieve the transcript (a list of all courses and grades) of ‘Smith’.
- List the names of students who took the section of the ‘Database’ course offered in fall 2008 and their grades in that section.
- List the prerequisites of the ‘Database’ course.

Examples of updates include the following:

- Change the class of ‘Smith’ to sophomore.
- Create a new section for the ‘Database’ course for this semester.
- Enter a grade of ‘A’ for ‘Smith’ in the ‘Database’ section of last semester.

# UNIT – I

## 1.3) Characteristics of the Database Approach

The main characteristics of the database approach versus the file-processing approach are the following:

1. Self-describing nature of a database system
2. Insulation between programs and data
3. Data abstraction
4. Support of multiple views of the data
5. Sharing of data and multiuser transaction processing

### 1. Self-Describing Nature of a Database System

- The database system contains not only the database itself but also a complete **definition or description** of the database structure and constraints.
- This definition is stored in the DBMS **catalog**, which contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data.
- The information stored in the catalog is called **meta-data** and it describes the structure of the primary database.
- Figure 1.3 shows some sample entries in a database catalog.

**RELATIONS**

**COLUMNS**

Relation_name	No_of_columns
STUDENT	4
COURSE	4
SECTION	5
GRADE_REPORT	3
PREREQUISITE	2

Column_name	Data_type	Belongs_to_relation
Name	Character (30)	STUDENT
Student_number	Character (4)	STUDENT
Class	Integer (1)	STUDENT
Major	Major_type	STUDENT
Course_name	Character (10)	COURSE
Course_number	XXXXNNNN	COURSE
.....	.....	.....
.....	.....	.....
.....	.....	.....
Prerequisite_number	XXXXNNNN	PREREQUISITE

**Figure 1.3** An example of a database catalog for the database in Figure 1.2.

### 2. Insulation between Programs and Data

- **Program-data independence** means the structure of data files are stored in the DBMS catalog separately from the access programs.
- An **operation** (also called a *function* or *method*) is specified in two parts.
  - ✓ The **interface** of an operation includes the operation name and the data types of its arguments.
  - ✓ The **implementation** of the operation is specified separately and can be changed without affecting the interface.
- **Program-operation independence** means user application programs can operate on the data through their names and arguments, regardless of how the operations are implemented.

### 3. Data abstraction

- **Data abstraction** refers to the details of data organization and storage, and the highlighting of the essential features for an improved understanding of data.
- **Data abstraction** is allows program-data independence and program-operation independence.
- **Conceptual representation** of data that does not include many of the details of how the data is stored or how the operations are implemented.
- A **data model** is a collection of concepts that can be used to describe the structure of a database.

### 4. Support of Multiple Views of the Data

- A **view** may be a subset of the database or it may contain **virtual data** that is derived from the database files but is not explicitly stored.
- A multiuser DBMS whose users have a variety of distinct applications must provide facilities for defining multiple views.

### 5. Sharing of Data and Multiuser Transaction Processing

- A multiuser DBMS is must allow multiple users to access the database at the same time.

## UNIT – I

- The DBMS must include **concurrency control** software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct.
- A transaction is an *executing program* or *process* that includes one or more database accesses, such as reading or updating of database records.
- The DBMS must enforce several transaction properties:
  - The **isolation** property ensures that each transaction appears to execute in isolation from other transactions, even though hundreds of transactions may be executing concurrently.
  - The **atomicity** property ensures that either all the database operations in a transaction are executed or none are.

### 1.4) Actors on the Scene

The people who are involve the day-to-day use of a large database called them *actors on the scene*.

- **Database Administrators**
- **Database Designers**
- **End Users**
- **System Analysts and Application Programmers (Software Engineers)**

#### 1. Database Administrators:

- The **database administrator (DBA)** is responsible for authorizing access to the database, coordinating and monitoring its use, and acquiring software and hardware resources as needed.
- The DBA is accountable for problems such as security breaches and poor system response time.

#### 2. Database Designers:

- **Database designers** are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data.
- These tasks are undertaken before the database is actually implemented and populated with data.
- It is the responsibility of database designers to communicate with all prospective database users in order to understand their requirements and to create a design that meets these requirements.

#### 3. End Users:

- **End users** are the people whose jobs require access to the database for querying, updating, and generating reports.
- There are several categories of end users:
  - ✓ **Casual end users** occasionally access the database, but they may need different information each time.
  - ✓ **Naive or parametric end users** their main job function revolves around constantly querying and updating the database, using standard types of queries and updates(**canned transactions**) have been carefully programmed and tested.
  - ✓ **Sophisticated end users** include engineers, scientists, business analysts, and others who thoroughly familiarize themselves with the facilities of the DBMS in order to implement their own applications to meet their complex requirements.
  - ✓ **Standalone users** maintain personal databases by using ready-made program packages that provide easy-to-use menu-based or graphics-based interfaces.

#### 4. System Analysts and Application Programmers(Software Engineers):

- **System analysts** determine the requirements of end users, especially naive and parametric end users, and develop specifications for standard canned transactions that meet these requirements.
- **Application programmers** implement these specifications as programs; then they test, debug, document, and maintain these canned transactions.
- Such analysts and programmers commonly referred to as **software developers** or **software engineers**.

## UNIT – I

### 1.5) Workers behind the Scene

These persons are typically not interested in the database content itself call them the *workers behind the scene*.

- **DBMS system designers and implementers** design and implement the DBMS modules and interfaces as a software package. A DBMS is a very complex software system that consists of many components, or **modules**, including modules for implementing the catalog, query language processing, interface processing, accessing and buffering data, controlling concurrency, and handling data recovery and security.
- **Tool developers** design and implement **tools**—the software packages that facilitate database modeling and design, database system design, and improved performance.
- **Operators and maintenance personnel** (system administration personnel) are responsible for the actual running and maintenance of the hardware and software environment for the database system.

### 1.6) Advantages of Using the DBMS Approach

#### 1. Controlling Redundancy:

- **Redundancy** means **storing the same data multiple times in their own files**.
- This **redundancy** leads to several problems.
  - ✓ *Storage space is wasted* when the same data is stored repeatedly.
  - ✓ *Duplication of effort* to perform a single logical update multiple times.
  - ✓ *Inconsistent* because of files that represent the same data.

#### 2. Restricting Unauthorized Access:

- When multiple users share a large database, it is likely that most users will not be authorized to access all information in the database.
- A DBMS should provide a **security and authorization subsystem**, which the DBA uses to create accounts and to specify account restrictions.
- Then, the DBMS should enforce these restrictions automatically.

#### 3. Providing Persistent Storage for Program Objects:

- Databases can be used to provide **persistent storage** for program objects and data structures.
- Traditional database systems suffered from the **impedance mismatch problem**, since the data structures database system were **incompatible with the programming language's data structures**.
- Object-oriented database systems were offer data structure **compatibility** with one or more object-oriented programming languages.

#### 4. Providing Storage Structures and Search Techniques for Efficient Query Processing:

- Database systems must provide capabilities for *efficiently executing queries and updates*.
- Because the database is typically stored on disk, the DBMS must **provide specialized data structures and search techniques** to speed up disk search for the desired records.
- The **query processing and optimization** module of the DBMS is responsible for choosing an efficient query execution plan for each query based on the existing storage structures.

#### 5. Providing Backup and Recovery:

- A DBMS must provide facilities for recovering from hardware or software failures.
- The **backup and recovery subsystem** of the DBMS is responsible for recovery.
- For example, if the computer system fails in the middle of a complex update transaction, the recovery subsystem is responsible for making sure that the database is restored to the state it was in before the transaction started executing.

## UNIT – I

### 6. Providing Multiple User Interfaces:

- DBMS should provide a variety of **user interfaces**. These include query languages for casual users, programming language interfaces for application programmers, forms and command codes for parametric users, and menu-driven interfaces and natural language interfaces for standalone users.

### 7. Representing Complex Relationships among Data:

- A DBMS must have the capability to represent a variety of complex relationships among the data, to define new relationships as they arise, and to retrieve and update related data easily and efficiently.

### 8. Enforcing Integrity Constraints:

- Most database applications have certain **integrity constraints** that must hold for the data. A DBMS should provide capabilities for defining and enforcing these constraints.
- The simplest type of integrity constraint involves specifying a data type for each data item.
- There may have different types of constraints like **referential integrity**, **key** or **uniqueness** constraints.

## 1.7) When Not to Use a DBMS

The overhead costs of using a DBMS are due to the following:

- High initial investment in hardware, software, and training.
- The generality that a DBMS provides for defining and processing data.
- Overhead for providing security, concurrency control, recovery, and integrity functions.

It may be more desirable to use regular files under the following circumstances:

- Simple, well-defined database applications that are not expected to change at all.
- Stringent, real-time requirements for some application programs that may not be met because of DBMS overhead
- Embedded systems with limited storage capacity, where a general-purpose DBMS would not fit
- No multiple-user access to data
- Certain industries and applications have elected not to use general-purpose DBMSs.
- For example, many computer-aided design (CAD) tools used by mechanical and civil engineers have proprietary file and data management software that is geared for the internal manipulations of drawings and 3D objects.

## UNIT – I

# 2. Database System Concepts and Architecture

## 2.1) Data Models, Schemas, and Instances

### Data Model:

- A **data model** is a collection of concepts that can be used to describe the structure of a database.
- **Structure of a database** means the data types, relationships, and constraints that apply to the data.

### Categories of Data Models

#### 1. High-level or conceptual data models:

- High-level data model provide concepts that **the way many users perceive data**.
- Conceptual data models use concepts such as **entities, attributes, and relationships**.
  - An **entity** represents a **real-world object**, such as a student or a course from the mini-world that is described in the database.
  - An **attribute** represents some property that further describes an entity, such as the student name or class.
  - A **relationship** among two or more entities represents an association among the entities.

#### 2. Low-level or physical data models:

- Low-level data model provide concepts that describe the details of **how data is stored** on the computer storage media, typically magnetic disks.
- It represents information as record formats, record orderings, and access paths.
- Low-level data models concepts are generally meant for computer specialists, not for end users.
- An **access path** is a structure that makes the search for particular database records efficient.

#### 3. Representational (or implementation) data models:

- Representational data model provide concepts that may be easily understood by end users but that are not too far removed from the way data is organized in computer storage.
- Representational data models hide many details of data storage on disk but can be implemented on a computer system directly.
- Representational or implementation data models are the models used most frequently in traditional commercial DBMSs like **relational data model**, the **network** and **hierarchical models**.

### Schemas, Instances, and Database State:

- The description of a database is called the **database schema**, which is specified during database design and is not expected to change frequently.
- A displayed schema is called a **schema diagram**.

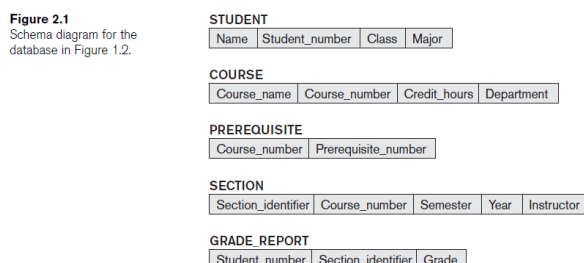


Figure 2.1 shows a schema diagram for the database shown in Figure 1.2

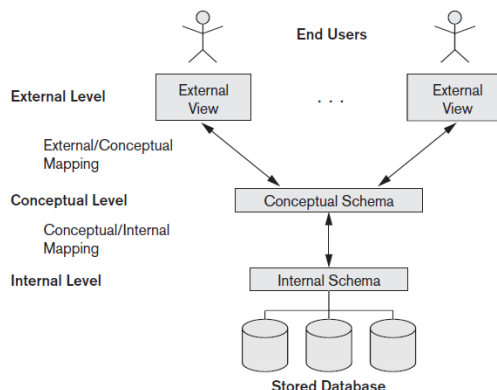
- Each object in the schema (Ex: STUDENT or COURSE) called **schema construct**.
- A schema diagram displays only **some aspects** of a schema, such as the names of record types and data items, and some types of constraints. Other aspects are not specified in the schema diagram.
- The data in the database at a particular moment in time is called a **database state** or **snapshot** or **instances** or **current set of occurrences**
- When we **define** a new database, we specify its database schema only to the DBMS and the corresponding database state is the **empty state** with no data.
- We get the **initial state** of the database when the database is first **populated** or **loaded** with the initial data.

# UNIT – I

## 2.2) Three-Schema Architecture and Data Independence

### The Three-Schema Architecture

- The goal of the three-schema architecture (Figure 2.2) is to **separate the user applications from the physical database**.
- In this architecture, schemas can be defined at the following three levels:



**Figure 2.2** The three-schema architecture

1. The **internal level** has an **internal schema**, which is uses physical data model and **describes the complete details of data storage and access paths for the database**.
  2. The **conceptual level** has a **conceptual schema**, which describes the structure of the whole database for a community of users.
    - The conceptual schema **hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints**.
    - This *implementation conceptual schema* is often based on a *conceptual schema design* in a high-level data model.
  3. The **external or view level** includes a number of **external schemas** or **user views**.
    - Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group.
    - Each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high-level data model.
- The DBMS must transform a request specified on an external schema into a request against the conceptual schema, and then into a request on the internal schema for processing over the stored database.
  - If the request is database retrieval, the data extracted from the stored database must be reformatted to match the user's external view.
  - The processes of transforming requests and results between levels are called **mappings**.

### Data Independence

- The capacity to change the schema at one level of a database system without having to change the schema at the next higher level called **data independence**.
- We can define two types of data independence:
  1. **Logical data independence** is the capacity to change the conceptual schema without having to change external schemas or application programs.
    - We may change the conceptual schema to **expand the database** (by adding a record type or data item), to change constraints, or to **reduce the database** (by removing a record type or data item).
  2. **Physical data independence** is the capacity to change the internal schema without having to change the conceptual schema.
    - Changes to the internal schema may be needed because some physical files were reorganized to improve the performance of retrieval or update.
    - If the same data as before remains in the database, we should not have to change the conceptual schema.



## UNIT – I

### 2.3) Database Languages and Interfaces

The DBMS must provide appropriate languages and interfaces for each category of users.

#### DBMS Languages

- The first step is to specify conceptual and internal schemas for the database and any mappings between the two.
- **Data definition language (DDL)** is used by the DBA and by database designers to define both **conceptual** and **internal schemas**.
- DDL compiler is used to process DDL statements to identify descriptions of the schema constructs and to store the schema description in the DBMS catalog.
- **Storage definition language (SDL)** is used to specify the **internal schema**. The mappings between the two schemas may be specified in either DDL or SDL languages.
- **View definition language (VDL)** is used to specify **user views and their mappings to the conceptual schema**, but in most DBMSs *the DDL is used to define both conceptual and external schemas*.
- **Data manipulation language (DML)** is used to manipulate the database. Manipulation includes **retrieval, insertion, deletion, and modification** of the data.
- There are two main types of DMLs.
  1. A **high-level** or **nonprocedural** DML can be used on its own to specify complex database operations concisely.
    - ✓ DBMSs allow high-level DML statements either to be entered interactively from a **display monitor** or **terminal** or **to be embedded in a general-purpose programming language**.
    - ✓ DML statements must be identified within the program so that they can be extracted by a precompiler and processed by the DBMS.
    - ✓ High level DMLs, such as **SQL**, can specify and retrieve many records in a single DML statement; therefore, they are called **set-at-a-time** or **set-oriented** DMLs.
  2. A **low-level** or **procedural** DML *must* be embedded in a general-purpose programming language.
    - ✓ This type of DML typically retrieves individual records or objects from the database and processes each separately.
    - ✓ Therefore, it needs to use programming language constructs, such as looping, to retrieve and process each record from a set of records.
    - ✓ Low-level DMLs are also called **record-at-a-time**
- ❖ Whether high level or low level, are embedded in a general-purpose programming language and that language is called the **host language** and the DML is called the **data sublanguage**.
- ❖ A high-level DML used in a standalone interactive manner is called a **query language**.

#### DBMS Interfaces

- User-friendly interfaces provided by a DBMS may include the following:
  1. **Menu-Based Interfaces for Web Clients or Browsing.** These interfaces present the user with lists of options (called **menus**) that lead the user through the formulation of a request. Menus do away with the need to memorize the specific commands and syntax of a query language; rather, the query is composed step-by-step by picking options from a menu that is displayed by the system. Pull-down menus are a very popular technique in **Web-based user interfaces**
  2. **Forms-Based Interfaces.** A forms-based interface displays a form to each user. Users can fill out all of the **form** entries to insert new data, or they can fill out only certain entries, in which case the DBMS will retrieve matching data for the remaining entries. Forms are usually designed and programmed for naive users as interfaces to canned transactions. Many DBMSs have **forms specification languages**, which are special languages that help programmers, specify such forms.

## UNIT – I

- 3. Graphical User Interfaces.** A GUI typically displays a schema to the user in diagrammatic form. The user then can specify a query by manipulating the diagram. In many cases, GUIs utilize both menus and forms. Most GUIs use a **pointing device**, such as a mouse, to select certain parts of the displayed schema diagram.
- 4. Natural Language Interfaces.** These interfaces accept requests written in English or some other language and attempt to *understand* them. A natural language interface usually has its own *schema*, which is similar to the database conceptual schema, as well as a dictionary of important words. The natural language interface refers to the words in its schema, as well as to the set of standard words in its dictionary, to interpret the request. If the interpretation is successful, the interface generates a high-level query corresponding to the natural language request and submits it to the DBMS for processing; otherwise, a dialogue is started with the user to clarify the request.
- 5. Speech Input and Output.** Limited use of speech as an input query and speech as an answer to a question or result of a request is becoming commonplace. The speech input is detected using a library of predefined words and used to set up the parameters that are supplied to the queries. For output, a similar conversion from text or numbers into speech takes place. Example inquiries for telephone directory, flight arrival/departure etc.
- 6. Interfaces for Parametric Users.** Parametric users, such as bank tellers, often have a small set of operations that they must perform repeatedly. For example, a teller is able to use single function keys to invoke routine and repetitive transactions such as account deposits or withdrawals, or balance inquiries.
- 7. Interfaces for the DBA.** Most database systems contain privileged commands that can be used only by the DBA staff. These include commands for creating accounts, setting system parameters, granting account authorization, changing a schema, and reorganizing the storage structures of a database.

# UNIT - I

## 2.4) The Database System Environment

### DBMS Component Modules

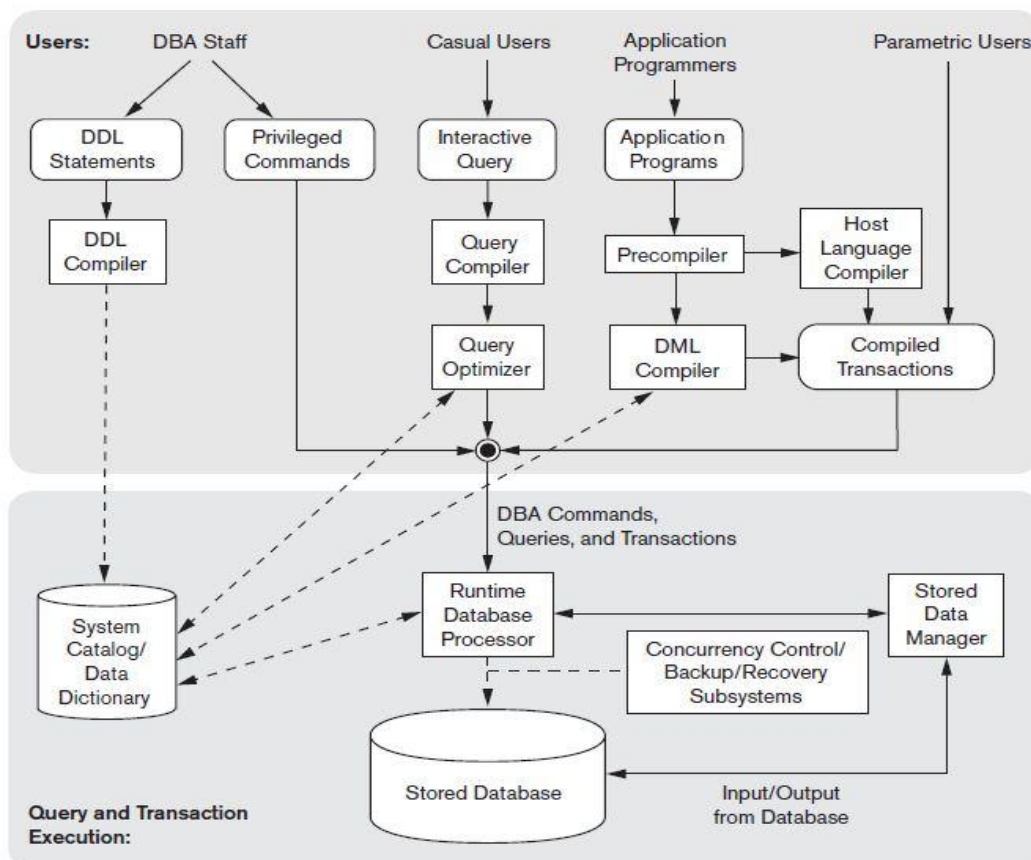


Figure 2.3: Component modules of a DBMS and their interactions.

- Figure 2.3 shows the typical DBMS components. The figure is divided into two parts.
  - The top part refers to the various users of the database environment and their interfaces.
  - The lower part shows the internals of the DBMS responsible for storage of data and processing of transactions.
- Let us consider the top part of Figure 2.3. It shows interfaces for the **DBA staff, casual, application, programmers, and parametric users**.
  - ❖ The **DBA staff** works on defining the database and makes changes in its definition using the DDL and other privileged commands. The DDL compiler processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta-data) in the DBMS **catalog**.
  - ❖ **Casual users and persons** with occasional need for information from the database interact using some form of interface, which we call the **interactive query** interface. The casual users generate the interactive query to parse and validated for correctness of the query syntax by a **query compiler** that compiles them into an internal form (query optimization). The **query optimizer** consults the system catalog for statistical and other physical information about the stored data and generates executable code that performs the necessary operations for the query and makes calls on the runtime processor.
  - ❖ **Application programmers** write programs in host languages such as Java, C, or C++ that are submitted to a precompiler. The **precompiler** extracts DML commands from an application program written in a host programming language. These commands are sent to the DML compiler for compilation into object code for database access. The rest of the program is sent to the host language compiler. The object codes for the DML commands and the rest of the program are linked, forming a canned transaction whose executable code includes calls to the runtime database processor. Canned transactions are executed repeatedly by parametric users, who simply supply the parameters to the transactions. Each execution is considered to be a separate transaction.

## UNIT – I

- In the lower part of Figure 2.3, the **runtime database processor** executes 1) The privileged commands 2) The executable query plans 3) The canned transactions with runtime parameters.
  - ❖ It works with the **system catalog** and may update it with statistics.
  - ❖ It also works with the **stored data manager**, which in turn uses basic operating system services for carrying out low-level input/output (read/write) operations between the disk and main memory.
  - ❖ The runtime database processor handles other aspects of data transfer, such as management of buffers in the main memory.
  - ❖ Some DBMSs have their own buffer management module while others depend on the OS for buffer management.
  - ❖ We have shown **concurrency control** and **backup and recovery systems** separately as a module in this figure. They are integrated into the working of the runtime database processor for purposes of transaction management.
  - ❖ The DBMS interacts with the operating system when disk accesses—to the database or to the catalog—are needed.
  - ❖ If the computer system is shared by many users, the OS will schedule DBMS disk access requests and DBMS processing along with other processes.
  - ❖ If the computer system is mainly dedicated to running the database server, the DBMS will control main memory buffering of disk pages.

### Database System Utilities

DBMSs have **database utilities** that help the DBA manage the database system. Common utilities have the following types of functions:

- ❖ **Loading.** A loading utility is used to load existing data files—such as text files or sequential files—into the database. Usually, the current (source) format of the data file and the desired (target) database file structure are specified to the utility, which then automatically reformats the data and stores it in the database. With the proliferation of DBMSs, transferring data from one DBMS to another is becoming common in many organizations. Some vendors are offering products that generate the appropriate loading programs, given the existing source and target database storage descriptions (internal schemas). Such tools are also called **conversion tools**.
- ❖ **Backup.** A backup utility creates a backup copy of the database, usually by dumping the entire database onto tape or other mass storage medium. The backup copy can be used to restore the database in case of catastrophic disk failure. Incremental backups are also often used, where only changes since the previous backup are recorded. Incremental backup is more complex, but saves storage space.
- ❖ **Database storage reorganization.** This utility can be used to reorganize a set of database files into different file organizations, and create new access paths to improve performance.
- ❖ **Performance monitoring.** Such a utility monitors database usage and provides statistics to the DBA. The DBA uses the statistics in making decisions such as whether or not to reorganize files or whether to add or drop indexes to improve performance.

### Tools, Application Environments, and Communications Facilities

- ❖ **Data dictionary (or data repository) system** tool stores catalog information and also stores design decisions, usage standards, application program descriptions, and user information. Such a system is also called an **information repository**.
- ❖ **Application development environments** [PowerBuilder (Sybase) or JBuilder (Borland)] provide an environment for developing database applications and include facilities that help in many facets of database systems, including database design, GUI development, querying and updating, and application program development.
- ❖ **Communications software**, whose function is to allow users at locations remote from the database system site to access the database through computer terminals, workstations, or personal computers. These are connected to the database site through data communications hardware such as Internet routers, phone lines, long-haul networks, local networks, or satellite communication devices.

# UNIT – I

## 2.5) Centralized and Client/Server Architectures for DBMSs

### 2.5.1 Centralized DBMSs Architecture

- Earlier architectures used mainframe computers to provide the processing for all system functions, including user application programs and user interface programs, as well as all the DBMS functionality.
- Because of most users **accessed such systems via computer terminals**.
- Terminals **did not have processing power and only provided display capabilities**.
- Therefore, all processing was performed remotely on the computer system, and only display information and controls were sent from the computer to the display terminals, which were connected to the central computer via various types of communications networks.
- The DBMS itself was still a **centralized DBMS** in which all the DBMS functionality, application program execution, and user interface processing were carried out on one machine.

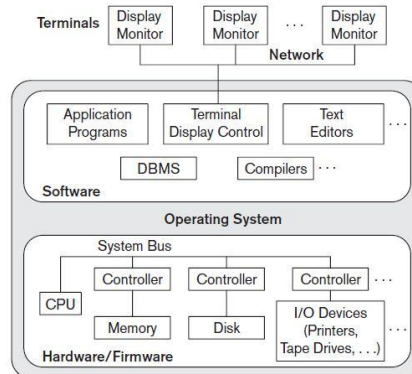


Figure 2.4 physical centralized architecture.

### 2.5.2 Basic Client/Server Architectures

- The concept of client/server architecture consists of many PCs and workstations as well as a smaller number of mainframe machines, connected via LANs and other types of computer networks.
- A **client** in this framework is typically a user machine that provides user interface capabilities and local processing.
- When a client requires access to additional functionality—such as database access—that does not exist at that machine, it connects to a server that provides the needed functionality.
- A **server** is a system containing both hardware and software that can provide services to the client machines, such as file access, printing, archiving, or database access.
- It is more common that client and server software usually run on separate machines.
  - There are two types of client/server frameworks: **two-tier** and **three-tier**.

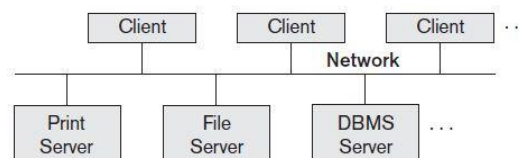


Figure 2.5 illustrates client/server architecture at the logical level;

### 2.5.3 Two-Tier Client/Server Architectures for DBMSs

- In relational database management systems (RDBMSs), many of which started as centralized systems, the system components that were first moved to the client side were the user interface and application programs. Because SQL provided a standard language for RDBMSs, this created a logical dividing point between client and server. Hence, the query and transaction functionality related to SQL processing remained on the server side. In such an architecture, the server is often called a **query server** or **transaction server** because it provides these two functionalities. In an RDBMS, the server is also often called an **SQL server**.

## UNIT - I

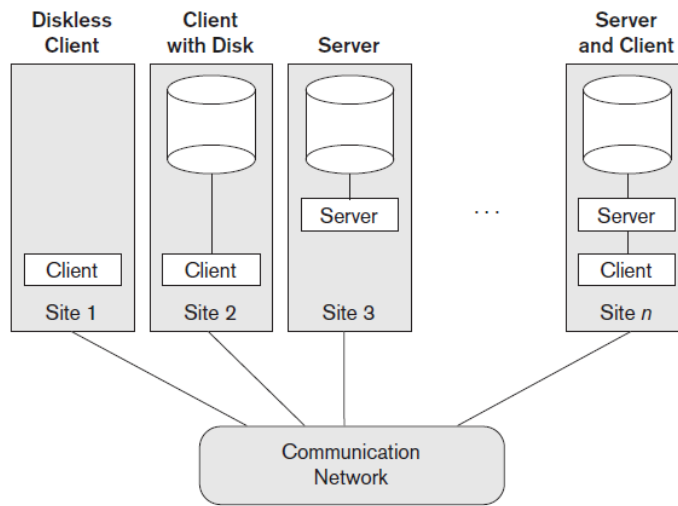


Fig: Physical two-tier client/server architecture.

The user interface programs and application programs can run on the client side. When DBMS access is required, the program establishes a connection to the DBMS (which is on the server side); once the connection is created, the client program can communicate with the DBMS. A standard called **Open Database Connectivity (ODBC)** provides an **application programming interface (API)**, which allows client-side programs to call the DBMS, as long as both client and server machines have the necessary software installed. Most DBMS vendors provide ODBC drivers for their systems. A client program can actually connect to several RDBMSs and send query and transaction requests using the ODBC API, which are then processed at the server sites. Any query results are sent back to the client program, which can process and display the results as needed. The architectures described here are called **two-tier architectures** because the software components are distributed over two systems: client and server. The advantages of this architecture are its simplicity and seamless compatibility with existing systems.

### 2.5.4 Three-Tier and $n$ -Tier Architectures for Web Applications

Many Web applications use an architecture called the **three-tier architecture**, which adds an intermediate layer between the client and the database server, as illustrated in Figure.

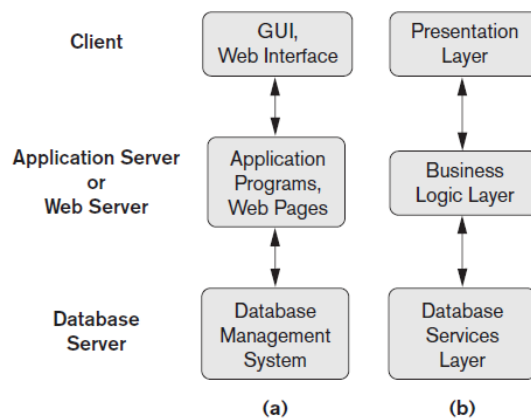


Fig: Logical three-tier client/server architecture, with a couple of commonly used nomenclatures.

This intermediate layer or **middle tier** is called the **application server** or the **Web server**, depending on the application. This server plays an intermediary role by running application programs and storing business rules (procedures or constraints) that are used to access data from the database server. It can also improve database security by checking a client's credentials before forwarding a request to the database server. Clients contain user interfaces and Web browsers. The intermediate server accepts requests from the client, processes the request and sends database queries and commands to the database server, and then acts as a conduit for passing (partially) processed data from the database server to the clients, where it may be processed further and filtered to be presented to the users. Thus, the *user interface*, *application rules*, and *data access* act as the three tiers.

# UNIT – I

## 2.6) Classification of Database Management Systems

1. Based on data model
2. Based on simultaneous access
3. Based on architecture
4. Based on usage

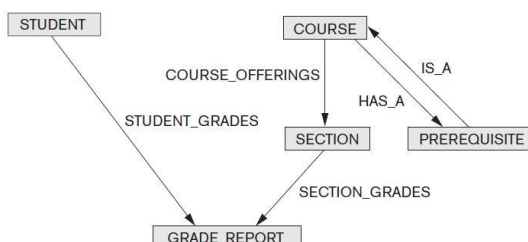
### 1. Based on data model

#### Hierarchical data models

- The **hierarchical model** represents data as hierarchical tree structures.
- Each hierarchy represents a number of related records.
- A popular hierarchical DML is DL/1 of the IMS system.
- DML is procedure and record oriented (no query processor).
- Currency issues during navigation are also handled with additional features in the language
- Examples of hierarchical DBMSs include IMS (IBM), SAS, and TDMS

#### Network data models

- The **network model** represents data as record types and also represents a limited type of 1: N relationship, called a **set type**.
- A 1: N, or one-to-many, relationship relates one instance of a record to many record instances using some pointer linking mechanism in these models.



**Figure 2.8** The schema of Figure 2.1 in network model notation.

- The network model, also known as the CODASYL DBTG model, has an associated record-at-a-time language that must be embedded in a host programming language.
- The language also handles many additional considerations, such as the currency of record types and set types, which are defined by the current position of the navigation process within the database.
- It is prominently used by IDMS, IMAGE, and SUPRA DBMSs today.

#### Relational data model

- The basic **relational data model** represents a database as a collection of tables, where each table can be stored as a separate file.
- Most relational databases use the high-level query language called SQL and support a limited form of user views.

#### Object data model

- The **object data model** defines a database in terms of objects, their properties, and their operations.
- Objects with the same structure and behavior belong to a **class**, and classes are organized into **hierarchies** (or **acyclic graphs**). The operations of each class are specified in terms of predefined procedures called **methods**.

#### Object-relational model

- Relational DBMSs have been extending their models to incorporate object database concepts and other capabilities; these systems are referred to as **object-relational** or **extended relational systems**.
- This added the feature like collections, user defined types, inheritance etc.

#### XML model

- The **XML model** has emerged as a standard for exchanging data over the Web, and has been used as a basis for implementing several prototype native XML systems.
- XML uses hierarchical tree structures.

## UNIT – I

- It combines database concepts with concepts from document representation models.
- Data is represented as elements; with the use of tags, data can be nested to create complex hierarchical structures.
- This model conceptually resembles the object model but uses different terminology.
- XML capabilities have been added to many commercial DBMS products.

### 2. Based on simultaneous access

- The criterion about the **number of users** supported by the system
- **Single-user systems** support only one user at a time and are mostly used with PCs.
- **Multuser systems** support concurrent multiple users.

### 3. Based on architecture

- The third criterion is the **number of sites** over which the database is distributed.
- A DBMS is **centralized** if the data is stored at a single computer site.
- A centralized DBMS can support multiple users, but the DBMS and the database reside totally at a single computer site.
- A **distributed** DBMS (DDBMS) can have the actual database and DBMS software distributed over many sites, connected by a computer network.
- **Homogeneous** DDBMSs use the same DBMS software at all the sites, whereas **heterogeneous** DBMSs can use different DBMS software at each site.
- It is also possible to develop **middleware software** to access several autonomous preexisting databases stored under heterogeneous DBMSs (**federated DBMS** (or) **multi-database system**).
- Many DDBMSs use client-server architecture.

### 4. Based on usage

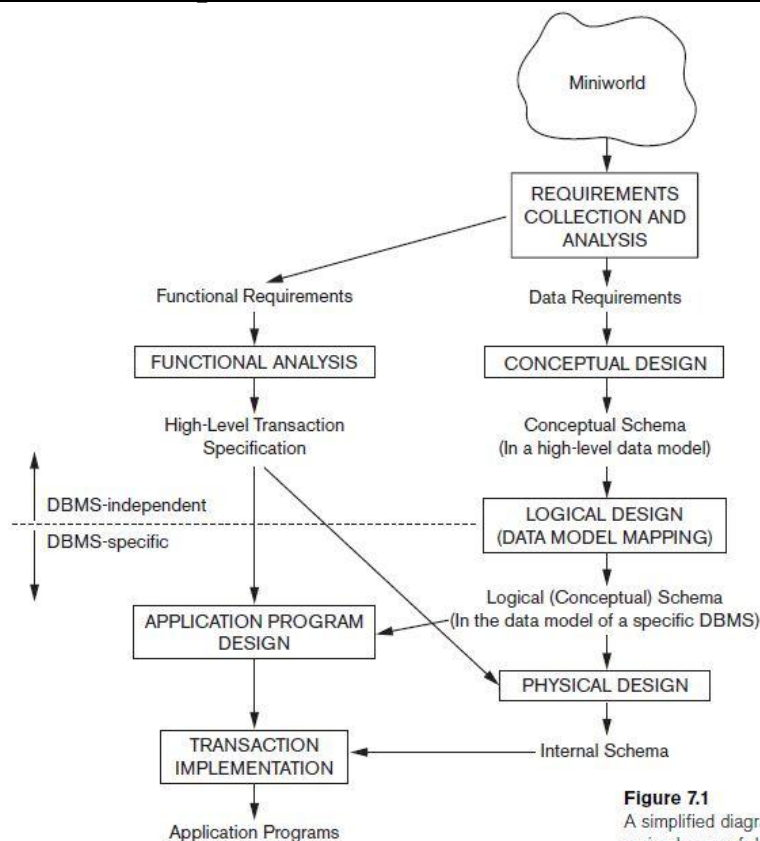
- It is difficult to propose a classification of DBMSs based on cost.
- Today we have open source (free) DBMS products like MySQL and PostgreSQL that are supported by third-party vendors with additional services.



## UNIT – I

### 3. Data Modeling Using the Entity-Relationship (ER) Model

#### 3.1) Using High-Level Conceptual Data Models for Database Design



**Figure 7.1**  
A simplified diagram to illustrate the main phases of database design.

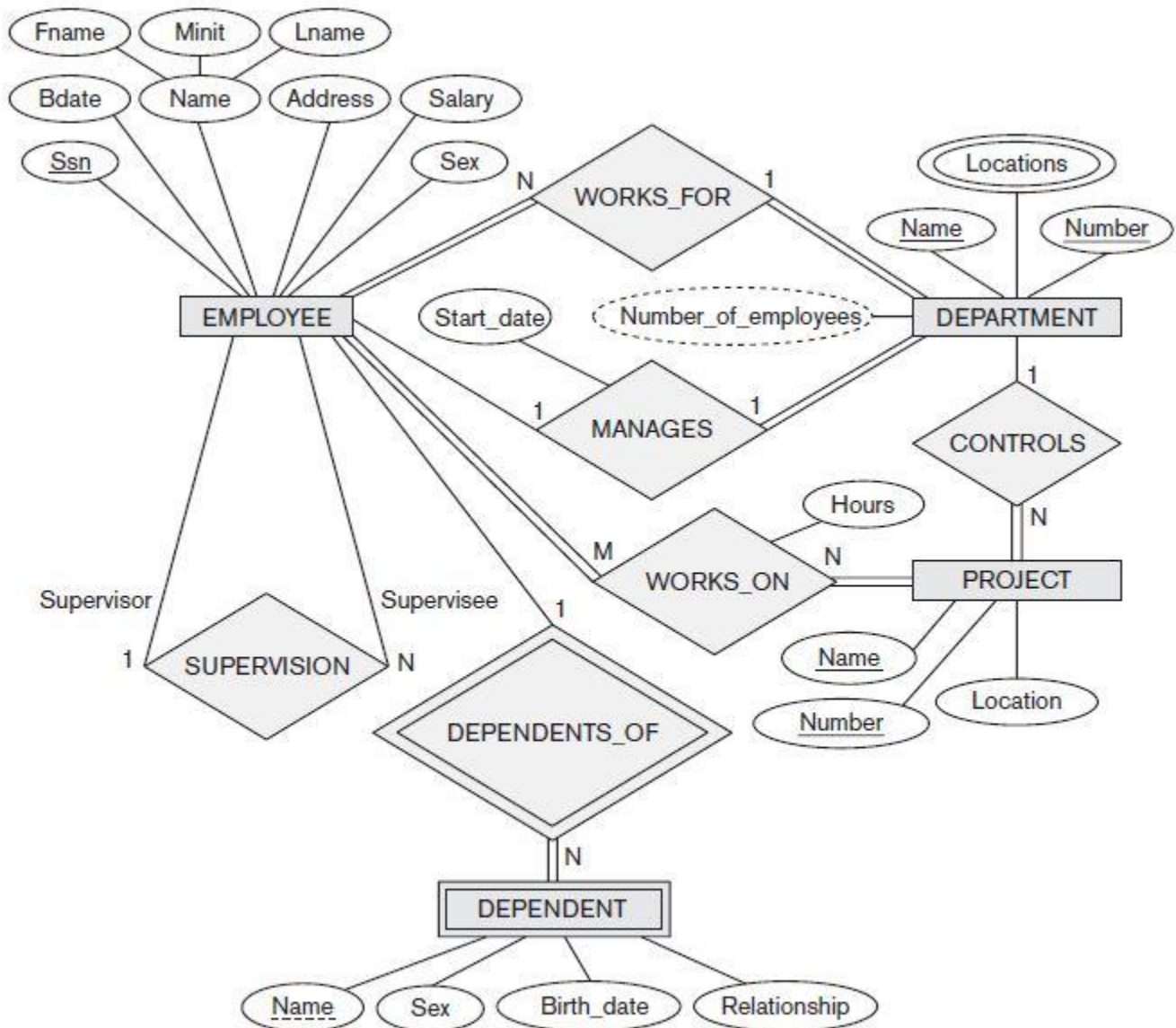
Figure 7.1 shows a simplified overview of the database design process.

- ❖ The first step shown is **requirements collection and analysis**. During this step, the database designers interview prospective database users to understand and document their **data requirements**. These requirements should be specified in as detailed and complete a form as possible.
- ❖ In parallel with specifying the data requirements, it is useful to specify the known **functional requirements** of the application. These consist of the user defined **operations** (or **transactions**) that will be applied to the database, including both retrievals and updates.
- ❖ The next step is to create a **conceptual schema** for the database, using a high-level conceptual data model is called **conceptual design**. The conceptual schema is a concise description of the data requirements of the users and includes detailed descriptions of the entity types, relationships, and constraints. Because these concepts do not include implementation details, they are usually easier to understand and can be used to communicate with nontechnical users. The high-level conceptual schema can also be used as a reference to ensure that all users' data requirements are met and that the requirements do not conflict. This approach enables database designers to concentrate on specifying the properties of the data, without being concerned with storage and implementation details.
- ❖ During or after the conceptual schema design, the basic data model operations can be used to specify the high-level user queries and operations identified during functional analysis.
- ❖ The next step **logical design** or **data model mapping** is the conceptual schema is transformed from the high-level data model into the implementation data model is called **logical design**. **Here** the actual implementation of the database is done using a commercial DBMS such as relational or the object-relational database model. Data model mapping is often automated or semi-automated within the database design tools.
- ❖ The last step is the **physical design** phase, during which the internal storage structures, file organizations, indexes, access paths, and physical design parameters for the database files are specified. In parallel with these activities, application programs are designed and implemented as database transactions corresponding to the high level transaction specifications.

# UNIT – I

## 3.2) A Sample Database Application

- In this section we describe a sample database application, called COMPANY, which serves to illustrate the basic ER model concepts and their use in schema design.
  - COMPANY database keeps track of a company's employees, departments, and projects.
  - ❖ The company is organized into departments. Each department has a unique name, a unique number, and a particular employee who manages the department. We keep track of the start date when that employee began managing the department. A department may have several locations.
  - ❖ A department controls a number of projects, each of which has a unique name, a unique number, and a single location.
  - ❖ We store each employee's name, Social Security number, address, salary, sex (gender), and birth date. An employee is assigned to one department, but may work on several projects, which are not necessarily controlled by the same department. We keep track of the current number of hours per week that an employee works on each project. We also keep track of the direct supervisor of each employee (who is another employee).
  - ❖ We want to keep track of the dependents of each employee for insurance purposes. We keep each dependent's first name, sex, birth date, and relationship to the employee.
- Figure 7.2 shows how the schema for this database application can be displayed by means of the graphical notation known as **ER diagrams**.



**Figure 7.2**

An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter and is summarized in Figure 7.14.

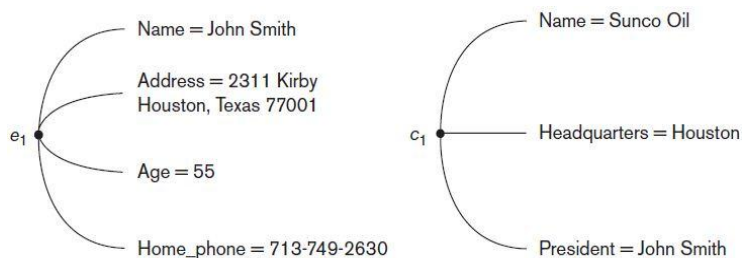
# UNIT – I

## 3.3 ) Entity Types, Entity Sets, Attributes, and Keys

The ER model describes data as *entities*, *relationships*, and *attributes*.

### 3.3.1 Entities and Attributes:

- An **entity** may be a **thing** or **object that can exist in the real world** (example, a person, a house, employee, a company, or a job).
- An **attribute** represents some **property that further describes an entity**, for example, an EMPLOYEE entity may be described by the employee's name, age, address, salary, and job.
- A particular entity will have a value for each of its attributes.
- Figure 3.3 shows two entities and the values of their attributes.



**Figure 7.3**  
Two entities, EMPLOYEE  $e_1$ , and COMPANY  $c_1$ , and their attributes.

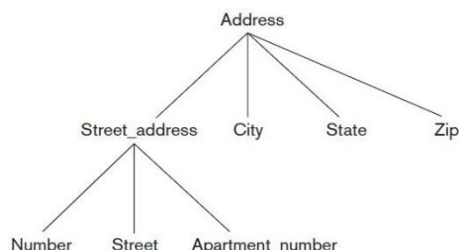
**Figure 3.3** Two entities, EMPLOYEE  $e_1$ , and COMPANY  $c_1$ , and their attributes.

- Several types of attributes occur in the ER model:

- simple versus composite*
- single valued versus multivalued*
- Stored versus derived.*
- NULL value*
- Complex Attributes*

#### i) Simple (Atomic) versus Composite Attributes:

- An attribute **having only one part** is called simple (or atomic) attribute. For example, age of a person.
- An attribute can be **divided into smaller subparts** is called **Composite attributes**. For example, Name or Address attribute of Employee entity shown in Figure 7.3.
- Composite attributes **can form a hierarchy**; for example, Street\_address can be further subdivided into three simple component attributes: Number, Street, and Apartment\_number, as shown in Figure 3.4.



**Figure 3.4** A hierarchy of composite attributes.

#### ii) Single-Valued versus Multivalued Attributes:

- An attributes have a single value for a particular entity is are called **single-valued**. For example, gender of employee.
- An attribute can have a set of values for the same entity called **multivalued**. For example a Colors of a car, or a College\_degrees for a person.
- A multivalued attribute may **have lower and upper bounds** to constrain the *number of values* allowed for each individual entity.

#### iii) Stored versus Derived Attributes:

- In some cases, two (or more) attribute values are related. For example, the Age and Birth\_date attributes of a person.
- For a particular person entity, the value of Age can be determined from the current (today's) date and the value of that person's Birth\_date.
- The Age attribute is hence called a **derived attribute** and is said to be **derivable from** the Birth\_date attribute, which is called a **stored attribute**.

## UNIT – I

### iv) NULL Values:

- In some cases, a particular entity may not have an applicable value for an attribute that value is called **NULL**. For example middle name of a person.
- NULL can also be used if we **do not know**( or **unknown**) the value of an attribute for a particular entity
- The **unknown** category of NULL can be further classified into two cases.
  - ✓ The first case arises **when it is known** that the attribute value exists but is **missing** - for example, if the Height attribute of a person is listed as NULL.
  - ✓ The second case arises **when it is not known** whether the attribute value exists - for example, Minit of a person is NULL.

### v) Complex Attributes:

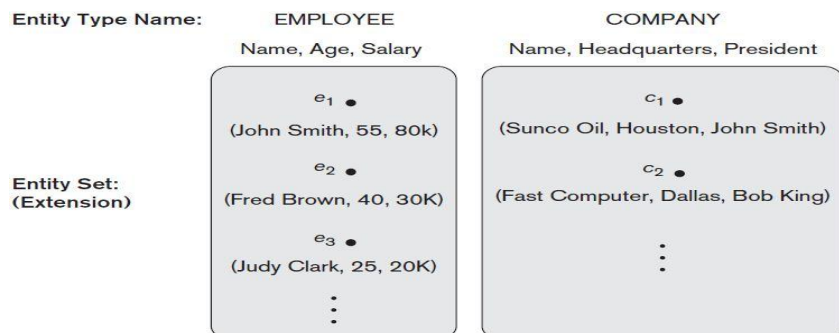
- **Composite** and **multivalued** attributes can be nested arbitrarily.
- We can represent arbitrary nesting by grouping components of a composite attribute between parentheses ( ) and separating the components with commas, and by displaying multivalued attributes between braces { }. Such attributes are called **complex attributes**.

```
{ Address_phone ( { Phone(Area_code, Phone_number) },
                  Address( Street_address ( Number, Street, Apartment_number, City,State,Zip) ) ) }
```

### 3.3.2 Entity Types, Entity Sets, Keys, and Value Sets

#### Entity Types and Entity Sets

- An **entity type** defines a **collection (or set) of entities that have the same attributes**. Each entity type in the database is described by its name and attributes.
- An **entity set** defines a **collection of all entities of a particular entity type in the database at any point in time (or extension)**.
- The entity set is usually referred to using the same name as the entity type.
- **The entity set is usually referred to using the same name as the entity type.**
- For example, **EMPLOYEE** refers to both a type of entity as well as the current set of all employee entities in the database.



**Figure 3.6** Two entity types, EMPLOYEE and COMPANY, and some member entities of each.

- An entity type is represented in ER diagrams (see Figure 7.2) as a rectangular box enclosing the entity type name.
- Attribute names are enclosed in ovals and are attached to their entity type by straight lines.
- Composite attributes are attached to their component attributes by straight lines.
- Multivalued attributes are displayed in double ovals.

#### Key Attributes of an Entity Type:

- An attribute values can be used to identify each entity uniquely. Such an attribute is called a **key attribute**.
- For example, the Name attribute is a key of the COMPANY entity type, because no two companies are allowed to have the same name.
- Sometimes several attributes together form a key is called **composite key** (meaning that the *combination* of the attribute values must be distinct for each entity).

## UNIT – I

- A composite key must be *minimal*.
- In ER diagrammatic notation, each key attribute has its name **underlined** inside the oval.
- An entity types have *more than one* key attribute (example, Vehicle\_id and Registration attributes of the entity type CAR)
- An entity type may also have *no key*, in which case it is called a *weak entity type*.

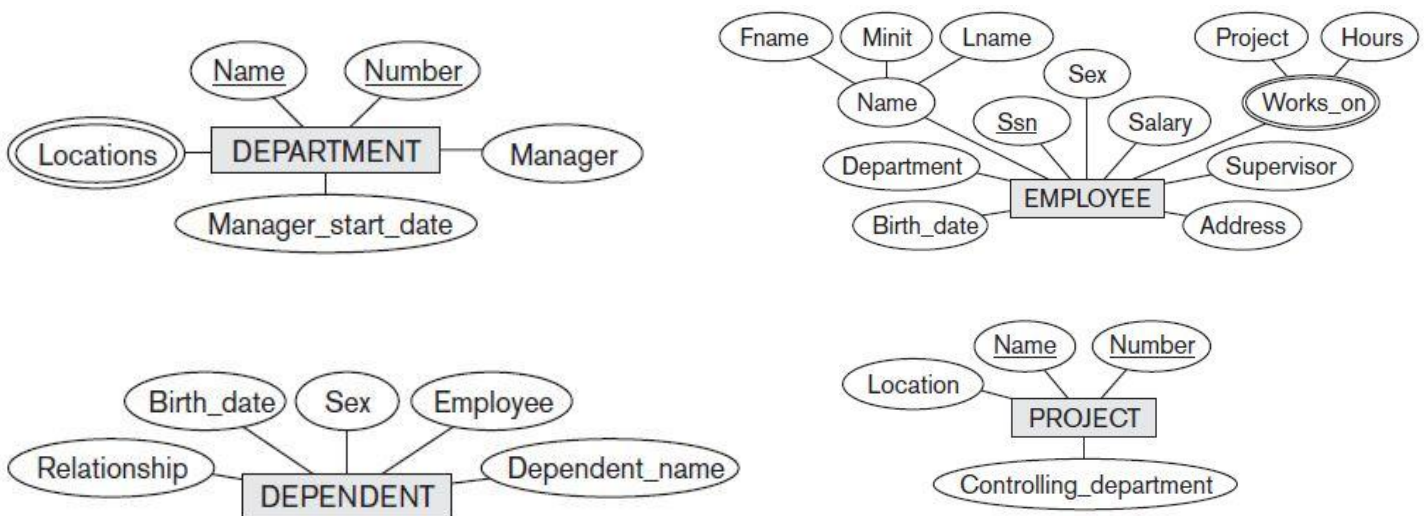
### Value Sets (Domains) of Attributes

- The set of values that can be assigned to that attribute for each individual entity is called **value set** (or **domain** of values).
- For example, if the range of ages allowed for employees is between 16 and 70, we can specify the value set of the Age to be the set of integer numbers between 16 and 70.
- Value sets are not displayed in ER diagrams, and are typically specified using the basic **data types** available in most programming languages, such as integer, string, Boolean etc.
- Mathematically, an attribute  $A$  of entity set  $E$  whose value set is  $V$  can be defined as a **function** from  $E$  to the power set<sup>6</sup>  $P(V)$  of  $V$ :

$$A : E \rightarrow P(V)$$

### 3.3.3 Initial Conceptual Design of the COMPANY Database

We can identify four entity types—one corresponding to each of the four items in the specification (see figure 3.8):



**Figure 3.8** Preliminary design of entity types for the COMPANY database.

Some of the shown attributes will be refined into relationships.

1. An entity type DEPARTMENT with attributes Name, Number, Locations, Manager, and Manager\_start\_date. Locations is the only multivalued attribute. We can specify that both Name and Number are (separate) key attributes because each was specified to be unique.
2. An entity type PROJECT with attributes Name, Number, Location, and Controlling\_department. Both Name and Number are (separate) key attributes.
3. An entity type EMPLOYEE with attributes Name, Ssn, Sex, Address, Salary, Birth\_date, Department, and Supervisor. Both Name and Address may be composite attributes; however, this was not specified in the requirements. We must go back to the users to see if any of them will refer to the individual components of Name—First\_name, Middle\_initial, Last\_name—or of Address.
4. An entity type DEPENDENT with attributes Employee, Dependent\_name, Sex, Birth\_date, and Relationship (to the employee).

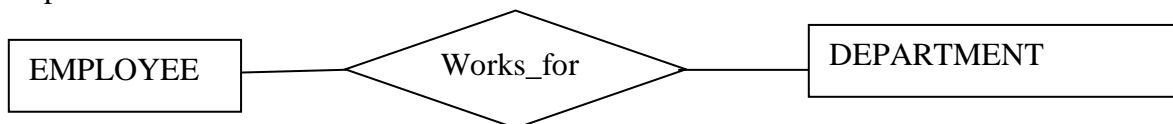
## UNIT – I

### 3.4) Relationship Types, Relationship Sets, Roles, and Structural Constraints

- Whenever an attribute of one entity type refers to another entity type, some relationship exists. For example, the attribute Manager of DEPARTMENT refers to an employee who manages the department.
- These references should not be represented as attributes but as **relationships**.
- Relationship is an association between several entities.

#### 3.4.1 Relationship Types, Sets, and Instances

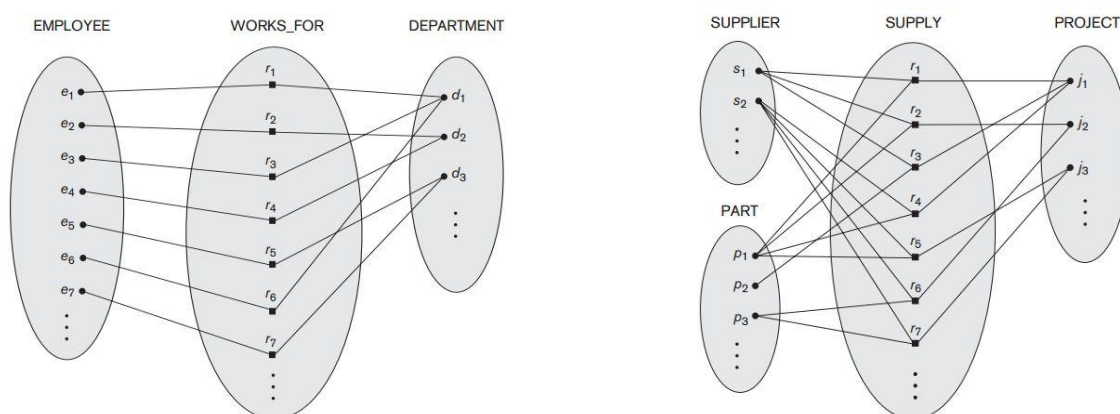
- **Relationship set** is collection of relationships of the same entity type.
- **Relationship type R** among  $n$  entity types  $E_1, E_2, \dots, E_n$  defines a set of associations (**relationship set**) among entities from these entity types.
- Each of the entity types  $E_1, E_2, \dots, E_n$  is said to **participate** in the relationship type R.
- Similarly, each of the individual entities  $e_1, e_2, \dots, e_n$  is said to **participate** in the relationship instance  $r_i = (e_1, e_2, \dots, e_n)$ .
- Each relationship instance  $r_i$  in R is an association of entities, where the **association includes exactly one entity from each participating entity type**.
- In ER diagrams, relationship types are displayed as **diamond-shaped boxes, which are connected by straight lines to the rectangular boxes representing the participating entity types**.
- The relationship name is displayed in the diamond-shaped box.
- Example:



#### 3.4.2 Relationship Degree, Role Names, and Recursive Relationships

##### Degree of a Relationship Type:

- The degree of a relationship type is **the number of participating entity types**.
- A relationship type of degree two is called **binary**. An example for binary relationship is WORKS\_FOR. Where each relationship instance  $r_i$  associates two entities—a employee  $e$ , and a department  $d$ .
- A relationship type of degree three is called **ternary**. An example of a ternary relationship is SUPPLY.
- Where each relationship instance  $r_i$  associates three entities—a supplier  $s$ , a part  $p$ , and a project  $j$ .



- Relationships can generally be of any degree, but the ones most common are binary relationships. Higher-degree relationships are generally more complex than binary relationships

##### Role Names and Recursive Relationships:

- Each entity type that participates in a relationship type plays a particular role in the relationship.
- The **role name** signifies the role that a participating entity from the entity type plays in each relationship instance, and helps to explain what the relationship means.
- Example, in the WORKS\_FOR relationship type, EMPLOYEE plays the role of employee or worker.

## UNIT – I

- In some cases the *same* entity type participates more than once in a relationship type in different roles. In such cases the role name becomes essential for distinguishing the meaning of the role that each participating entity plays. Such relationship types are called **recursive relationships**.
- Figure 3.11 shows an example. The SUPERVISION relationship type relates an employee to a supervisor, where both employee and supervisor entities are members of the same EMPLOYEE entity set. Hence, the EMPLOYEE entity type participates twice in SUPERVISION: once in the role of supervisor (or boss), and once in the role of supervisee (or subordinate).

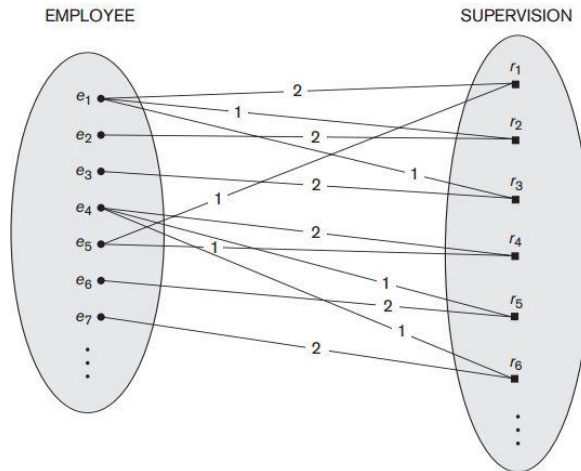


Figure 3.11 A recursive relationship SUPERVISION between EMPLOYEE in the supervisor role (1) and EMPLOYEE in the subordinate role (2).

### 3.4.3 Constraints on Binary Relationship Types

- Relationship types usually have certain constraints that limit the possible combinations of entities that may participate in the corresponding relationship set.
- We can distinguish two main types of binary relationship constraints: **cardinality ratio** and **participation**.

#### Cardinality Ratios for Binary Relationships:

- The cardinality ratio for a binary relationship specifies the **maximum number of relationship instances that an entity can participate in**.
- For example, in the WORKS\_FOR binary relationship type, **DEPARTMENT: EMPLOYEE** is of cardinality ratio 1:N, meaning that each department can be related to (that is, employs) any number of employees, but an employee can be related to (work for) only one department.
- The possible cardinality ratios for binary relationship types are **1:1**, **1:N**, **N:1**, and **M:N**.
- **1:1**:- An entity having only one relationship with another entity. Example MANAGES.
- **1:N**:- An entity having many relationship with another entities.
- **N:1**:- Many entities having only one relationship with another entity.
- **M:N**:- Many entities having many relationship with another entities. Example WORKS\_ON.

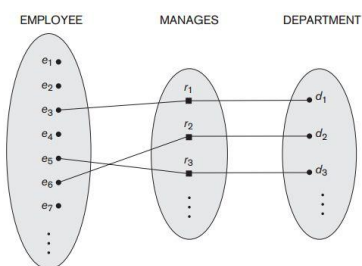


Figure 3.12 A 1:1 relationship, MANAGES.

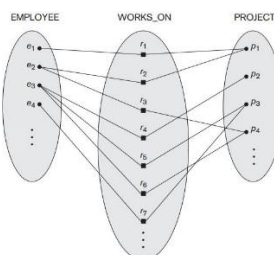


Figure 3.13 An M:N relationship, WORKS\_ON.

#### Participation Constraints and Existence Dependencies:

- This constraint specifies the **minimum number of relationship instances that each entity can participate in**.
- Sometimes called the **minimum cardinality constraint**.
- There are two types of participation constraints: **total** and **partial**.

## UNIT – I

- **Total participation (existence dependency):** if an entity in ‘E’ participate in at least one relation in ‘R’. For example EMPLOYEE in WORKS\_FOR, the total set of employee entities must be related to a department entity via WORKS\_FOR.
- **Partial participation:** some of entities in ‘E’ participate in relation in ‘R’. For example EMPLOYEE in the MANAGES relationship type is partial, meaning that some or part of the set of employee entities are related to some department entity via MANAGES, but not necessarily all.
- We will refer to the cardinality ratio and participation constraints, taken together, as the **structural constraints** of a relationship type.
- In ER diagrams, total participation (or existence dependency) is displayed as a double line connecting the participating entity type to the relationship, whereas partial participation is represented by a single line.



### 3.4.4) Attributes of Relationship Types

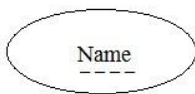
- Relationship types can also have attributes, similar to those of entity types. For example, to record the number of hours per week that an employee works on a particular project, we can include an attribute Hours for the WORKS\_ON relationship type.
- Notice that attributes of **1:1** or **1:N relationship types** can be migrated to one of the participating entity types. For example, the Start\_date attribute for the MANAGES relationship can be an attribute of either EMPLOYEE or DEPARTMENT, although conceptually it belongs to MANAGES. This is because MANAGES is a 1:1 relationship, so every department or employee entity participates in at most one relationship instance. Hence, the value of the Start\_date attribute can be determined separately, either by the participating department entity or by the participating employee (manager) entity.
- For a 1:N relationship type, a relationship attribute can be migrated only to the entity type on the N-side of the relationship. For example, if the WORKS\_FOR relationship also has an attribute Start\_date that indicates when an employee started working for a department, this attribute can be included as an attribute of EMPLOYEE. This is because each employee works for only one department, and hence participates in at most one relationship instance in WORKS\_FOR.
- In both **1:1** and **1:N** relationship types, the decision where to place a relationship attribute is determined subjectively by the schema designer.
- For M:N relationship types, some attributes may be determined by the combination of participating entities in a relationship instance, not by any single entity. Such attributes must be specified as relationship attributes. An example is the Hours attribute of the M:N relationship WORKS\_ON the number of hours per week an employee currently works on a project is determined by an employee project combination and not separately by either entity.



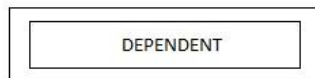
## UNIT – I

### 3.5) Weak Entity Types

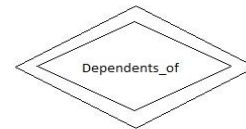
- ❖ An entity types that **do have a key attribute** is called **strong entity types**.
- ❖ An entity types that **do not have key attributes** of their own are called **weak entity types**.
- ❖ Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with one of their attribute values.
- ❖ We call this other entity type the **identifying** or **owner entity type**, and we call the relationship type that relates a weak entity type to its owner the **identifying relationship** of the weak entity type.
- ❖ For example in ER mode **DEPENDENT**, related to **EMPLOYEE**, **EMPLOYEE is owner entity** and **DEPENDENT identifying relationship**.
- ❖ A weak entity type always has a **total participation constraint** (existence dependency) with respect to its identifying relationship because a weak entity cannot be identified without an owner entity.
- ❖ However, not every existence dependency results in a weak entity type.
- ❖ A weak entity type normally has a **partial key**, which is the attribute that can uniquely identify weak entities that are related to the **same owner entity**.
- ❖ In our example, if we assume that no two dependents of the same employee ever have the same first name, the attribute Name of **DEPENDENT** is the **partial key**.
- ❖ Weak entity types can sometimes be represented as **complex** (composite, multivalued) attributes.
- ❖ In the worst case, a composite attribute of all the weak entity's attributes will be the partial key.
- ❖ In ER diagrams, both a weak entity type and its identifying relationship are distinguished by surrounding their boxes and diamonds with double lines.
- ❖ The partial key attribute is underlined with a dashed or dotted line.



Partial key attribute



Weak entity type



identifying relationship

### 3.6) Refining the ER Design for the COMPANY Database

- We can now refine the database design in Figure 3.8 by changing the attributes that represent relationships into relationship types.
- The **cardinality ratio** and **participation constraint** of each relationship type are determined from the requirements listed in Section 7.2.
- In our example, we specify the following **relationship types**:
  - ❖ **MANAGES**, a 1:1 relationship type between **EMPLOYEE** and **DEPARTMENT**. **EMPLOYEE** participation is **partial**. **DEPARTMENT** participation is not clear from the requirements, which implies **total** participation.
  - ❖ **WORKS FOR**, a 1:N relationship type between **DEPARTMENT** and **EMPLOYEE**. Both participations are **total**.
  - ❖ **CONTROLS**, a 1:N relationship type between **DEPARTMENT** and **PROJECT**. The participation of **PROJECT** is **total**, whereas that of **DEPARTMENT** is determined to be **partial**.
  - ❖ **SUPERVISION**, a 1:N relationship type between **EMPLOYEE** (in the supervisor role) and **EMPLOYEE** (in the supervisee role). **Both participations are determined to be partial**.
  - ❖ **WORKS ON**, an M:N relationship type with attribute **Hours**, after the users indicate that a project can have several employees working on it. **Both participations are determined to be total**.
  - ❖ **DEPENDENTS OF**, a 1:N relationship type between **EMPLOYEE** and **DEPENDENT**. The participation of **EMPLOYEE** is **partial**, whereas that of **DEPENDENT** is **total**.
- It is important to have the least possible redundancy when we design the conceptual schema of a database.

## UNIT – I

### 3.7) ER Diagrams, Naming Conventions, and Design Issues



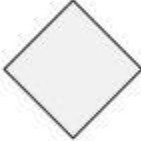
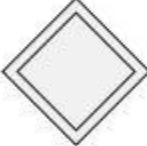



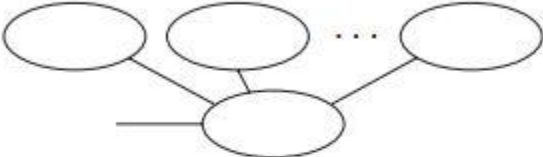
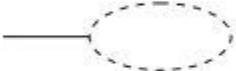
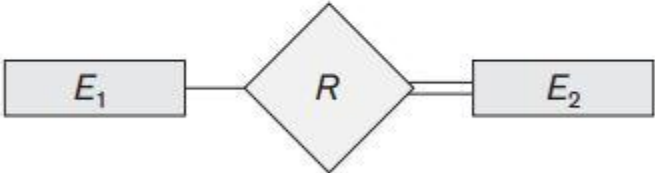
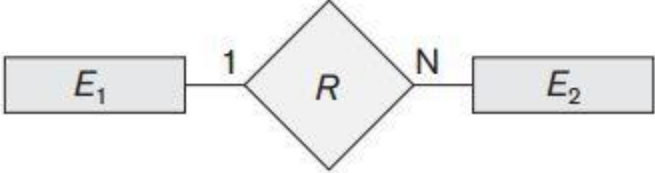
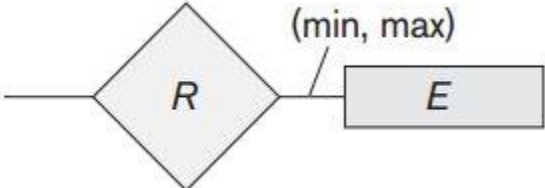
#### 3.7.1 Summary of Notation for ER Diagrams

- ❖ Figure 7.2 displays the COMPANY ER database schema as an ER diagram. We now review the full ER diagram notation. Entity types such as EMPLOYEE, DEPARTMENT, and PROJECT are shown in rectangular boxes. Relationship types such as WORKS\_FOR, MANAGES, CONTROLS, and WORKS\_ON are shown in diamond-shaped boxes attached to the participating entity types with straight lines. Attributes are shown in ovals, and each attribute is attached by a straight line to its entity type or relationship type. Component attributes of a composite attribute are attached to the oval representing the composite attribute, as illustrated by the Name attribute of EMPLOYEE. Multivalued attributes are shown in double ovals, as illustrated by the Locations attribute of DEPARTMENT. Key attributes have their names underlined. Derived attributes are shown in dotted ovals, as illustrated by the Number\_of\_employees attribute of DEPARTMENT.
- ❖ Weak entity types are distinguished by being placed in double rectangles and by having their identifying relationship placed in double diamonds, as illustrated by the DEPENDENT entity type and the DEPENDENTS\_OF identifying relationship type. The partial key of the weak entity type is underlined with a dotted line.
- ❖ In Figure 7.2 the cardinality ratio of each binary relationship type is specified by attaching a 1, M, or N on each participating edge.

#### 3.7.2 Proper Naming of Schema Constructs

- When designing a database schema, the choice of names for entity types, attributes, relationship types, and (particularly) roles is not always straightforward.
  - ❖ One should choose names that convey, as much as possible, the meanings attached to the different constructs in the schema. We choose to use singular names for entity types, rather than plural ones, because the entity type name applies to each individual entity belonging to that entity type.
  - ❖ Another naming consideration involves choosing binary relationship names to make the ER diagram of the schema readable from left to right and from top to bottom.
- Figure 7.2—the DEPENDENTS\_OF relationship type, which reads from bottom to top. When we describe this relationship, we can say that the DEPENDENT entities (bottom entity type) are DEPENDENTS\_OF (relationship name) an EMPLOYEE (top entity type).
- To change this to read from top to bottom, we could rename the relationship type to HAS\_DEPENDENTS, which would then read as follows: An EMPLOYEE entity (top entity type) HAS\_DEPENDENTS (relationship name) of type DEPENDENT (bottom entity type). Notice that this issue arises because each binary relationship can be described starting from either of the two participating entity types.

# UNIT - I

Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Identifying Relationship
	Attribute
	Key Attribute
	Multivalued Attribute
	Composite Attribute
	Derived Attribute
	Total Participation of $E_2$ in $R$
	Cardinality Ratio 1: N for $E_1:E_2$ in $R$
	Structural Constraint (min, max) on Participation of $E$ in $R$

## UNIT – I

### 3.7.3 Design Choices for ER Conceptual Design

In general, the schema design process should be considered an iterative refinement process, where an initial design is created and then iteratively refined until the most suitable design is reached.

Some of the refinements that are often used include the following:

- A concept may be first modeled as an attribute and then refined into a relationship because it is determined that the attribute is a reference to another entity type. It is often the case that a pair of such attributes that are inverses of one another are refined into a binary relationship. It is important to note that in our notation, once an attribute is replaced by a relationship, the attribute itself should be removed from the entity type to avoid duplication and redundancy.
- Similarly, an attribute that exists in several entity types may be elevated or promoted to an independent entity type. For example, suppose that several entity types in a UNIVERSITY database, such as STUDENT, INSTRUCTOR, and COURSE, each has an attribute Department in the initial design; the designer may then choose to create an entity type DEPARTMENT with a single attribute Dept\_name and relate it to the three entity types (STUDENT, INSTRUCTOR, and COURSE) via appropriate relationships.
- An inverse refinement to the previous case may be applied—for example, if an entity type DEPARTMENT exists in the initial design with a single attribute Dept\_name and is related to only one other entity type, STUDENT. In this case, DEPARTMENT may be reduced or demoted to an attribute of STUDENT.

### 7.7.4 Alternative Notations for ER Diagrams

- We describe one alternative ER notation for specifying structural constraints on relationships, which replaces the **cardinality ratio (1:1, 1:N, M:N)** and **single/double line notation for participation constraints**.
- This notation involves **associating a pair of integer numbers (min, max)** with each participation of an entity type E in a relationship type R, where  $0 \leq \min \leq \max$  and  $\max \geq 1$ .
- The numbers mean that for each entity e in E, e must participate in at least min and at most max relationship instances in R at any point in time.
- In this method, **min = 0 implies partial participation**, whereas **min > 0 implies total participation**.

