

## **Module 1**

**Module 1:** Introduction to Databases: Introduction, Characteristics of database approach, Advantages of using the DBMS approach, History of database applications. Overview of Database Languages and Architectures: Data Models, Schemas, and Instances. Three schema architecture and data independence, database languages, and interfaces, The Database System environment. Conceptual Data Modelling using Entities and Relationships: Entity types, Entity sets, attributes, roles, and structural constraints, Weak entity types, ER diagrams, examples, Specialization and Generalization.

### **Textbook:**

1. Fundamentals of Database Systems, Ramez Elmasri and Shamkant B. Navathe, 7th Edition, 2017, Pearson.
2. Database management systems, Ramakrishnan and Gehrke, 3rd Edition, 2014, McGraw Hill.

### **Introduction**

A **database** is a collection of related data.

A **data** mean known facts that can be recorded and that have implicit meaning.

For ex: consider the names, telephone numbers, and addresses of the people you know. You may have recorded this data in an indexed address book or you may have stored it on a hard drive, using a personal computer and software such as Microsoft Access or Excel. This collection of related data with an implicit meaning is a database.

A database has the following implicit properties:

- ✓ A database represents some aspect of the real world, sometimes called the miniworld 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. It has an intended group of users and some preconceived applications in which these users are interested.

Other properties:

- ✓ A database can be of any size and complexity. For example, the list of names and addresses

referred to earlier may consist of only a few hundred records, each with a simple structure. An example of a large commercial database is Amazon.com.

- ✓ A database may be generated and maintained manually or it may be computerized. For example, a library card catalog is a database that may be created and maintained manually. A computerized database may be created and maintained either by a group of application programs written specifically for that task or by a database management system.

A database management system (DBMS) is a collection of programs that enables users to create and maintain a database.

**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 by the DBMS 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 miniworld, 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.

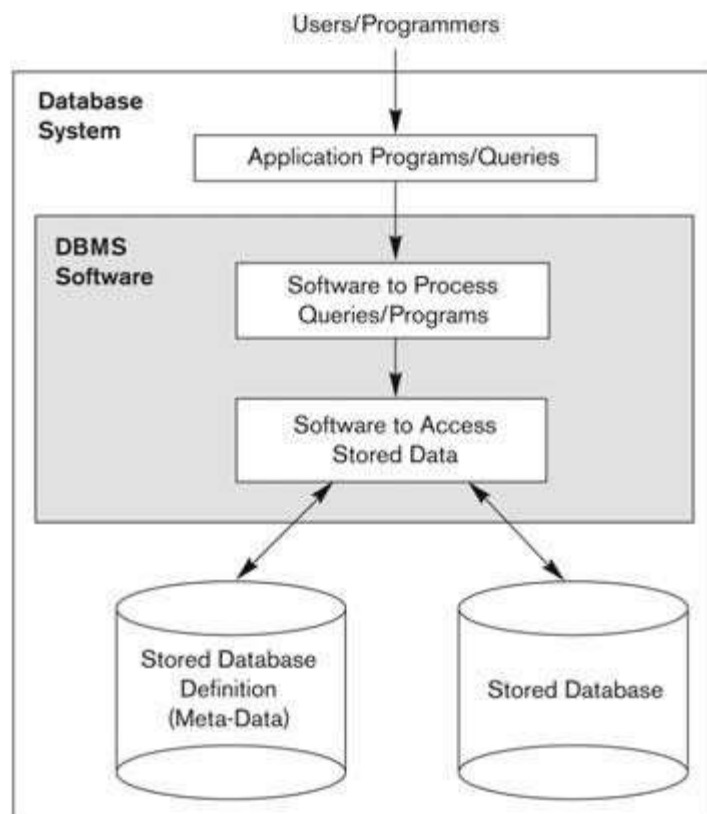
Other important functions provided by the DBMS include protecting the database and maintaining it over a long period of time.

**Protection** includes system protection against hardware or software malfunction (or crashes) and security protection against unauthorized or malicious access.

A typical large database may have a life cycle of many years, so the DBMS must be able to **maintain** the database system by allowing the system to evolve as requirements change over time.

Database and DBMS software together a **Database system**.

### A simplified database system environment



The Information Technology (IT) department within a company designs and maintains an information system consisting of various computers, storage systems, application software, and databases. Design of a new application for an existing database or design of a brand new database starts off with a phase called **requirements specification and analysis**. These requirements are documented in detail and transformed into a **conceptual design** that can be represented and manipulated using some computerized tools so that it can be easily maintained, modified, and transformed into a database implementation. The design is then translated to a **logical design** that can be expressed in a data model implemented in a commercial DBMS. The final stage is physical design, during which further specifications are provided for **storing and accessing** the database. The database design is implemented, populated with actual data, and continuously maintained to reflect the state of the miniworld.

### Characteristics of the Database Approach

In traditional file processing, each user defines and implements the files needed for a specific software application as part of programming the application. For example, one user, the grade reporting office, may keep files on students and their grades. Programs to print a student's transcript and to enter

new grades are implemented as part of the application. A second user, the accounting office, may keep track of students' fees and their payments. Although both users are interested in data about students, each user maintains separate files— and programs to manipulate these files—because each requires some data not available from the other user's files.

This redundancy in defining and storing data results in wasted storage space and in redundant efforts to maintain common up-to-date data.

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

- ✓ Self-describing nature of a database system
- ✓ Insulation between programs and data, and data abstraction
- ✓ Support of multiple views of the data
- ✓ Sharing of data and multiuser transaction processing

### **Self-Describing Nature of a Database System DB approach**

- A fundamental characteristic of the database approach is that 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.
- Whenever a request is made to access, say, the Name of a STUDENT record, the DBMS software refers to the catalog to determine the structure of the STUDENT file and the position and size of the Name data item within a STUDENT record

### **Traditional file processing**

- In traditional file processing, data definition is typically part of the application programs themselves. Hence, these programs are constrained to work with only one specific database, whose structure is declared in the application programs.
- In a typical file-processing application, the file structure and, in the extreme case, the exact location of Name within a STUDENT record are already coded within each program that accesses this data item

**Insulation between programs and data, and data abstraction**

- In traditional file processing, the structure of data files is embedded in the application programs, so any changes to the structure of a file may require changing all programs that access that file.
- For example, a file access program may be written in such a way that it can access only STUDENT records of the structure, If we want to add another piece of data to each STUDENT record, say the Birth\_date, such a program will no longer work and must be changed

**DB approach**

- By contrast, DBMS access programs do not require such changes in most cases. The structure of data files is stored in the DBMS catalog separately from the access programs. We call this property **program-data independence**.
  - By contrast, in a DBMS environment, we only need to change the description of STUDENT records in the catalog to reflect the inclusion of the new data item Birth\_date; no programs are changed. The next time a DBMS program refers to the catalog, the new structure of STUDENT records will be accessed and used.
- ✓ In an object-oriented and object-relational systems, An **operation** (also called a function or method) is specified in two parts. The **interface** (or signature) of an operation includes the operation name and the data types of its arguments (or parameters).
- ✓ The implementation (or method) of the operation is specified separately and can be changed without affecting the interface. User application programs can operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented. This may be termed **program-operation independence**.
- ✓ The characteristic that allows **program-data independence and program-operation independence** is called **data abstraction**.

**Support of Multiple Views of the Data**

- A database typically has many types of users, each of whom may require a different perspective or view of the database.
- 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.
- For example, one user of the database may be interested only in accessing and printing the

transcript of each student. A second user, who is interested only in checking that students have taken all the prerequisites of each course for which the student registers, may require the view.

### **Sharing of Data and Multiuser Transaction Processing**

- A multiuser DBMS, as its name implies, must allow multiple users to access the database at the same time.
- This is essential if data for multiple applications is to be integrated and maintained in a single database.
- A fundamental role of multiuser DBMS software is to ensure that concurrent transactions operate correctly and efficiently.
- 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.

### **Actors on the Scene**

#### **Database Administrators**

- ✓ In a database environment, the primary resource is the database itself, and the secondary resource is the DBMS and related software.
- ✓ Administering these resources is the responsibility of the database administrator (DBA).
- ✓ The 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. In large organizations, the DBA is assisted by a staff that carries out these functions.

#### **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.
- ✓ 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.
- ✓ Database designers typically interact with each potential group of users and develop views of the

database that meet the data and processing requirements of these groups. Each view is then analyzed and integrated with the views of other user groups. The final database design must be capable of supporting the requirements of all user groups.

## End Users

End users are the people whose jobs require access to the database for querying, updating, and generating report.

There are several categories of end users:

■ **Casual end users** occasionally access the database, but they may need different information each time. They use a sophisticated database query language to specify their requests and are typically middle- or high-level managers or other occasional browsers.

■ **Naive or parametric end users** make up a sizable portion of database end users. Their main job function revolves around constantly querying and updating the database, using standard types of queries and updates—called **canned transactions**—that have been carefully programmed and tested.

The tasks that such users perform are varied:

- ✓ Bank tellers check account balances and post withdrawals and deposits.
- ✓ Reservation agents for airlines, hotels, and car rental companies check availability for a given request and make reservations.
- ✓ shipping clerks (e.g., at UPS) who use buttons, bar code scanners, etc., to update status of in-transit packages.

■ **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.

■ **Stand-alone users** Use "personal" databases, possibly employing a special purpose (e.g., financial) software package. Mostly maintain personal databases using ready-to-use packaged applications.

An example is a tax program user that creates its own internal database. Another example is maintaining an address book.

## System Analysts, Application Programmers, Software Engineers:

**System Analysts:** determine needs of end users, especially naive and parametric users, and develop specifications for canned transactions that meet these needs.

**Application Programmers:** Implement, test, document, and maintain programs that satisfy the specifications mentioned above.

### **Workers behind the Scene**

Persons are typically not interested in the database content itself known as workers behind the scene, and they include the following categories:

**DBMS system designers and implementers** design and implement the DBMS modules and interfaces as a software package.

**Tool developers design and implement tools**—the software packages that facilitate database modeling and design, database system design, and improved performance. They include packages for database design, performance monitoring, natural language or graphical interfaces, prototyping, simulation, and test data generation.

**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.

### **Advantages of Using the DBMS Approach**

#### **1. Controlling Redundancy:**

Data redundancy (such as tends to occur in the "file processing" approach) leads to wasted storage space, duplication of effort (when multiple copies of a datum need to be updated), and a higher likelihood of the introduction of inconsistency.

On the other hand, redundancy can be used to improve performance of queries. Indexes, for example, are entirely redundant, but help the DBMS in processing queries more quickly.

A DBMS should provide the capability to automatically enforce the rule that no inconsistencies are introduced when data is updated.

#### **2. Restricting Unauthorized Access:**

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:**

Object-oriented database systems make it easier for complex runtime objects (e.g., lists, trees) to be saved in secondary storage so as to survive beyond program termination and to be retrievable at a later time.

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

Database systems must provide capabilities for efficiently executing queries and updates. 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. The recovery subsystem could ensure that the transaction is resumed from the point at which it was interrupted so that its full effect is recorded in the database.

### 6. Providing Multiple User Interfaces:

Many types of users with varying levels of technical knowledge use a database, a DBMS should provide a variety of user interfaces. For example, query languages for casual users, programming language interfaces for application programmers, forms and/or command codes for parametric users, menu-driven interfaces for stand-alone 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. The simplest type of integrity constraint involves specifying a data type for each data item.

A more complex type of constraint that frequently occurs involves specifying that a record in one file must be related to records in other files. This is known as a referential integrity constraint.

Another type of constraint specifies uniqueness on data item values, this is known as a key or uniqueness constraint.

### 9. Permitting Inferencing and Actions Via Rules:

In a **deductive** database system, one may specify *declarative* rules that allow the database to infer new data! E.g., Figure out which students are on academic probation. Such capabilities would take the place of application programs that would be used to ascertain such information otherwise.

**Active** database systems go one step further by allowing "active rules" that can be used to initiate actions automatically.

**A Brief History of Database Applications**

- Early Database Applications:
  - ✓ The Hierarchical and Network Models were introduced in mid 1960s and dominated during the 70's
  - ✓ A bulk of the worldwide database processing still occurs using these models
- Relational Model based Systems:
  - ✓ Relational model was originally introduced in 1970, was heavily researched and experimented with in IBM Research and several universities

Object-oriented and emerging applications: Object-Oriented Database Management Systems (OODBMSs) were introduced in late 1980s and early 1990s to cater to the need of complex data processing in CAD and other applications.

- Their use has not taken off much.

Many relational DBMSs have incorporated object database concepts, leading to a new category called object- relational DBMSs (ORDBMSs)

Extended relational systems add further capabilities (e.g. for multimedia data, XML, and other data types)

- Relational DBMS Products emerged in the 1980s
- Data on the Web and E-commerce Applications
- Web contains data in HTML (Hypertext markup language) with links among pages.
- This has given rise to a new set of applications and E-commerce is using new standards like XML (eXtended Markup Language)
- Script programming languages such as PHP and JavaScript allow generation of dynamic Web pages that are partially generated from a database
- New functionality is being added to DBMSs in the following areas:
  - ✓ Scientific Applications
  - ✓ XML(extensible Markup Language)
  - ✓ Image Storage and Management
  - ✓ Audio and Video data management
  - ✓ Data Warehousing and Data Mining
  - ✓ Spatial data management
  - ✓ Time Series and Historical Data Management
  - ✓ The above gives rise to new research and development in incorporating new data types,

complex data structures, new operations and storage and indexing schemes in database systems.

- ✓ Also allow database updates through Web pages

### **When Not to Use a DBMS**

- Main inhibitors (costs) of using a DBMS:
  - ✓ High initial investment and possible need for additional hardware.
  - ✓ Overhead for providing generality, security, concurrency control, recovery, and
  - ✓ When a DBMS may be unnecessary:
    - ✓ If the database and applications are simple, well defined, and not expected to change.
    - ✓ If there are stringent real-time requirements that may not be met because of DBMS overhead.
    - ✓ If access to data by multiple users is not required.
  - ✓ When no DBMS may suffice:
    - ✓ If the database system is not able to handle the complexity of data because of modeling limitations.
    - ✓ If the database users need special operations not supported by the DBMS.

## **OVERVIEW OF DATABASE LANGUAGES AND ARCHITECTURES:**

### **Data Models, Schemas, and Instances**

- One fundamental characteristic of the database approach is that it provides some level of data abstraction.
- **Data abstraction** generally refers to the ***suppression of details of data organization and storage***, and the highlighting of the essential features for an improved understanding of data.
- One of the main characteristics of the database approach is to support data abstraction so that different users can perceive data at their preferred level of detail.
- A **data model**—a collection of concepts that can be used to describe the ***structure of a database***—provides the necessary means to achieve this abstraction
- By structure of a database means the ***data types, relationships, and constraints*** that apply to the data. Most data models also include a set of basic operations for specifying retrievals and updates on the database.

## Categories of Data Models

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

- provide concepts that are close to the way many users perceive data
- use concepts such as **entities, attributes, and relationships**.
  - ✓ An **entity** represents a real-world object or concept, such as an employee or a project from the miniworld that is described in the database.
  - ✓ An **attribute** represents some property of interest that further describes an entity, such as the employee's name or salary.
  - ✓ A **relationship** among two or more entities represents an association among the entities, for example, a works-on relationship between an employee and a project.

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

- provide concepts that describe the details of how data is stored on the computer storage media, typically magnetic disks.
- Concepts provided by low-level data models are generally meant for computer specialists, not for end users

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

- 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
- in between high level and low level
- hide many details of data storage on disk but can be implemented on a computer system directly

## Schemas

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**.

Object in the schema—such as STUDENT or COURSE—a **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.

The actual data in a database may change quite frequently. Changes every time we add a new student or enter a new grade.

The data in the database at a particular moment in time is called a **database state or SNAPSHOT**. It is also called the **current set of occurrences or instances** in the database.

In a given database state, each schema construct has its own current set of instances; for example, the STUDENT construct will contain the set of individual student entities (records) as its instances.

#### STUDENT

Name	StudentNumber	Class	Major
------	---------------	-------	-------

#### COURSE

CourseName	CourseNumber	CreditHours	Department
------------	--------------	-------------	------------

#### PREREQUISITE

CourseNumber	PrerequisiteNumber
--------------	--------------------

#### SECTION

SectionIdentifier	CourseNumber	Semester	Year	Instructor
-------------------	--------------	----------	------	------------

#### GRADE\_REPORT

StudentNumber	SectionIdentifier	Grade
---------------	-------------------	-------

**Fig: Schema diagram for the University database**

When we define a new database, we specify its database schema only to the DBMS. At this point, 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. From then on, every time an update operation is applied to the database, we get another **database state**.

At any point in time, the database has a **current state**.

The DBMS is partly responsible for ensuring that every state of the database is a **valid state**—that is, a state that satisfies the structure and constraints specified in the schema.

The DBMS stores the descriptions of the schema constructs and constraints—also called the **meta-data**—in the DBMS catalog.

The schema is sometimes called the **INTENSION**, and a database state is called an **EXTENSION** of the schema.

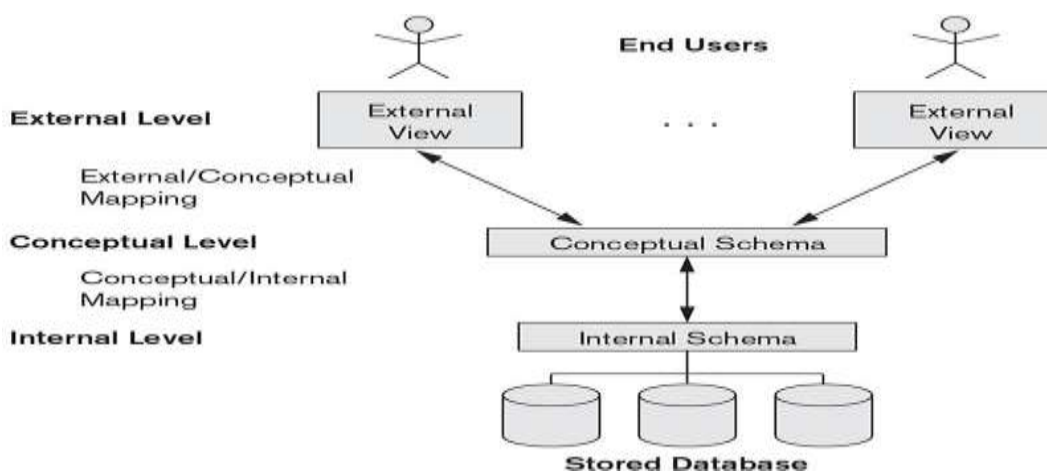
Application requirements change occasionally, which is one of the reasons why software maintenance is important. On such occasions, a change to a database's schema may be called for. An example would

be to add a Date\_of\_Birth field/attribute to the STUDENT table. Making changes to a database schema is known as **SCHEMA EVOLUTION**. Most modern DBMS's support schema evolution operations that can be applied while a database is operational.

### Three schema architecture and data independence:

#### The Three-Schema Architecture

The goal of the three-schema architecture is to separate the user applications from the physical database.



In this architecture, schemas can be defined at the following three levels:

**1. The internal level has an internal schema**

- describes the physical storage structure of the database
- uses a 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,**

- describes the structure of the whole database for a community of users
- hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints
- representational data model is used to describe the conceptual schema when a database system is implemented
- 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,

- 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.
- As in the previous level, 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 a 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**.

These mappings may be time-consuming, so some DBMSs—especially those that are meant to support small databases—do not support external views.

Even in such systems, however, a certain amount of mapping is necessary to transform requests between the conceptual and internal levels.

## Data Independence

**Data independence**, which can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level.

We can define two types of data independence:

✓ **Logical data independence**

- capacity to change the conceptual schema without having to change external schemas or application programs
- change the conceptual schema to expand the database, to change constraints, or to reduce the database
- changes to constraints can be applied to the conceptual schema without affecting the external schemas or application programs

✓ **Physical data independence**

- capacity to change the internal schema without having to change the conceptual schema. Hence, the external schemas need not be changed as well
- by creating additional access structures—to improve the performance of retrieval or update.

## Database Languages and Interfaces DBMS Languages

DBMS packages provide an integrated feature of above languages into a single language called **Structured Query Language**.

- ✓ **Data definition language (DDL)**, is used by the DBA and by database designers to define both schemas.
- ✓ **Storage definition language (SDL)**, is used to specify the internal schemas.
- ✓ **View definition language (VDL)**, to specify user views and their mappings to the conceptual schema.
- ✓ **Data manipulation language (DML)** provides set of operations like retrieval, insertion, deletion, and modification of the data. There are two main types of **DMLs**. A **high-level or nonprocedural**.

### DML

- ✓ can be used on its own to specify complex database operations concisely Many DBMS
- ✓ 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.
- ✓ can specify and retrieve many records in a single DML statement; therefore, they are called **set-at-a-time or set- oriented DMLs**
- ✓ declarative language A low level or procedural **DML**
- ✓ must be embedded in a general-purpose programming language
- ✓ retrieves individual records or objects from the database and processes each separately
- ✓ to retrieve and process each record from a set of records. Low-level DMLs are also called **record-at-a-time DMLs**

Whenever DML commands, whether high level or low level, are embedded in a general-purpose programming language, 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**.

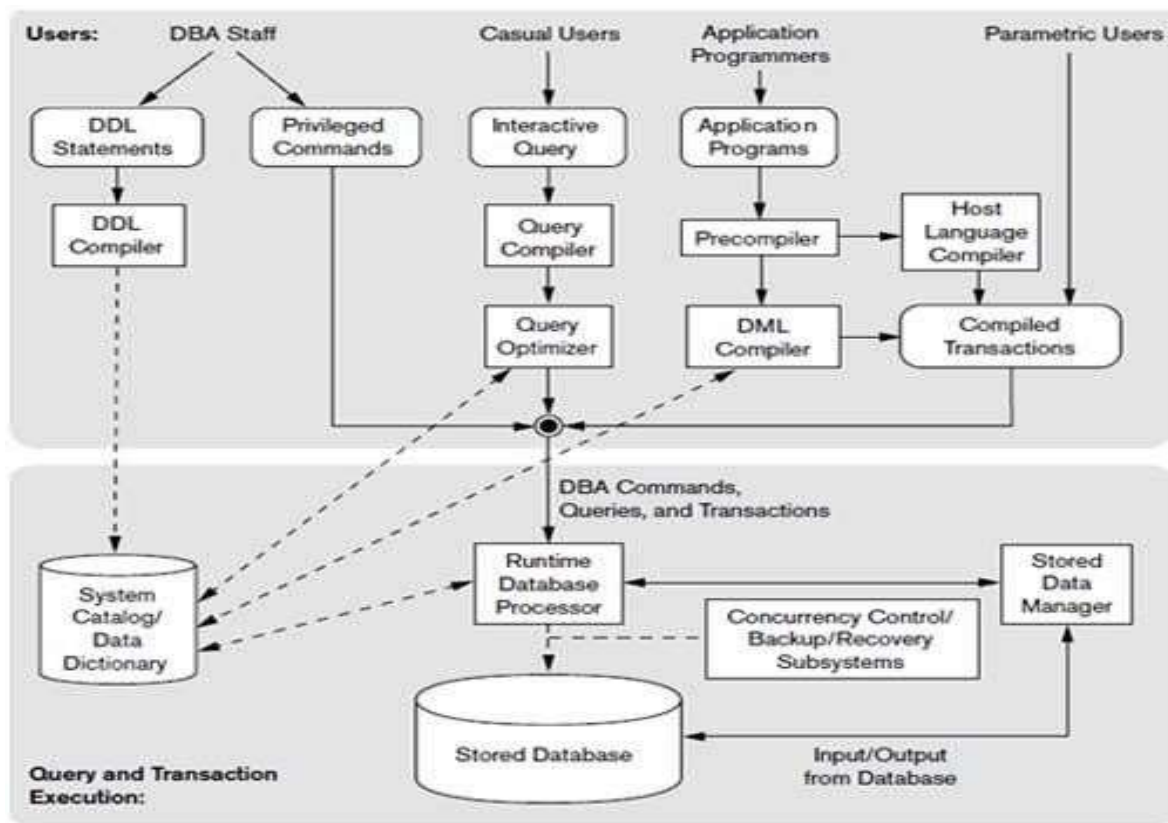
## DBMS Interfaces

- ✓ **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.



- ✓ **Forms-Based Interfaces** 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.
- ✓ **Graphical User Interfaces** displays a schema to the user in diagrammatic form. The user then can specify a query by manipulating the diagram. GUIs utilize both menus and forms. Most GUIs use a pointing device.
- ✓ **Natural Language Interfaces** accepts requests written in English or some other language and attempt to understand them.
- ✓ **Speech Input and Output** use of speech as an input query and speech as an answer to a question or result. The speech input is detected using a library of predefined words and used to setup the parameters that are supplied to the queries.
- ✓ **Interfaces for Parametric Users** such as bank tellers, often have a small set of operations that they must perform repeatedly.
- ✓ **Interfaces for the DBA.** DBA use privileged commands. These include commands for creating accounts, setting system parameters, granting account authorization, changing a schema, and reorganizing the storage structures of a database.

### The Database System Environment



The figure is divided into two parts.

The **top part** of the figure 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.

The database and the DBMS catalog are usually stored on disk. Access to the disk is controlled primarily by the operating system (OS), which schedules disk read/write.

Many DBMSs have their own **buffer management** module to schedule disk read/write.

**Stored data manager** controls access to DBMS information that is stored on disk, whether it is part of the database or the catalog.

**Top half figure:**

- ✓ it shows interfaces for the DBA staff, casual users who work with interactive interfaces to formulate queries
- ✓ application programmers who create programs using some host programming languages
- ✓ Parametric users who do data entry work by supplying parameters to predefined transactions.
- ✓ the DBA staff works on defining the database and tuning it by making changes to its definition using the DDL and other privileged commands

**DBA staff:**

- ✓ The DDL compiler processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta-data) in the DBMS catalog
- ✓ The catalog includes information such as the names and sizes of files, names and data types of data items, storage details of each file, mapping information among schemas, and constraints

**Casual users:**

- ✓ interact using some form of interface, which we call the interactive query interface
- ✓ queries are parsed and validated for correctness of the query syntax, the names of files and data elements, and so on by a query compiler that compiles them into an internal form.

This internal query is subjected to query optimization.

query optimizer is concerned with the rearrangement and possible reordering of operations, elimination of redundancies, and use of correct algorithms and indexes during execution.

- ✓ It 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

- ✓ precompiler extracts DML commands from an application program
- ✓ commands are sent to the DML compiler for compilation
- ✓ 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

An example is a bank withdrawal transaction where the account number and the amount may be supplied as parameters

In the **lower part of Figure,**

- ✓ the runtime database processor executes
  - the privileged commands
  - the executable query plans, and
  - 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 concurrency control and backup and recovery systems, integrated into the working of the runtime database processor for purposes of transaction management

### Database System Utilities

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

- ✓ Loading
  - used to load existing data files—such as text files or sequential files—into the database
  - automatically reformats the data and stores it in the database
  - for loading programs, conversion tools are available like IDMS (Computer Associates), SUPRA (Cincom), and IMAGE (HP)
- ✓ Backup
  - creates a backup copy of the database, by dumping the entire database onto tape
  - used to restore the database in case of catastrophic disk failure
  - Incremental backups are also often used, to save space
- ✓ Database storage reorganization
  - used to reorganize a set of database files into different file organizations to improve performance
- ✓ Performance monitoring

- monitors database usage and provides statistics to the DBA.
- Statistics used for making decisions

Other utilities may be available for sorting files, handling data compression, monitoring access by users, interfacing with the network, and performing other functions.

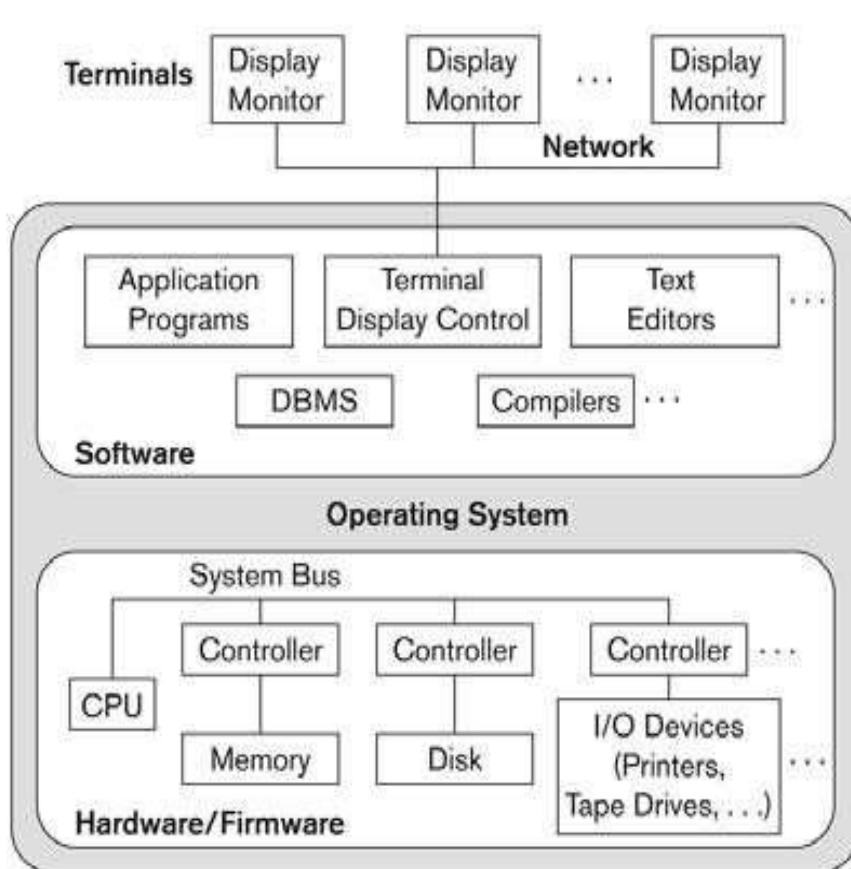
### Tools, Application Environments, and Communications Facilities

- ✓ **CASE** tools are used in the design phase of database systems
- ✓ **Data dictionary** (or data repository) **system** for storing catalog information about schemas and constraints
- ✓ **Information repository** stores information such as design decisions, usage standards, application program descriptions, and user information
- ✓ **Application development environments** systems provide an environment for developing database applications, 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
- ✓ The integrated DBMS and data communications system is called a **DB/DC system**

### Centralized and Client/Server Architectures for DBMSs:

#### Centralized DBMSs Architecture:

- ✓ Earlier architectures used mainframe computers to provide the main processing for all system functions
- ✓ These mainframes replaced by users with their terminals with PCs and workstations
- ✓ DB systems used these computers similarly to how they had used display terminals
- ✓ So that 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

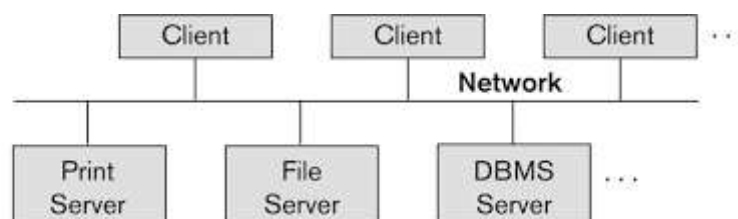


**Figure 2.4**  
A physical centralized architecture.

### Client/server DBMS architecture:

**Figure 2.5**

Logical two-tier client/server architecture.

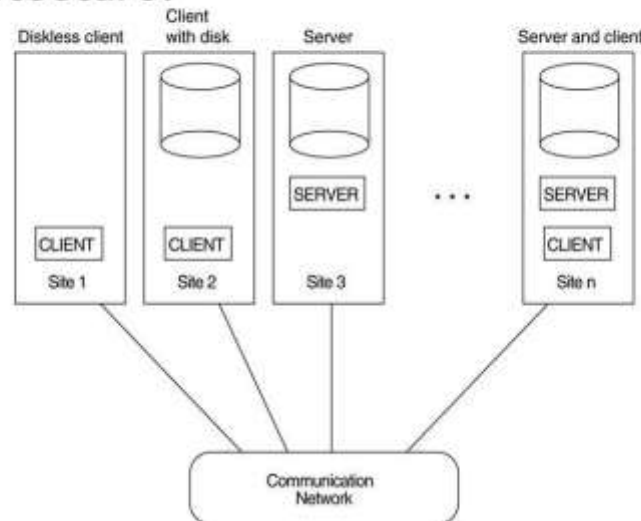


- ✓ The client/server architecture was developed to deal with computing environments in which a large number of PCs, workstations, file servers, printers, database servers, Web servers, e-mail servers, and other software and equipment are connected via a network.
- ✓ The idea is to define specialized servers with specific functionalities.
- ✓ it is possible to connect a number of PCs or small workstations as clients to a file server that maintains the files of the client machines.
- ✓ Another machine can be designated as a printer server by being connected to various printers; all print requests by the clients are forwarded to this machine.
- ✓ Web servers or e-mail servers also fall into the specialized server category. The resources

provided by specialized servers can be accessed by many client machines.

- ✓ The client machines provide the user with the appropriate interfaces to utilize these servers, as well as with local processing power to run local applications.
- ✓ This concept can be carried over to other software packages, with specialized programs—such as a CAD (computer-aided design) package.

### Physical two-tier client-server architecture.



#### Physical two-tier client/server architecture:

- ✓ Some machines would be client sites only, other machines would be dedicated servers, and others would have both client and server functionality
- ✓ The concept of client/server architecture assumes an underlying framework that 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 machine provides user machine that provides user interface capabilities and local processing
  - ✓ 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.
- ✓ In general, some machines install only client software, others only server software, and still others may include both client and server software

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

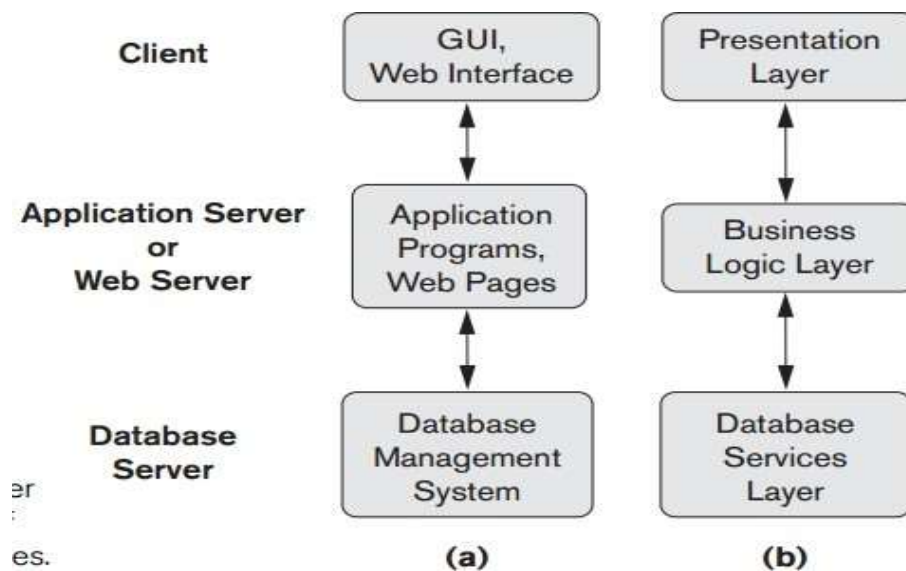
- ✓ in relational database management systems (RDBMSs), many of which started as centralized systems, the system components that were first moved to the client side were the user

interface and application programs

- ✓ 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
- ✓ in an RDBMS, the server is also often called an SQL server
- ✓ 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)
- ✓ The 2<sup>nd</sup> approach to two-tier client/server architecture was taken by some object-oriented DBMSs, where the software modules of the DBMS were divided between client and server
- ✓ The **server level** may include the part of the DBMS software responsible for handling data storage on disk pages, local concurrency control and recovery, buffering and caching of disk pages, and other such functions.
- ✓ the **client level** may handle the user interface, data dictionary functions, DBMS interactions with programming language compilers, global query optimization, concurrency control, and recovery across multiple servers, structuring of complex objects from the data in the buffers
- ✓ 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: - simplicity and seamless compatibility with existing systems

### 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.
- ✓ 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 GUI interfaces and some additional application-specific business rules.
- ✓ 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.
- ✓ Thus, the user interface, application rules, and data access act as the three tiers.
- ✓ The **presentation layer** displays information to the user and allows data entry.
- ✓ The **business logic layer** handles intermediate rules and constraints before data is passed up to the user or down to the DBMS.
- ✓ The **bottom layer** includes all data management services. The middle layer can also act as a **Web server**, which retrieves query results from the database server and formats them into dynamic Web pages that are viewed by the Web browser at the client side.
- ✓ If business logic layer is divided into multiple layer, then called as n-tier architecture.



## Classification of Database Management Systems

### Data Model

Used in commercial DBMS [eg: relational data model, object data model].

Many legacy applications still run on database systems based on the **hierarchical** and **network data models**.

### Number of users

- Single-user systems support only one user at a time and are mostly used with PCs
- Multiuser systems, which include the majority of DBMSs, support concurrent multiple users

### Number of sites

- **Centralized DBMS:** the data is stored at a single computer site
- **Distributed DBMS [DDBMS]:** DBMS software distributed over many sites
- **Homogeneous DDBMSs** use the same DBMS software at all the sites
- **Heterogeneous DDBMSs** can use different DBMS software at each site

### Cost

- Open source like MYSQL & Postgre SQL
- 30 day copy versions
- Sold in form of licenses

### Types of access path

- inverted file structures
- general purpose or special purpose
- online transaction processing (OLTP) system

### Other important data models Network model

- represents data as record types and also represents a limited type of 1:N relationship, called a set type

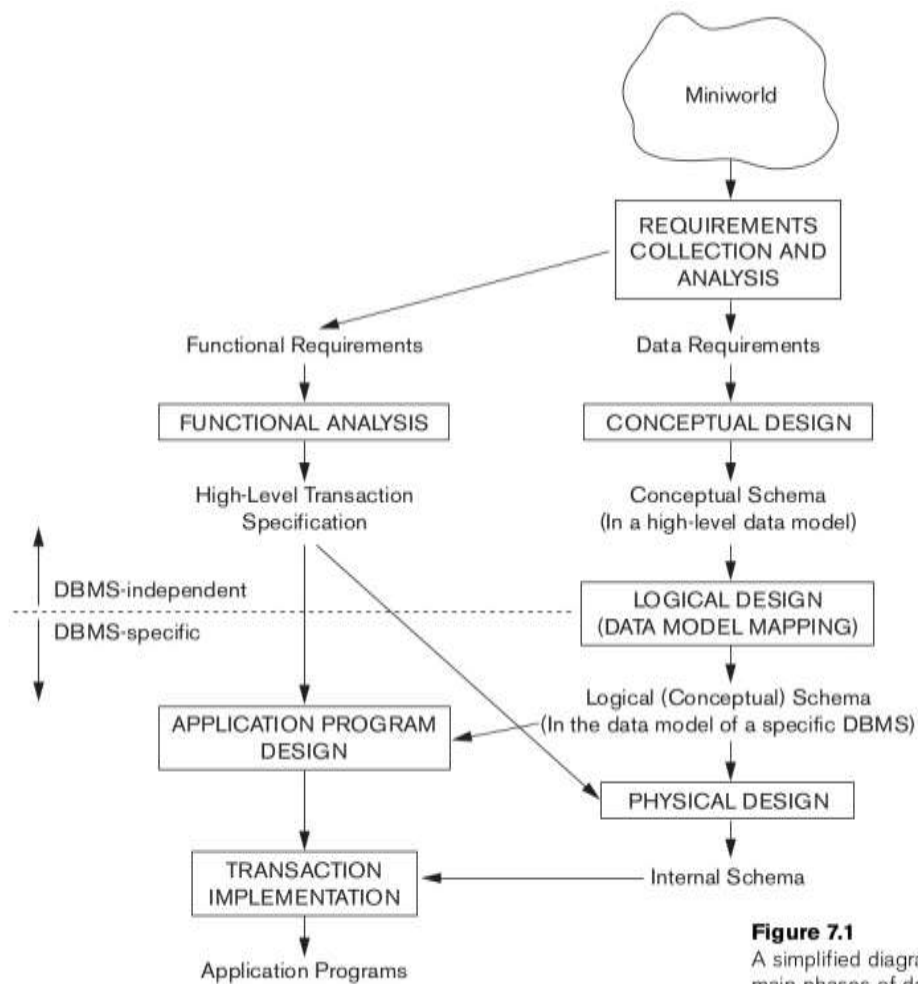
### Hierarchical model

- represents data as hierarchical tree structures
- Each hierarchy represents a number of related records

## CONCEPTUAL DATA MODELLING USING ENTITIES AND RELATIONSHIPS

Conceptual modeling is a very important phase in designing a successful database application

Using High-Level Conceptual Data Models for Database Design



**Figure 7.1**

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

The main phases of database design are:

- ✓ **Requirements Collection and Analysis:** purpose is to produce a description of the users' requirements.
- ✓ **Conceptual Design:** purpose is to produce a conceptual schema for the database, including detailed descriptions of entity types, relationship types, and constraints. All these are expressed in terms provided by the data model being used. Eg: ER model
- ✓ **Implementation:** purpose is to transform the conceptual schema (which is at a high/abstract level) into a (lower-level) representational/implementational model supported by whatever DBMS is to be used.

- ✓ **Physical Design:** purpose is to decide upon the internal storage structures, access paths (indexes), etc., that will be used in realizing the representational model produced in previous phase.

## A Sample Database Application

The COMPANY database keeps track of a company's employees, departments, and projects.

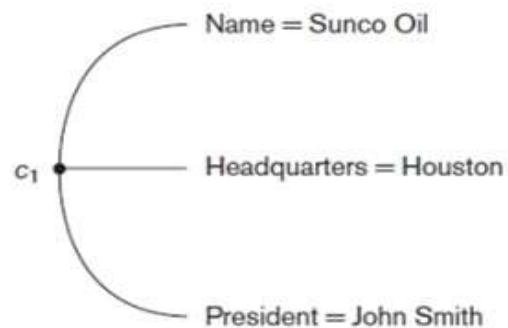
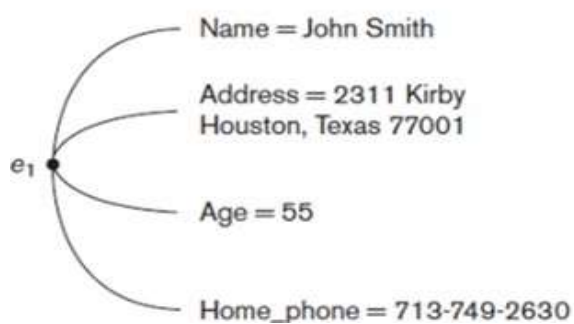
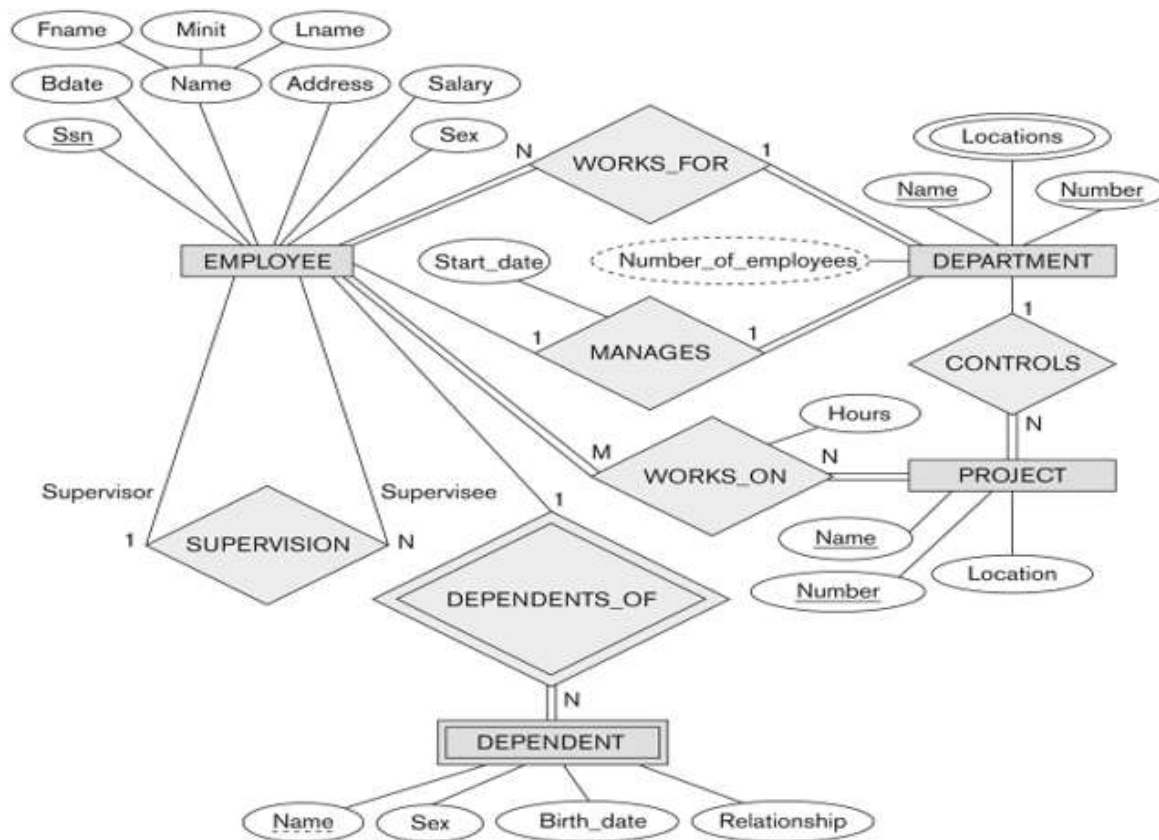
- ✓ The company is organized into departments. Each department has a unique name, a unique number, and a particular employee who manages the department. We keep track of the start date when that employee began managing the department. A department may have several locations.
- ✓ A department controls a number of projects, each of which has a unique name, a unique number, and a single location.
- ✓ The database will 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. It is required to keep track of the current number of hours per week that an employee works on each project, as well as the direct supervisor of each employee (who is another employee).
- ✓ The database will keep track of the dependents of each employee for insurance purposes, including each dependent's first name, sex, birth date, and relationship to the employee.

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

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

### Entities and Attributes

- ✓ **Entity**, which is a **thing or object in the real world** with an independent existence.
- ✓ An entity may be an
  - object with a **physical existence** (for example, a particular person, car, house, or employee) or
  - object with a **conceptual existence** (for instance, a company, a job, or a university course)
- ✓ Each entity has **attributes**—the particular **properties that describe it**. For example, an EMPLOYEE entity may be described by the employee's name, age, address, salary, and job



- ✓ The above fig shows two entities and the values of their attributes.
- ✓ The EMPLOYEE entity e<sub>1</sub> has four attributes: **Name, Address, Age, and Home\_phone**; their values are 'John Smith', '2311 Kirby, Houston, Texas 77001', '55', and '713-749-2630', respectively.
- ✓ The COMPANY entity c<sub>1</sub> has three attributes: **Name, Headquarters, and President**;

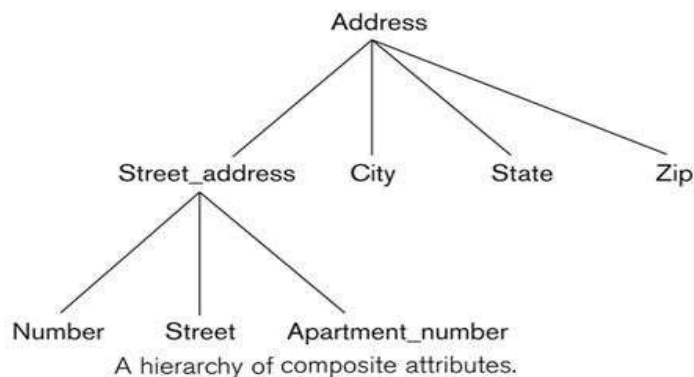
their values are 'Sunco Oil', 'Houston', and 'John Smith', respectively.

Types of attributes occur in the ER model

- ✓ **simple** versus **composite**
- ✓ **single valued** versus **multivalued**
- ✓ **stored** versus **derived**

**Composite attributes** can be divided into smaller subparts.

- ✓ For eg: the Address attribute of the EMPLOYEE entity can be subdivided into Street\_address, City, State, and Zip, 3 with the values '2311 Kirby', 'Houston', 'Texas', and '77001'.
- ✓ Attributes that are not divisible are called **simple or atomic attributes**.
- ✓ For eg: attribute **Age** cannot be divided.



### single-valued

Most attributes have a single value for a particular entity

For eg: **Age** is a single-valued attribute of a person

### Multivalued

An entity having multiple values for that attribute For eg: color of a color color={black,red}

Person's degree degree={BE, MTech, PhD}

### Stored and Derived attribute

- ✓ Two (or more) attribute values are related—for eg: the Age and Birth\_date attributes of a person.
- ✓ 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
- ✓ Birth\_date attribute is called a **stored** attribute

### NULL Values

- ✓ In some cases, a particular entity may not have an applicable value for an attribute.

For eg: the Apartment\_number attribute of an address applies only to addresses that are in apartment buildings and not to other types of residences, such as single-family homes College\_degrees attribute applies only to people with college degrees

### Complex Attributes

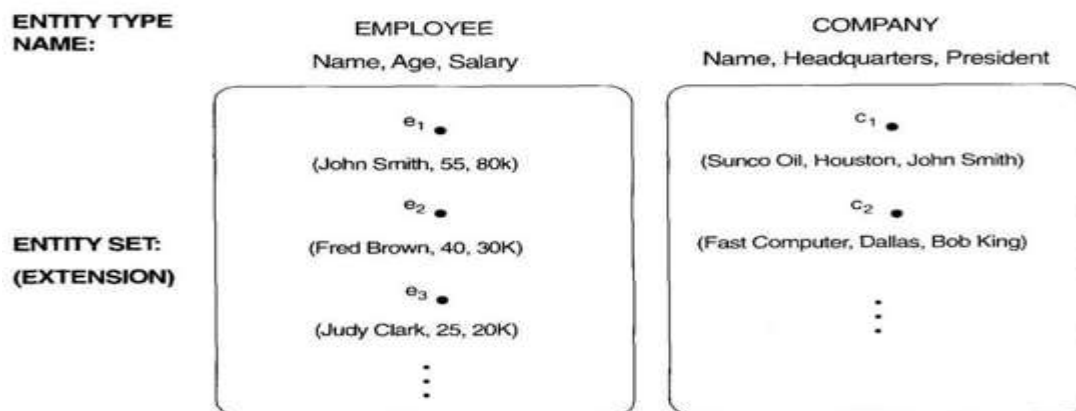
- ✓ composite and multivalued attributes can be nested arbitrarily
- ✓ arbitrary nesting by grouping components of a composite attribute between parentheses ( ) and separating the components with commas, and by displaying multivalued attributes between braces { }. Such attributes are called complex attributes
- ✓ For example, if a person can have more than one residence and each residence can have a single address and multiple phones, an attribute Address\_phone for a person

```
{AddressPhone( {Phone(AreaCode,PhoneNumber)},
Address(StreetAddress(Number,Street,ApartmentNumber),
City,State,Zip) ) }
```

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

#### Entity Types and Entity Sets

- ✓ **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
- ✓ below figure shows two entity types: **EMPLOYEE** and **COMPANY**, and a list of some of the attributes for each



The collection of all entities of a particular entity type in the database at any point in time is called an entity set or entity collection

- ✓ Entity set is usually referred to using the same name as the entity type.
- ✓ An **entity type** is represented in ER diagrams as a **rectangular box**.

- ✓ **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**.
- ✓ collection of entities of a particular entity type is grouped into an entity set, which is also called the extension of the entity type.

### Key Attributes of an Entity Type

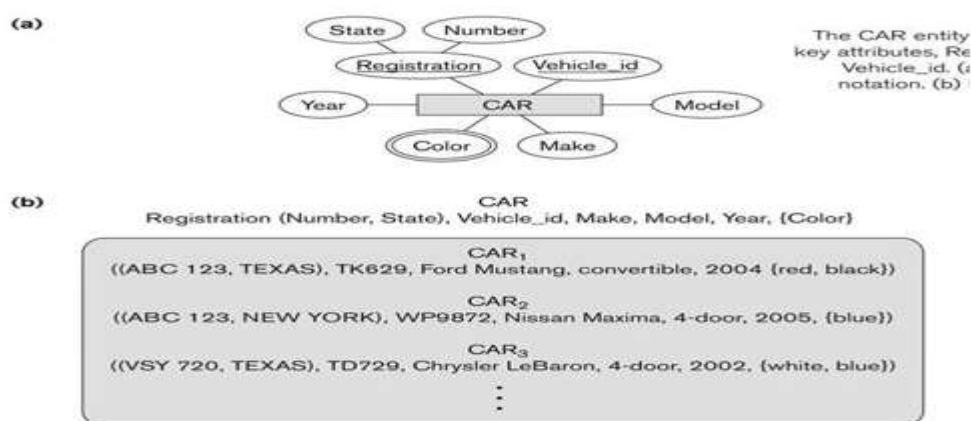
- ✓ important constraint on the entities of an entity type is the **key or uniqueness constraint** on attributes
- ✓ An entity type usually has one or more attributes whose values are distinct for each individual entity in the entity set. Such an attribute is called a **key attribute**, and its values can be used to identify each entity uniquely
- ✓ For eg, the **Name** attribute is a key of the **COMPANY** entity type in because no two companies are allowed to have the same name

For the **PERSON** entity type, a typical key attribute is **Ssn**

- ✓ Specifying that an attribute is a key of an entity type means that the preceding uniqueness property must hold for every entity set of the entity type
- ✓ Hence, it is a constraint that prohibits any two entities from having the same value for the key attribute at the same time

Some entity types have more than one key attribute

For eg, each of the Vehicle\_id and Registration attributes of the entity type CAR is a key in its own right



### Value Sets (Domains) of Attributes

Each simple attribute of an entity type is associated with a **value set** (or domain of values), which specifies the set of values that may be assigned to that attribute for each individual entity

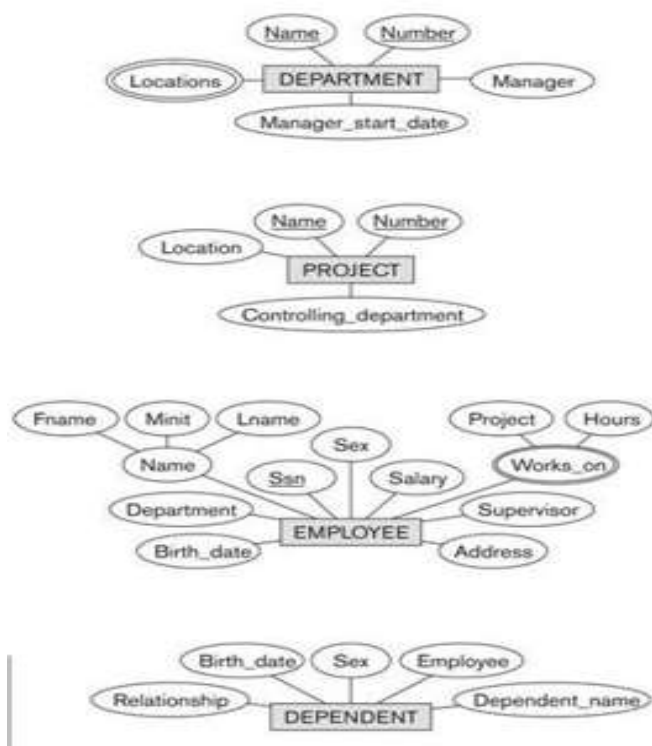
- ✓ if the range of ages allowed for employees is between 16 and 70, we can specify the

value set of the Age attribute of EMPLOYEE to be the set of integer numbers between 16 and 70

- ✓ Value sets are not typically displayed in basic ER diagrams and are similar to the basic data types available in most programming languages, such as integer, string, Boolean, float, enumerated type, subrange, and so on
- ✓ 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 P(V) of V:  $A : E \rightarrow P(V)$
- ✓ We refer to the value of attribute A for entity e as A(e).
- ✓ The previous definition covers both single-valued and multivalued attributes, as well as NULLs.
- ✓ A NULL value is represented by the empty set
- ✓ For single-valued attributes, A(e) is restricted to being a singleton set for each entity e in E
- ✓ no restriction on multivalued attributes

For a composite attribute A, the value set V is the power set of the Cartesian product of  $P(V_1), P(V_2), \dots, P(V_n)$ , where  $V_1, V_2, \dots, V_n$  are the value sets of the simple component attributes that form  $A: V = P(P(V_1) \times P(V_2) \times \dots \times P(V_n))$

### Initial Conceptual Design of the COMPANY Database



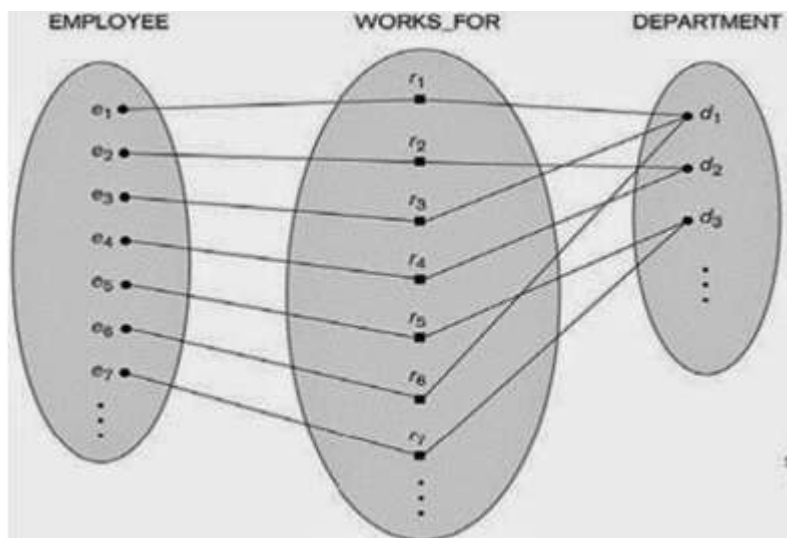


### Relationship Types, Relationship Sets, Roles, and Structural Constraints

- ✓ 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, the attribute Controlling\_department of PROJECT refers to the department that controls the project

- ✓ in the ER model, these references should not be represented as attributes but as **relationships**
- ### Relationship Types, Sets, and Instances

- ✓ A **relationship type**  $R$  among  $n$  entity types  $E_1, E_2, \dots, E_n$  defines a set of associations—or a relationship set—among entities from these entity types.
- ✓ entity types and entity sets, a relationship type and its corresponding relationship set are customarily referred to by the same name,  $R$ 
  - Mathematically, the relationship set  $R$  is a set of relationship instances  $r_i$ , where each  $r_i$  associates  $n$  individual entities  $(e_1, e_2, \dots, e_n)$ , and each entity  $e_j$  in  $r_i$  is a member of entity set  $E_j$ ,  $1 \leq j \leq n$
- ✓ a relationship set is a mathematical relation on  $E_1, E_2, \dots, E_n$ ; alternatively, it can be defined as a subset of the Cartesian product of the entity sets  $E_1 \times E_2 \times \dots \times E_n$
- ✓ each of the entity types  $E_1, E_2, \dots, E_n$  is said to participate in the relationship type  $R$
- ✓ each of the individual entities  $e_1, e_2, \dots, e_n$  is said to participate in the **relationship instance**  $r_i = (e_1, e_2, \dots, e_n)$



- Consider a relationship type WORKS\_FOR between the two entity types EMPLOYEE and DEPARTMENT, which associates each employee with the department for which the employee works. Each relationship instance in the relationship set WORKS\_FOR associates one EMPLOYEE entity and one DEPARTMENT entity.

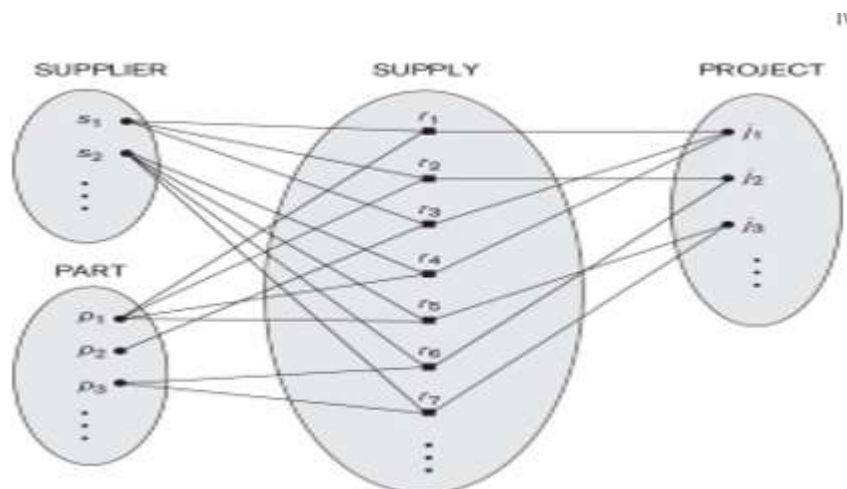
The employees e1, e3, and e6 work for department d1.

The employees e2 and e4 work for department d2; and the employees e5 and e7 work for department d3.

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

### 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
- Hence, the WORKS\_FOR relationship is of degree two.
- A relationship type of degree two is called **binary**, and one of degree three is called **ternary**
- An example of a ternary relationship is SUPPLY, shown in Figure, where each relationship instance  $r_i$  associates three entities—a supplier  $s$ , a part  $p$ , and a project  $j$ —whenever  $s$  supplies part  $p$  to project  $j$ .



### 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.
- ✓ For example, in the WORKS\_FOR relationship type, EMPLOYEE plays the role of employee or worker and DEPARTMENT plays the role of department or employer.
- ✓ Same entity type participates more than once in a relationship type in different roles, such relationship types are called **recursive relationships**.

## 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
- These constraints are determined from the miniworld situation that the relationships represent
- two main types of binary relationship constraints:
  - ✓ cardinality ratio
  - ✓ participation constraint



Structural constraints

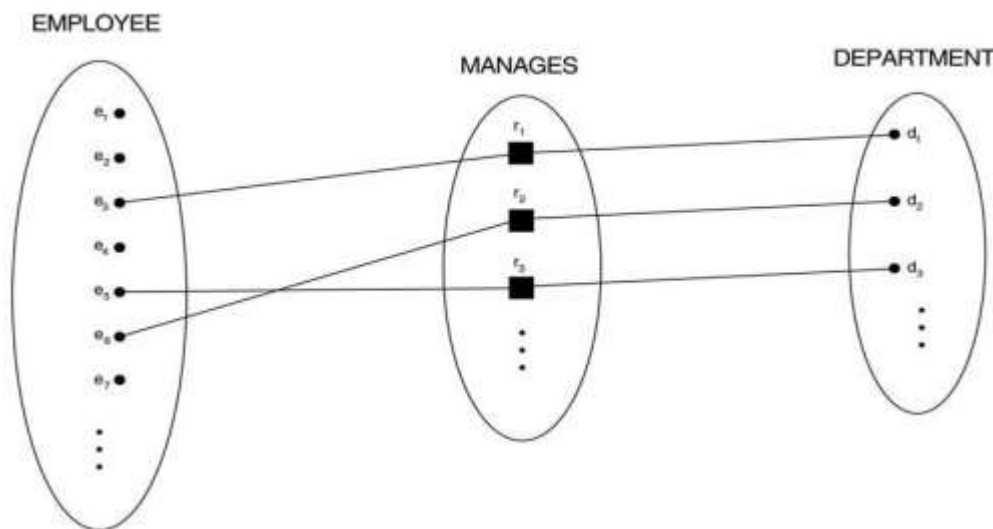
## Cardinality Ratios for Binary Relationships

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 (N), but an employee can be related to (work for) at most one department (1).

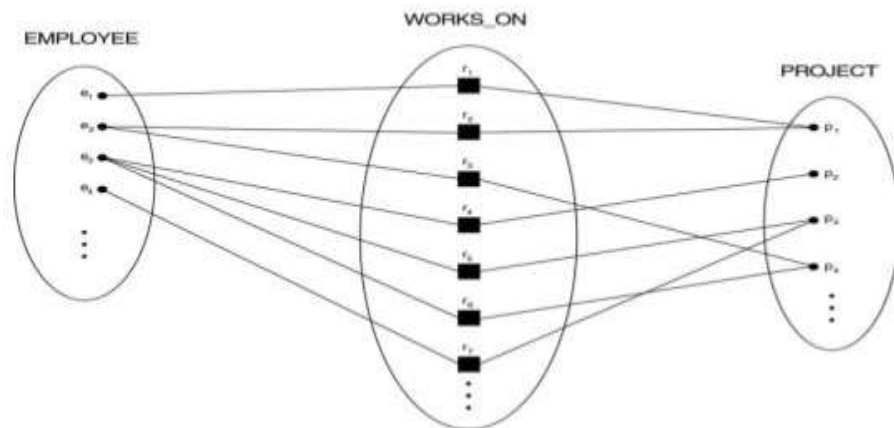
The possible cardinality ratios for binary relationship types are **1:1, 1:N, N:1, and M:N**

### A 1:1 relationship, MANAGES.



In 1:1 an employee can manage at most one department and a department can have at most one manager

### An M:N relationship, WORKS\_ON.



In M:N an employee can work on several projects and a project can have several employees.

Cardinality ratios for binary relationships are represented on ER diagrams by displaying 1, M, and N on the diamonds.

### Participation Constraints and Existence Dependencies

The participation constraint specifies whether the existence of an entity depends on its being related to another entity via the relationship type.

- ✓ This constraint specifies the minimum number of relationship instances that each entity can participate in and is sometimes called the minimum cardinality constraint.
- ✓ There are two types of participation constraints—**total** and **partial**.
- ✓ If a company policy states that every employee must work for a department, then an employee entity can exist only if it participates in at least one WORKS\_FOR relationship instance. Thus, the participation of EMPLOYEE in WORKS\_FOR is called **total participation**, meaning that every entity in the total set of employee entities must be related to a department entity via WORKS\_FOR. Total participation is also called **existence dependency**.
- ✓ We do not expect every employee to manage a department, so the participation of 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.
- ✓ 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**.

### 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 a particular employee works on

a particular project, we can include an attribute **Hours** for the **WORKS\_ON** relationship type

- ✓ to include the date on which a manager started managing a department via an attribute **Start\_date** for the:
  - MANAGES relationship type

### Weak Entity Types

- ✓ Entity types that **do not have key attributes** of their own are called **weak entity types**.
- ✓ In contrast, **regular entity types** that do have a key attribute—are called **strong 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.
- ✓ A weak entity type always has a total participation constraint with respect to its identifying relationship because a weak entity cannot be identified without an owner entity
- ✓ 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.
- ✓ assume that no two dependents of the same employee ever have the same first name, the attribute Name of DEPENDENT is the partial key.

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

#### Proper Naming of Schema Constructs











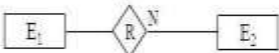

- ✓ choose names that convey the meanings attached to the different constructs in the schema
- ✓ use singular names for entity types, rather than plural ones.
- ✓ Use the convention that entity type and relationship type names are in uppercase letters, attribute names have their initial letter capitalized, and role names are in lowercase letters.
- ✓ nouns appearing in the narrative tend to give rise to entity type names, and the verbs tend to indicate names of relationship types.
- ✓ choosing binary relationship names to make the ER diagram of the schema readable from left to right and from top to bottom.

## Design Choices for ER Conceptual Design

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.
- ✓ Similarly, an attribute that exists in several entity types may be elevated or promoted to an independent entity type. For example, suppose that each of several entity types in a UNIVERSITY database, such as STUDENT, INSTRUCTOR, and COURSE, 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.

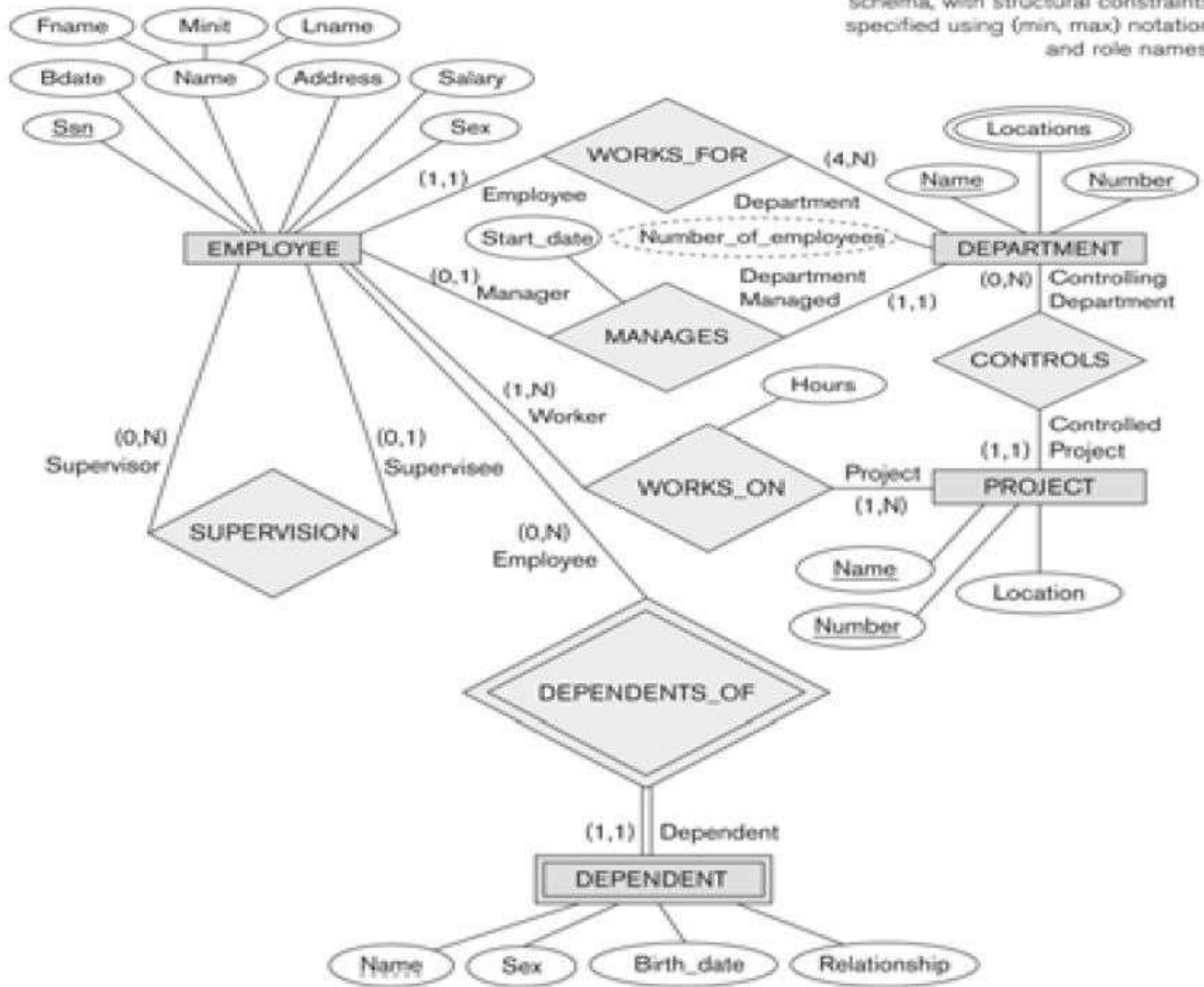
An inverse refinement to the previous case may be applied—for example, if an entity type DEPARTMENT exists in the initial design with a single attribute Dept\_name and is related to only one other entity type, STUDENT. In this case, DEPARTMENT may be reduced or demoted to an attribute of STUDENT.

Symbol	Meaning
	ENTITY TYPE
	WEAK ENTITY TYPE
	RELATIONSHIP TYPE
	IDENTIFYING RELATIONSHIP TYPE
	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 $R$

**ER diagrams for the company schema, with structural constraints specified using (min, max) notation and role names**

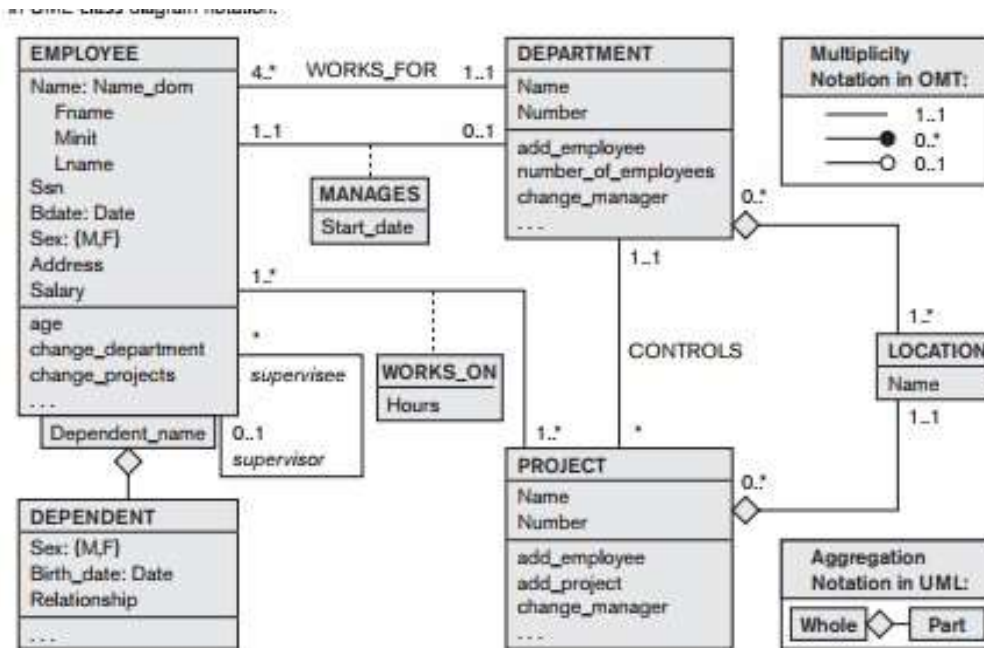
**Figure 3.15**

ER diagrams for the company schema, with structural constraints specified using (min, max) notation and role names.

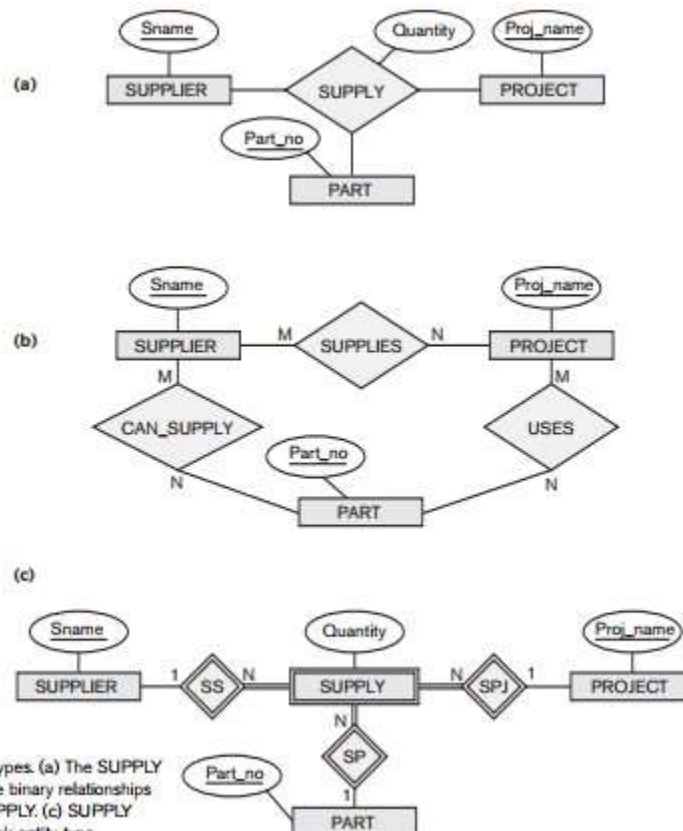




### Example of Other Notation: UML Class Diagrams



### Relationship Types of Degree Higher than Two

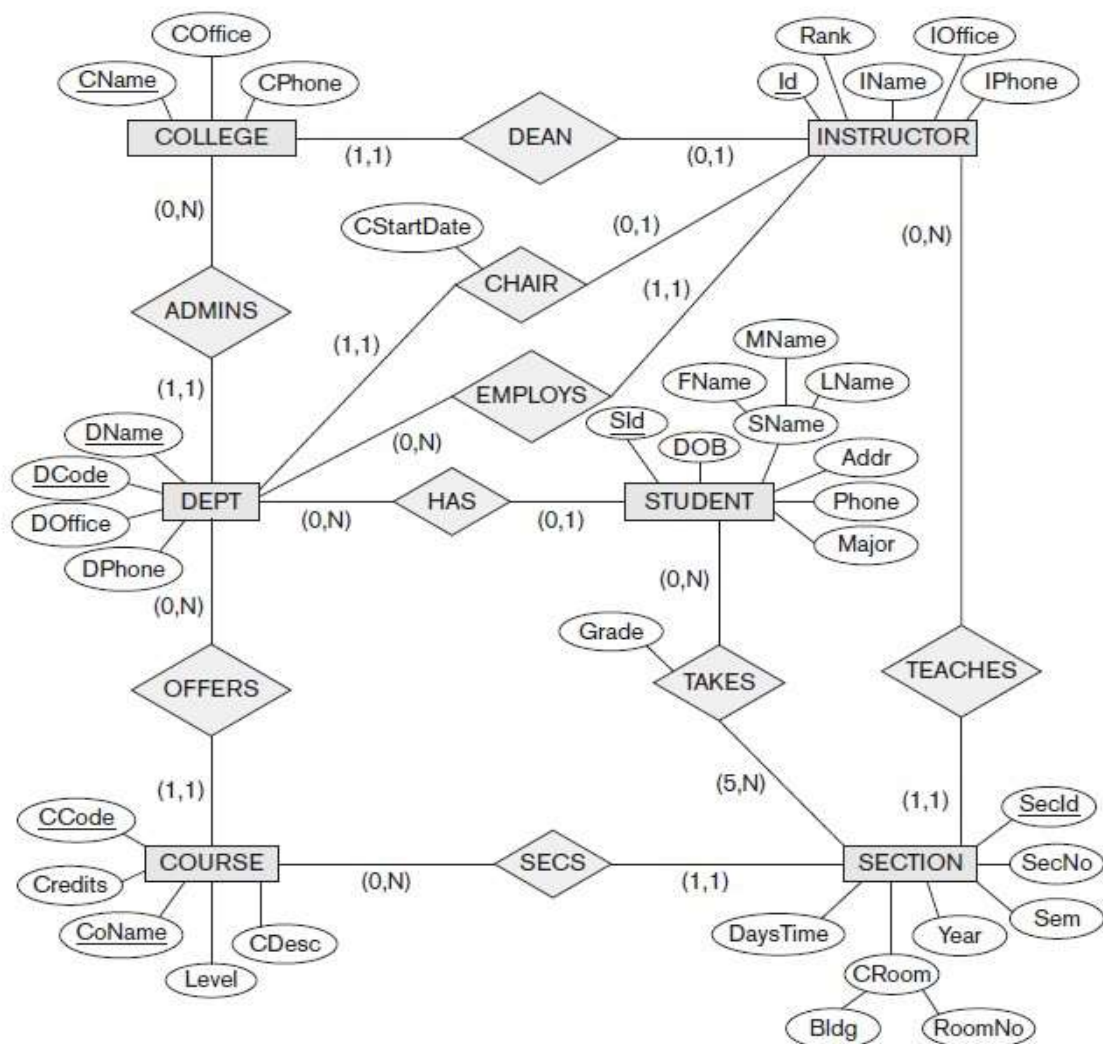


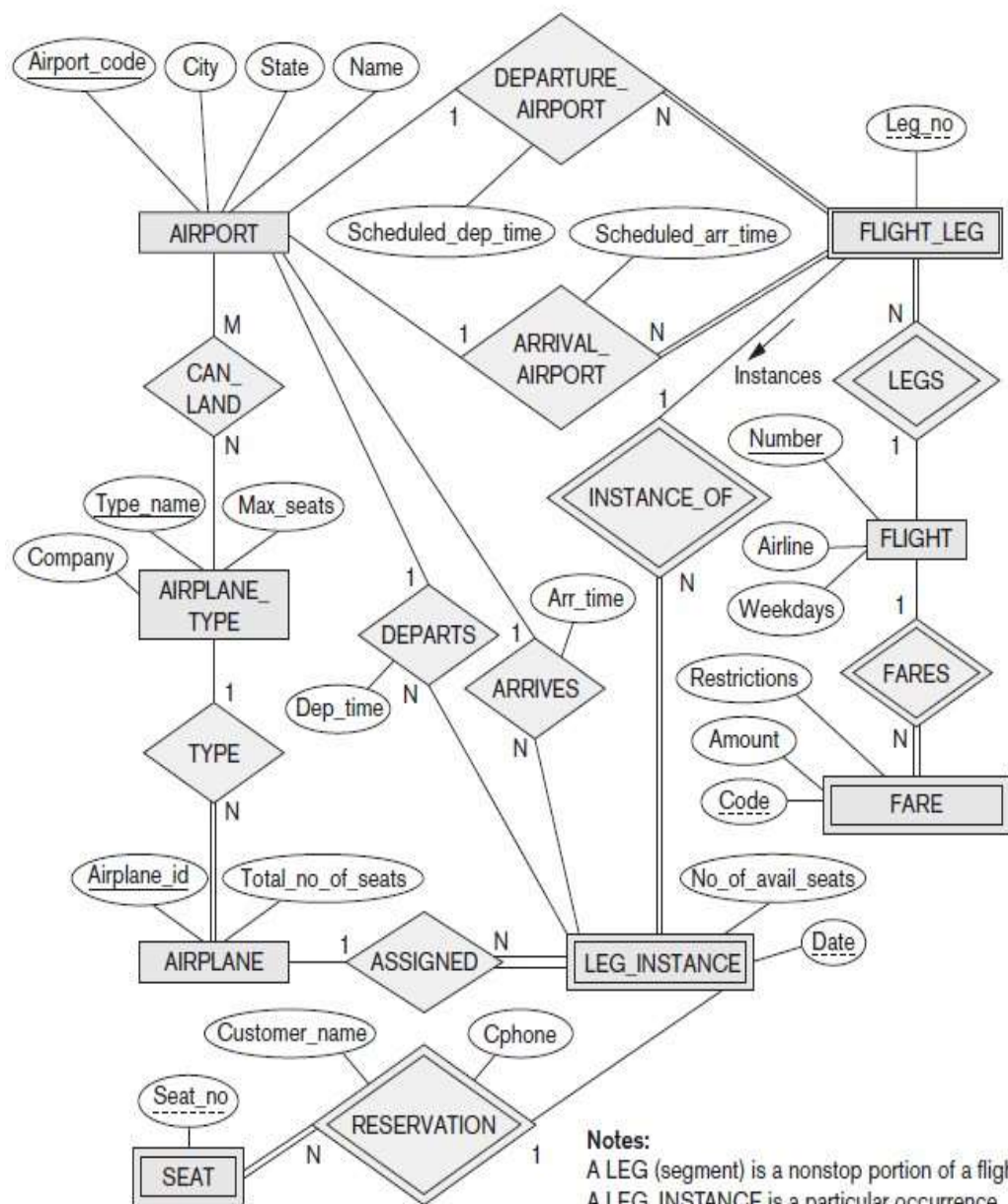
**Figure 7.17**  
Ternary relationship types. (a) The **SUPPLY** relationship. (b) Three binary relationships not equivalent to **SUPPLY**. (c) **SUPPLY** represented as a weak entity type.



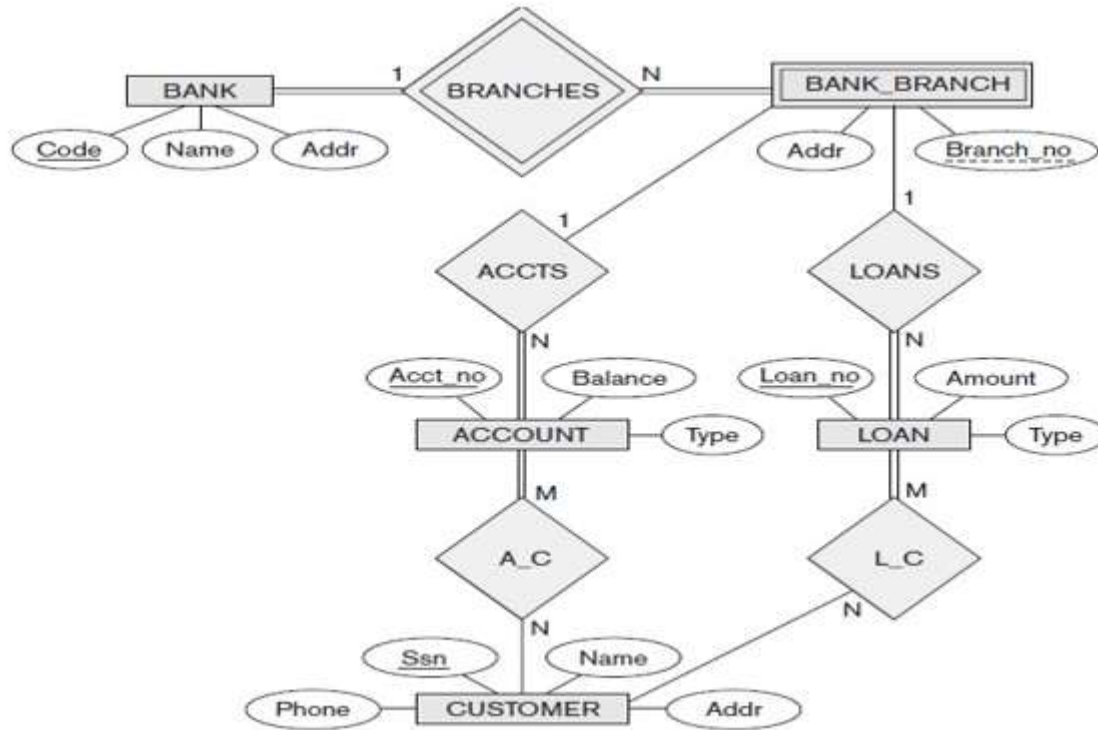
### Choosing between Binary and Ternary (or Higher-Degree) Relationships

- ✓ The ER diagram notation for a ternary relationship type is shown in Figure (a), which displays the schema for the SUPPLY relationship type that was displayed at the entity set/relationship set or instance level.
  - Recall that the relationship set of SUPPLY is a set of relationship instances  $(s, j, p)$ , where  $s$  is a SUPPLIER who is currently supplying a PART  $p$  to a PROJECT  $j$ .
  - In general, a relationship type  $R$  of degree  $n$  will have  $n$  edges in an ER diagram, one connecting  $R$  to each participating entity type.
- ✓ Figure (b) shows an ER diagram for three binary relationship types CAN\_SUPPLY, USES, and SUPPLIES.
- ✓ In general, a ternary relationship type represents different information than do three binary relationship types.
- ✓ Consider the three binary relationship types CAN\_SUPPLY, USES, and SUPPLIES. Suppose that CAN\_SUPPLY, between SUPPLIER and PART, includes an instance  $(s, p)$  whenever supplier  $s$  can supply part  $p$  (to any project); USES, between PROJECT and PART, includes an instance  $(j, p)$  whenever project  $j$  uses part  $p$ ; and SUPPLIES, between SUPPLIER and PROJECT, includes an instance  $(s, j)$  whenever supplier  $s$  supplies some part to project  $j$ . The existence of three relationship instances  $(s, p)$ ,  $(j, p)$ , and  $(s, j)$  in CAN\_SUPPLY, USES, and SUPPLIES, respectively, does not necessarily imply that an instance  $(s, j, p)$  exists in the ternary relationship SUPPLY, because the meaning is different.
- ✓ It is often tricky to decide whether a particular relationship should be represented as a relationship type of degree  $n$  or should be broken down into several relationship types of smaller degrees.
- ✓ The designer must base this decision on the semantics or meaning of the particular situation being represented.

**An ER diagram UNIVERSITY Database:**

**An ER diagram for an AIRLINE DB**

### An ER diagram for BANK DB

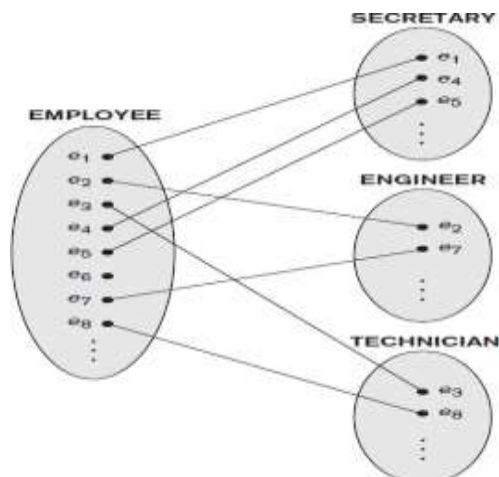


### SPECIALIZATION AND GENERALIZATION

**Specialization** is the process of defining a *set of subclasses* of an entity type; this entity type is called the **superclass** of the specialization.

- ✓ The set of subclasses that forms a specialization is defined on the basis of some distinguishing characteristic of the entities in the superclass.

For example, the set of subclasses {SECRETARY, ENGINEER, TECHNICIAN} is a specialization of the superclass EMPLOYEE that distinguishes among employee entities based on the *job type* of each employee.



## Generalization

- ✓ One can think of a *reverse process* of abstraction in which suppress the differences among several entity types, identify their common features, and **generalize** them into a single **superclass** of which the original entity types are special **subclasses**.
- ✓ For example, consider the entity types CAR and TRUCK shown in below figure. Because they have several common attributes, they can be generalized into the entity type VEHICLE, as shown in Figure.
- ✓ Both CAR and TRUCK are now subclasses of the **generalized superclass** VEHICLE. We use the term.

Generalization to refer to the process of defining a generalized entity type from the given entity types.

Figure 10.1  
Generalization. (a) Two entity types, CAR and TRUCK.  
(b) Generalizing CAR and TRUCK into the superclass VEHICLE.

