

Unit 1: Introduction to Databases

1. Databases and Database Users

2. Database System Concepts and Architecture

Databases and Database Users

Data is known fact that can be recorded and that have implicit meaning.

A **database** is a logically coherent collection of data with some inherent meaning.

A database represents some aspect of the real world, sometimes called the **Miniworld**

A database management system (**DBMS**) is a computerized system that enables users to create and maintain a database.

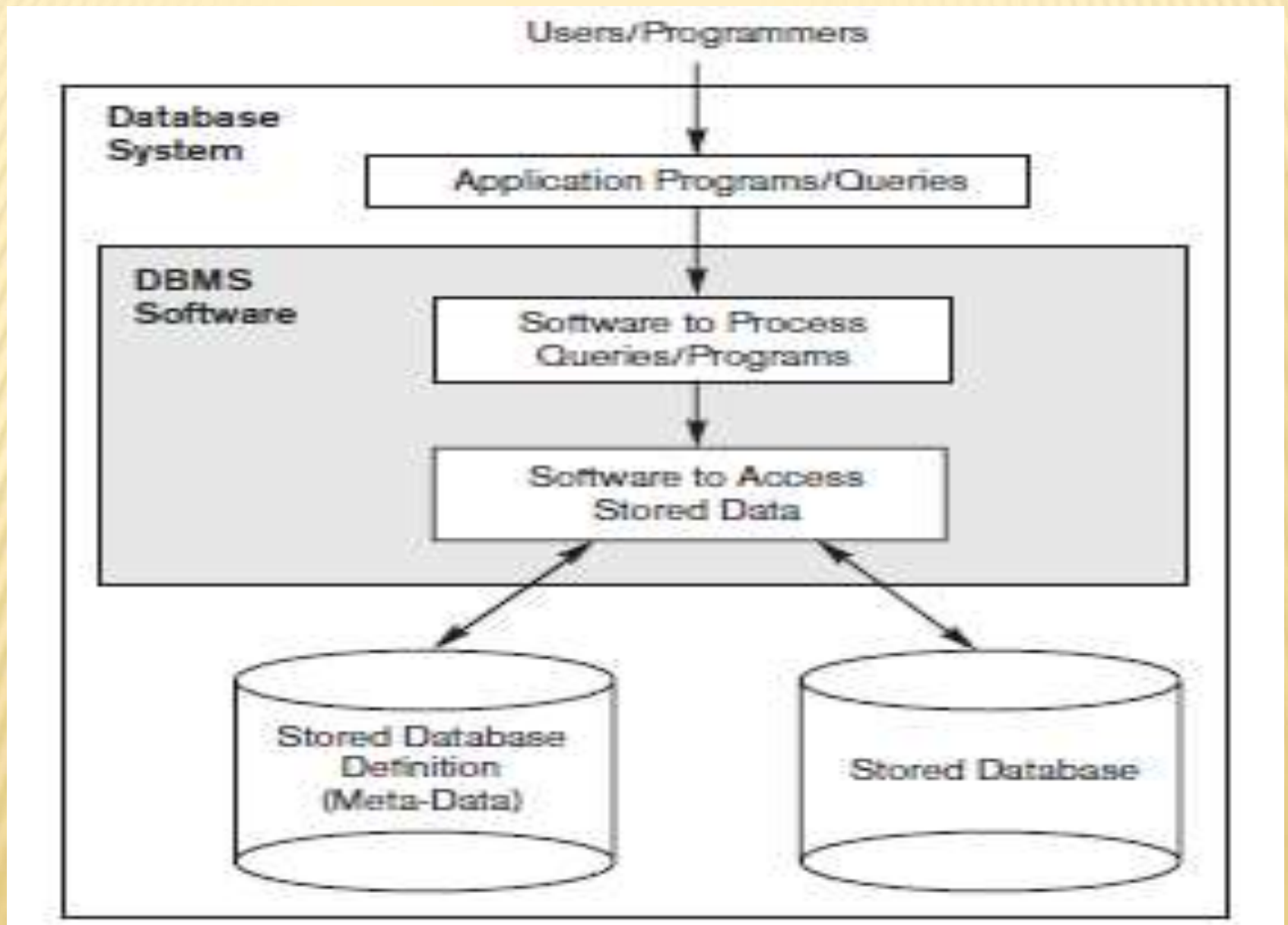
Defining a database involves specifying the data types, structures, and constraints of the data to be stored in the database.

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.

A simplified database system environment.



An Example

STUDENT

Name	Student_number	Class	Major
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

Characteristics of the Database Approach

Self-Describing Nature of a Database System

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**.

Given below is an example for database catalog

RELATIONS

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

COLUMNS

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

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*. 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**.

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.

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. 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. These types of applications are generally called **online transaction processing (OLTP) applications**.

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. These tasks are mostly 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.

End Users

Casual end users occasionally access the database, but they may need different information each time. They use a sophisticated database query interface 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. For example:

Bank customers and tellers check account balances and post withdrawals and deposits.

■ **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. An example is the user of a financial software package that stores a variety of personal financial data.

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**—should be familiar with the full range of capabilities provided by the DBMS to accomplish their tasks.

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. The DBMS must interface with other system software, such as the operating system and compilers for various programming languages.
- **Tool developers design and implement** tools—the software packages that facilitate database modeling and design, database system design, and improved performance. Tools are optional packages that are often purchased separately. They include packages for database design, performance monitoring, natural language or graphical interfaces, prototyping, simulation, and test data generation. In many cases, independent software vendors develop and market these tools.
- **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

Controlling Redundancy: In traditional software development utilizing file processing, every user group maintains its own files for handling its data-processing applications resulting in redundancy.

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.

Providing Persistent Storage for Program Objects: Databases can be used to provide persistent storage for program objects and data structures. This is one of the main reasons for object-oriented database systems.

Providing Storage Structures and Search Techniques for Efficient Query Processing : Database systems must provide capabilities for *efficiently executing queries and updates*. Indexes are used which are typically based on tree data structures or hash data structures that are suitably modified for disk search. The DBMS often has a **buffering or caching module** that maintains parts of the database in main memory buffers for faster retrieval. 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.

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.

Providing Multiple User Interfaces: Because many types of users with varying levels of technical knowledge use a database, a DBMS should provide a variety of user interfaces.

Representing Complex Relationships among Data: 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.

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.

Permitting Inferencing and Actions Using Rules and Triggers: Some database systems provide capabilities for defining *deduction rules for inferencing* new information from the stored database facts. Such systems are called **deductive database systems**.

Additional Implications of Using the Database Approach:

Potential for Enforcing Standards. The database approach permits the DBA to define and enforce standards among database users in a large organization. Potential for Enforcing Standards.

Reduced Application Development Time. Development time using a DBMS is estimated to be one sixth to one-fourth of that for a file system.

Flexibility. It may be necessary to change the structure of a database as requirements change. Modern DBMSs allow certain types of evolutionary changes to the structure of the database without affecting the stored data and the existing application programs.

Availability of Up-to-Date Information. A DBMS makes the database available to all users. As soon as one user's update is applied to the database, all other users can immediately see this update. This availability of up-to-date information is essential for many transaction-processing applications, such as reservation systems or banking databases.

Economies of Scale. The DBMS approach permits consolidation of data and applications, thus reducing the amount of wasteful overlap between activities of data-processing personnel in different projects or departments as well as redundancies among applications& also reducing overall costs of operation and management.

When Not to Use a DBMS

There are a few situations in which a DBMS may involve unnecessary overhead costs that would not be incurred in traditional file processing. 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.

Therefore, it may be more desirable to develop customized database applications 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.

Database System Concepts and Architecture

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.

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

Categories of Data Models

High-level or conceptual data models provide concepts that are close to the way many users perceive data. **Eg: ER Model, Object data model**

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. **Eg: Information is represented by record formats, record orderings, and access paths**

Representational or implementation data models which 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. **Eg: Record-based data models, Hierarchical models, Network Model**

Schemas, Instances, and Database State

Schema : The description of a database is called the **database schema**, which is specified during database design and is not expected to change frequently.

Schema diagram: It is a displayed schema & the diagram displays the structure of each record type but not the actual instances of records.

Eg:

STUDENT

Name	Student_number	Class	Major
------	----------------	-------	-------

COURSE

Course_name	Course_number	Credit_hours	Department
-------------	---------------	--------------	------------

PREREQUISITE

Course_number	Prerequisite_number
---------------	---------------------

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
--------------------	---------------	----------	------	------------

GRADE_REPORT

Student_number	Section_identifier	Grade
----------------	--------------------	-------

Schema construct : Each object in the schema—such as STUDENT or COURSE is a schema construct.

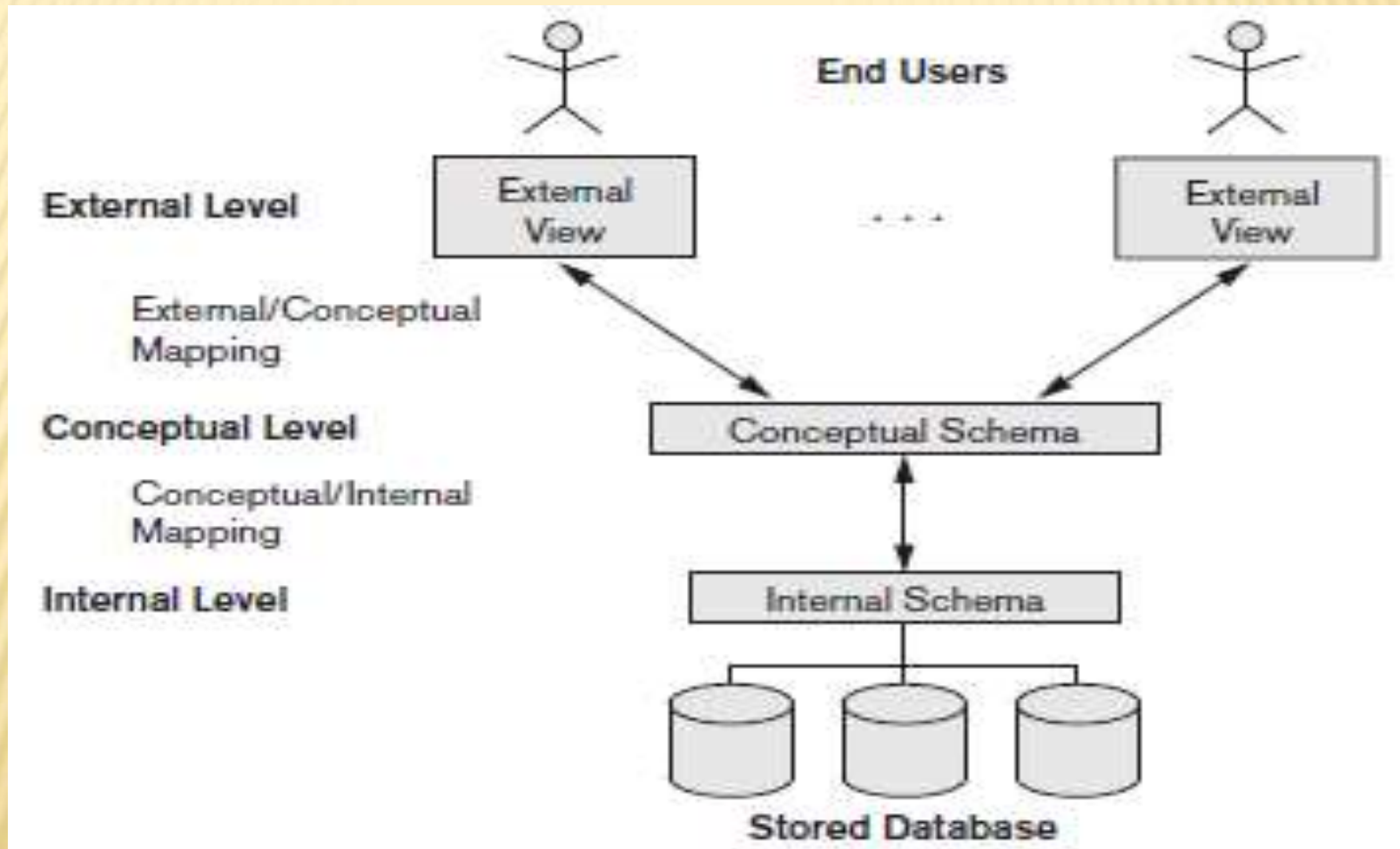
Database state or snapshot.: It is the data in the database at a particular moment in time. (occurrences or instances)

Intension : The schema is sometimes called the **intension**.

Extension : A database state is called an **extension** of the schema.

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.



The Three-Schema Architecture

1. **The internal level has an internal schema**, which describes the physical storage structure of the database. The internal schema 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**, 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.
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.

In the three-schema architecture, each user group refers to its own external schema. Hence, 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**.

Data Independence

Data independence 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:

1. Logical data independence is the capacity to change the conceptual schema without having to change external schemas or application programs.

Eg: By introducing a constraint the views are not changed or affected.

2. Physical data independence is the capacity to change the internal schema without having to change the conceptual schema.

Eg: If there is a change in the path where data is stored, it will not affect the views of users.

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. Web-based user interfaces are used in browsing interfaces.

Apps for Mobile Devices: The apps have built-in programmed interfaces that typically allow users to login using their account name and password. The apps then provide a limited menu of options for mobile access.

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.

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.

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.

Keyword-based Database Search: These are somewhat similar to Web search engines, which accept strings of natural language words and match them with documents at specific sites or Web pages.

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. Applications with limited vocabularies, such as inquiries for telephone directory, flight arrival / departure allowing speech for input and output to enable customers to access this information.

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.

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.

The Database System Environment

DBMS Component Modules

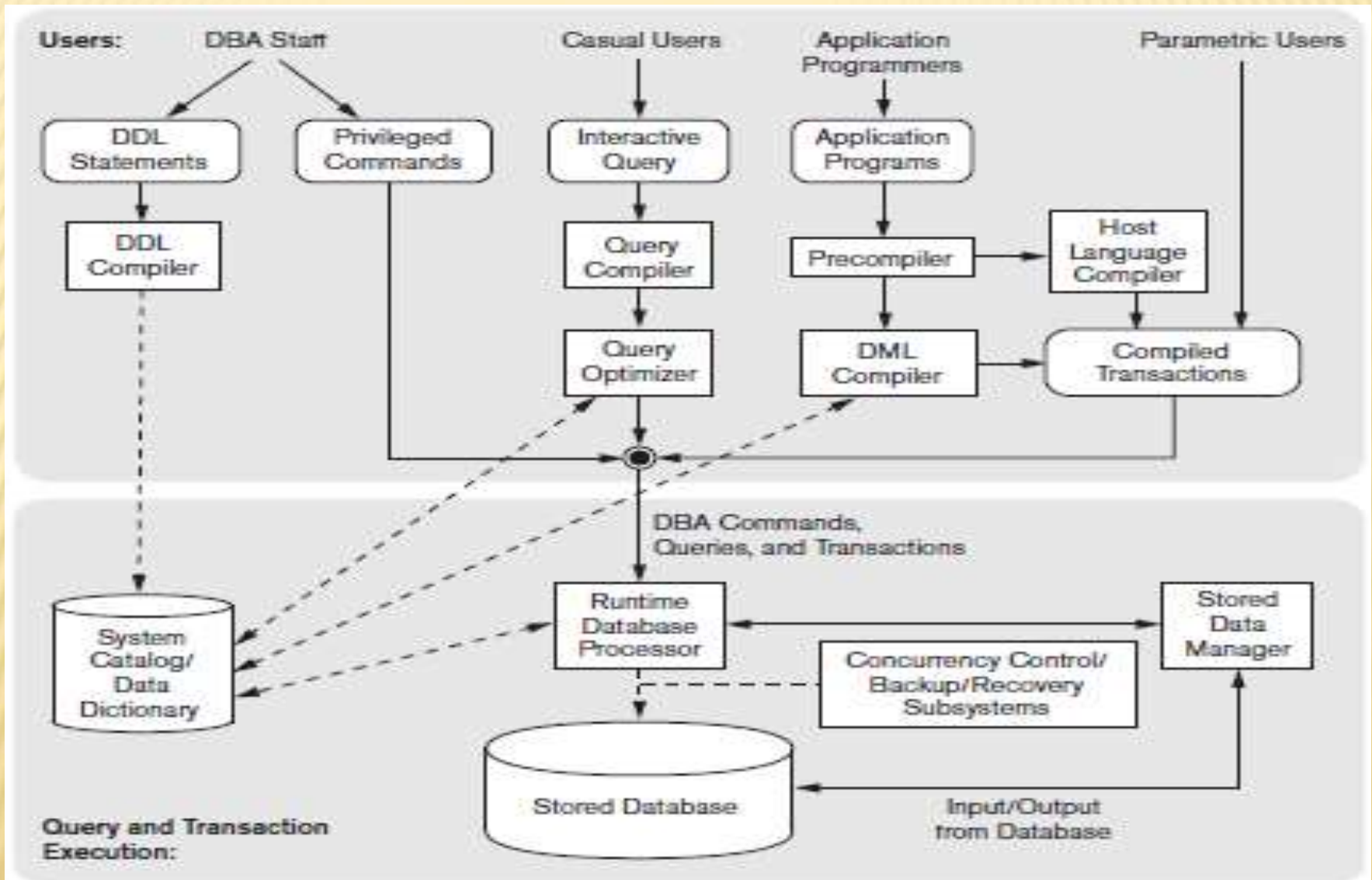


Figure 2.3

Component modules of a DBMS and their interactions.

- ❑ 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.
- ❑ A higher-level **stored data manager** module of the DBMS controls access to DBMS information that is stored on disk, whether it is part of the database or the catalog.
- ❑ Casual users and persons with occasional need for information from the database interact using the **interactive query interface**.
- ❑ The 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.
- ❑ The **query optimizer** is concerned with the rearrangement and possible reordering of operations, elimination of redundancies, and use of efficient search algorithms during execution.

❑ The **pre-compiler** 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.

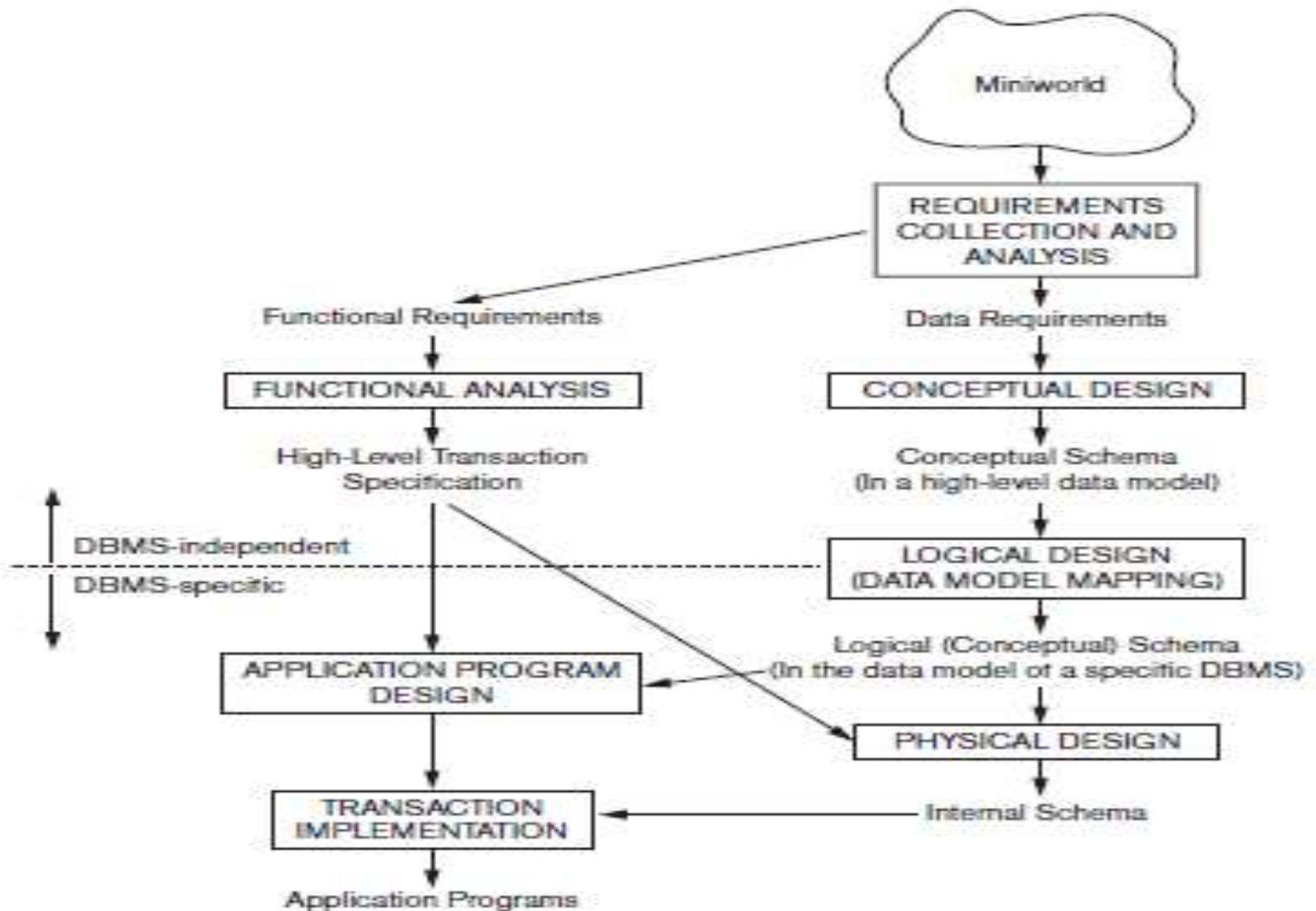
❑ The **runtime database processor** executes

- (1) The privileged commands,
- (2) The executable query plans and
- (3) The canned transactions with runtime parameters.

❑ **Concurrency control** and **backup and recovery systems** are a separate module. They are integrated into the working of the runtime database processor for purposes of transaction management.

Data Modeling Using the Entity-Relationship (ER) Model

USING HIGH-LEVEL CONCEPTUAL DATA MODELS FOR DATABASE DESIGN



USING HIGH-LEVEL CONCEPTUAL DATA MODELS FOR DATABASE DESIGN

- ✖ The 1st step shown is **requirements collection and analysis**. During this step, the database designers interview prospective database users to understand and document their data requirements resulting in concisely written set of users' requirements.
- ✖ Parallely, **functional requirements** of the application are specified using data flow diagrams, sequence diagrams, scenarios, and other techniques.
- ✖ The 2nd step is to create a **conceptual schema** for the database, using a high-level conceptual data model. This step 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; these are expressed using the concepts provided by the high-level data model.

USING HIGH-LEVEL CONCEPTUAL DATA MODELS FOR DATABASE DESIGN

- ✖ The 3rd step in **database design** is the actual implementation of the database, using a commercial DBMS. Most current commercial DBMSs use an implementation data model—such as the relational (SQL) model—so the conceptual schema is transformed from the high-level data model into the implementation data model. This step is called logical design or data model mapping; its result is a database schema in the implementation data model of the DBMS.
- ✖ 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.

ER MODEL CONCEPTS

ER Model describes data in terms of Entities, Attributes & Relationships between the entities.

- ✖ **Entity:** is a thing or object in the real world with an independent existence. An entity may be an object with a physical existence (for eg, a particular person, car, house, or employee) or it may be an object with a conceptual existence (for eg, a company, a job, or a university course).
- ✖ **Attribute:** is a property that describes an entity. Each attribute will have a value or set of values. For eg, an EMPLOYEE entity may have a Name, SSN, Address, Gender BirthDate.

TYPES OF ATTRIBUTES

Simple or atomic attribute

Attributes that cannot be subdivided. Each entity has a single atomic value for the attribute. For example, SSN, CourseID

Composite attribute

Attributes that can be further subdivided into smaller sub-parts representing more basic attributes with independent meaning.

For example, Address (Apt#, House#, Street, City, State, ZipCode, Country) or Name (FirstName, MiddleName, LastName).

Single-Valued attribute

Attributes have a single value for a particular entity.

For example, age, DOB

Multi-valued attribute

An entity may have multiple values for that attribute. For example, Color of a CAR or PreviousDegrees of a STUDENT. Denoted as {Color} or {PreviousDegrees}.

TYPES OF ATTRIBUTES

Derived attribute

If the value of an attribute can be derived from some other attribute then it is known as derived attribute. For example, age is derived from (Current date-DOB)

Stored attribute

If the value of an attribute cannot be derived from any other attribute then it is known as stored attribute. For example, DOB

Complex Attributes

Composite and multi-valued 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 multi-valued attributes between braces { }. Such attributes are called **complex attributes**. For eg, 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 can be specified.

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

TYPES OF ATTRIBUTES

NULL Values

In some cases, a particular entity may not have an applicable value for an attribute. For such attributes, a special value called NULL is assigned. NULL is assigned in 2 cases:

- * A particular attribute is **not applicable** for an entity.

For eg. For UG students, the attribute Degree is not applicable so NULL is assigned.

- * The value of an attribute is **not known**, but it is applicable

For eg. If phone number of student is not known, NULL is assigned.

ENTITY TYPES AND KEY ATTRIBUTES

- ✖ Entities with the same basic attributes are grouped or typed into an **entity type**. For example, the EMPLOYEE entity type or the PROJECT entity type.
- ✖ **Key Attribute (Primary Key):** An entity type having an attribute whose values are distinct for each individual entity in the entity set is called key attribute & its values can be used to identify each entity uniquely. For eg, SSN of EMPLOYEE.
- ✖ **Candidate Key:** A Candidate Key uniquely identifies the rows in a table. There can be more than one candidate keys in a table of which one is chosen as primary key. Also called **Minimal Super Key**. For eg, Car(RegNo, ChassisNo, Color, Make, Year)
- ✖ **Alternate Key:** Any of the candidate keys that is not a part of the primary key is called Alternate key.

ENTITY TYPES AND KEY ATTRIBUTES

- ✖ **Foreign Key:** A foreign key is an attribute which refers to the primary key of another table used to establish & enforce a link between the data in 2 tables.
- ✖ **Super Key:** Any unique key with some non-unique key combination is called super key of the relationship.

For e.g., Emp(SSN, Fname, Lname, DOB, Addr,)

Value Sets (Domains) of Attributes: the set of values that may be assigned to that attribute for each individual entity in an entity set is called value set or domain. For e.g, For employee entity, if age limit is 20-58, then the value set of the attribute age consists of integers from 20 to 58. Age: Domain is [20-58]

It can be represented mathematically as:

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

where A is an attribute

E is an entity set

V is a value set

P(V) is a power set, which includes all subsets of V

We refer to the value of attribute A for entity e as A(e).

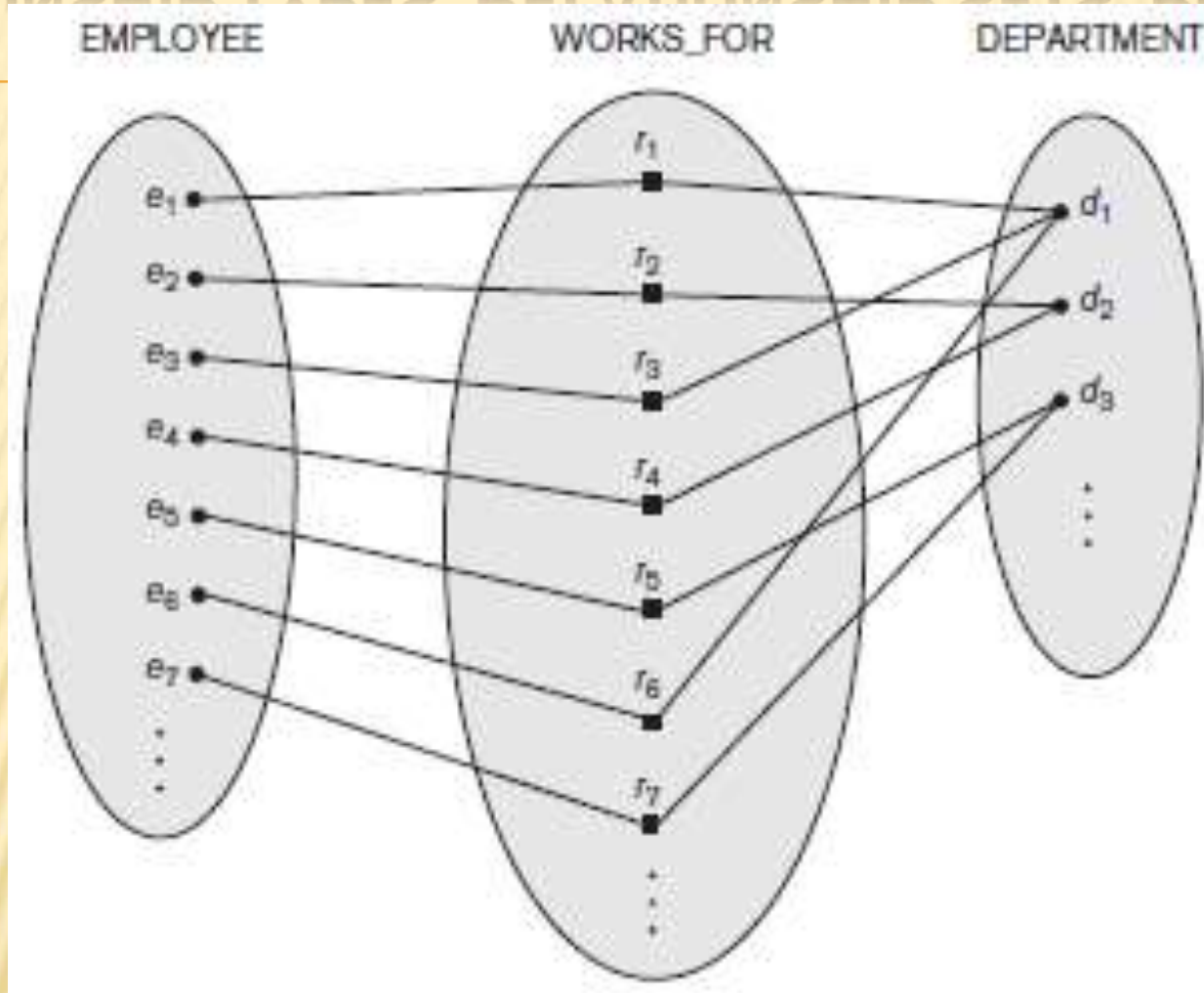
RELATIONSHIP TYPES, RELATIONSHIP SETS, ROLES

Relationship: is an association among 2 or more entities. It can be thought of as verbs linking 2 or more nouns.

Eg: a) Company OWNS Computer
 b) Employee SUPERVISES Dept

✦ **Relationship type, Set & 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. 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 , $1 \leq j \leq n$. Hence, 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 ; similarly, each of the individual entities e_1, e_2, \dots, e_n is said to participate in the relationship instance $r_i = (e_1, e_2, \dots, e_n)$.

RELATIONSHIP TYPES, RELATIONSHIP SETS, ROLES



Instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT

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**, and one of degree three is called **ternary**.

Eg: 1) Binary → **Employee** WORKSFOR **Department**

2) Ternary → **Supplier** SUPPLY **Parts**

Parts SUPPLY **Projects**

Role Names: 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 it helps to explain what the relationship means. 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*.

Recursive Relationships: 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** or **self-referencing relationships**. Eg: The SUPERVISION relationship type relates an employee to a supervisor. 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).

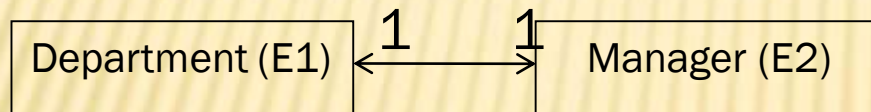
Constraints on Relationship Types: Relationship types usually have certain constraints that limit the possible combinations of entities that may participate in the corresponding relationship set. These constraints are determined from the mini world situation that the relationships represent. Eg: if the company has a rule that each employee must work for exactly one department, then we would like to describe this constraint in the schema.

There are two main types of binary relationship constraints: **cardinality ratio** and **participation**.

Cardinality Ratio: The **cardinality ratio** for a relationship specifies the maximum number of relationship instances that an entity can participate in. The possible cardinality ratios for binary relationship types are 1:1, 1:N, N:1, and M:N.

1:1 Relationship: Between entity sets E1 & E2 indicates that for each entity in either set there is at most 1 entity in the second set that is associated with it.

Eg: A constraint that only 1 manager per department & a manager manages only one department. It can be represented as:



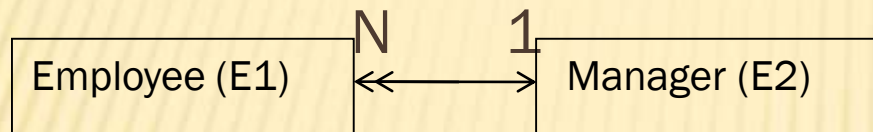
1:N Relationship: From entity set E1 to E2 indicates that for an occurrence of the entity from the set E1, there could be 0, 1 or more entities associated with it. Each entity in E2 is associated with at most 1 entity in the entity set E1.

Eg: Relationship between Manager & Employee



N:1 Relationship: From entity set E1 to E2 indicates that for an occurrence of the entity from the set E2, there could be 0, 1 or more entities associated with it. Each entity in E1 is associated with at most 1 entity in the entity set E2.

Eg: The reverse relationship between Employee & Manager



M:N Relationship: Between entity sets E1 & E2 indicates that there is no restriction on the number of entities in one set associated with an entity in another set. Eg: Relationship between Employee & Project



Participation Constraint: It 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 2 types of participation constraints - **total** and **partial**.

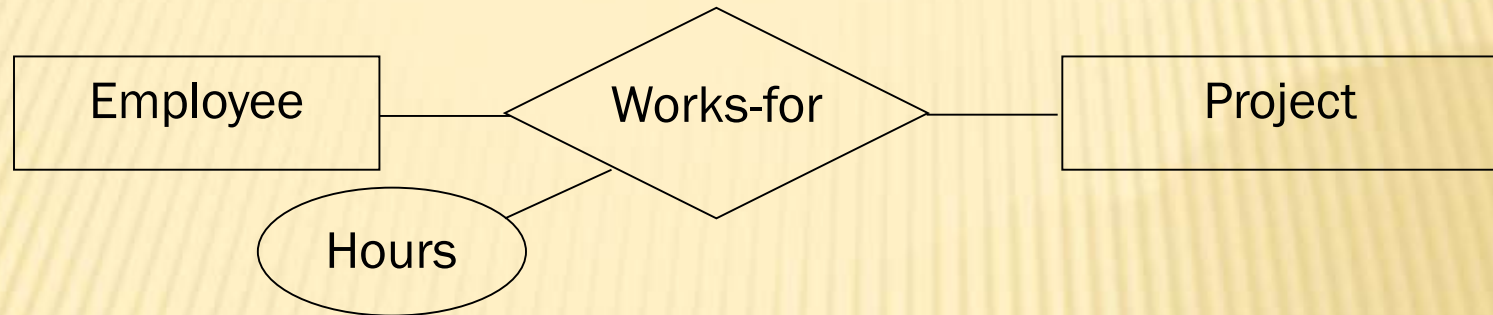
a. Total Participation Constraint (Existence Dependency):

Eg: 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**

b. Partial Participation Constraint:

Eg: Every employee does not manage a department, so the participation of EMPLOYEE in the MANAGES relationship type is **partial**.

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.



Weak Entity Types: 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**. We call the relationship type that relates a weak entity type to its owner the **identifying relationship** of the weak entity type.

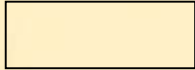
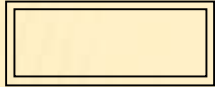

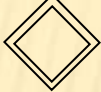






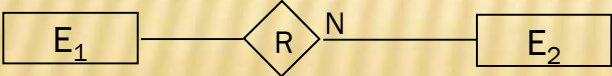

Partial Key: A weak entity type normally has a **partial key**, which is the attribute that can uniquely identify weak entities that are related to the same owner entity. Eg: Dependents (ESSN, Name, Bdate, Relationship)

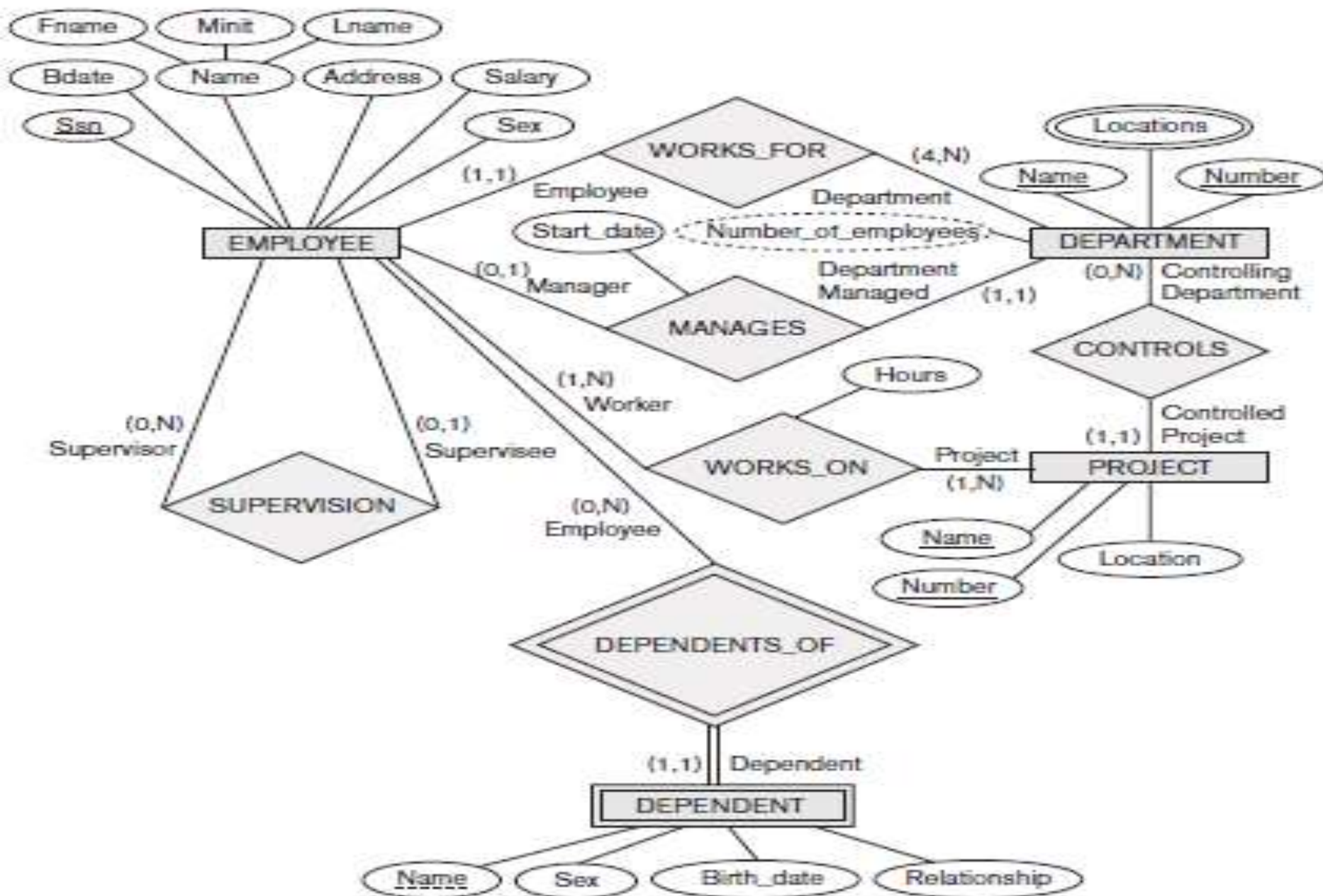
↓
Partial Key

Entity Relationship Diagram

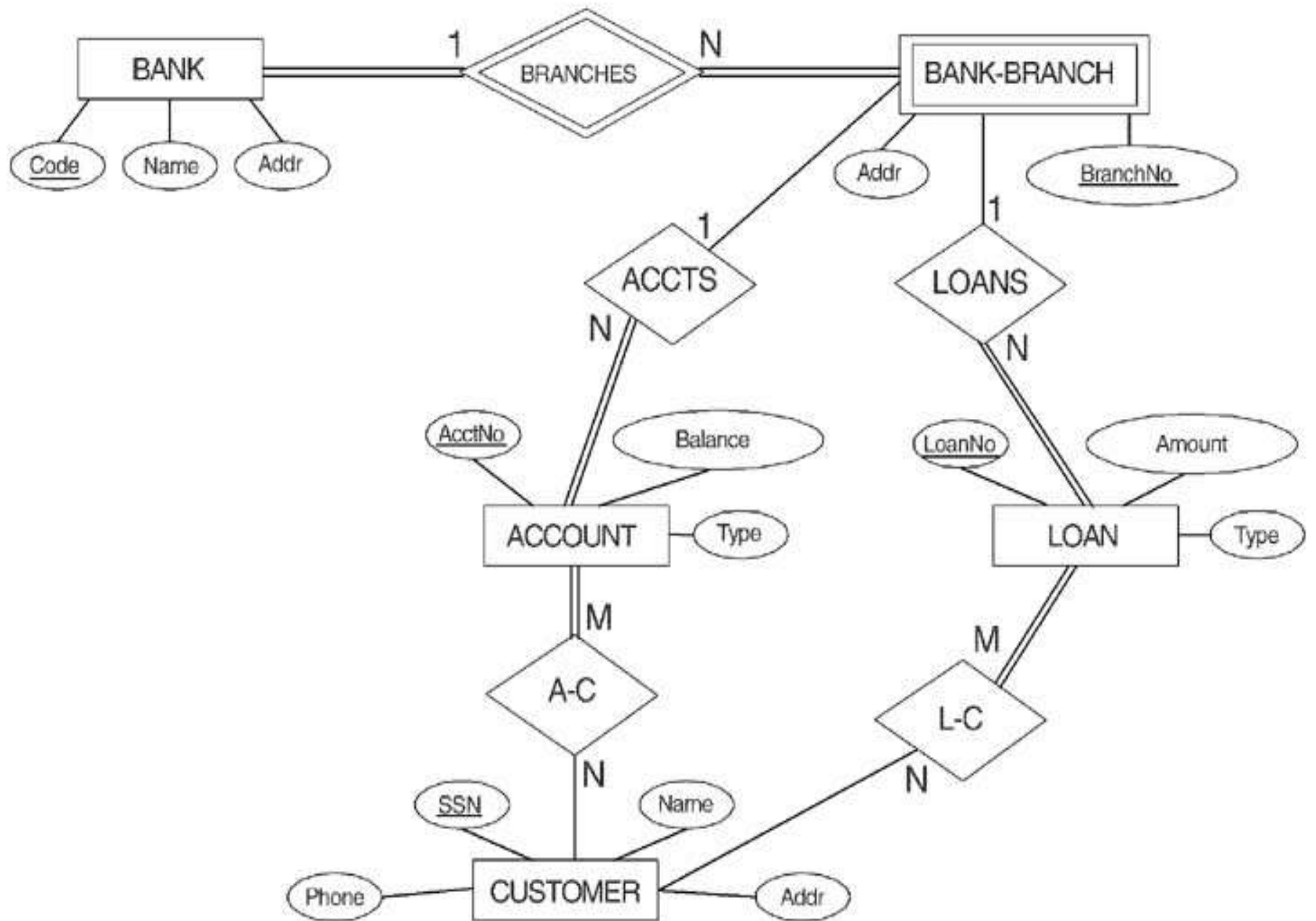
An ER diagram is a **graphical depiction** of organizational system elements & the association among the elements. ER diagrams can help define system boundaries. The elements that make up the system are referred to as entities & a relationship is the association that describes the interaction between entities. ER diagram also indicates the cardinality of a relationship

SUMMARY OF ER-DIAGRAM NOTATIONS

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-DIAGRAM FOR COMPANY DATABASE



ER-DIAGRAM FOR BANK DATABASE

ER-to-Relational Mapping Algorithm

Step 1: Mapping of Regular Entity Types. For each regular (strong) entity type E in the ER schema, create a relation R that includes all the simple attributes of E . Include only the simple component attributes of a composite attribute. Choose one of the key attributes of E as the primary key for R . If the chosen key of E is a composite, then the set of simple attributes that form it will together form the primary key of R .

Step 2: Mapping of Weak Entity Types. For each weak entity type W in the ER schema with owner entity type E , create a relation R and include all simple attributes of W as attributes of R . In addition, include as foreign key attributes of R , the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s); this takes care of mapping the identifying relationship type of W . The primary key of R is the combination of the primary key(s) of the owner(s) and the partial key of the weak entity type W , if any.

Step 3: Mapping of Binary 1:1 Relationship Types. For each binary 1:1 relationship type R in the ER schema, identify the relations S & T that correspond to the entity types participating in R . Choose one of the relations— S , say—and include as a foreign key in S the primary key of T . It is better to choose an entity type with total participation in R in the role of S . Include all the simple attributes of the 1:1 relationship type R as attributes of S .

Step 4: Mapping of Binary 1:N Relationship Types. For each regular binary 1:N relationship type R , identify the relation S that represents the participating entity type at the N-side of the relationship type. Include as foreign key in S the primary key of the relation T that represents the other entity type participating in R . Include any simple attributes of the 1:N relationship type as attributes of S .

Step 5: Mapping of Binary M:N Relationship Types: For each binary M:N relationship type R , create a new relation S to represent R . Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types; their combination will form the primary key of S . Also include any simple attributes of the M:N relationship type as attributes of S .

Step 6: Mapping of Multivalued Attributes. For each multivalued attribute A, create a new relation R. This relation R will include an attribute corresponding to A, plus the primary key attribute K—as a foreign key in R—of the relation that represents the entity type or relationship type that has A as a multivalued attribute. The primary key of R is the combination of A and K. If the multivalued attribute is composite, we include its simple components.

Step 7: Mapping of *N-ary Relationship Types*. For each n-ary relationship type R, where $n > 2$, create a new relationship relation S to represent R. Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types. Also include any simple attributes of the n-ary relationship type as attributes of S.

ER Model	Relational Model
Entity type	Entity relation
1:1 or 1:N relationship type	Foreign key
M:N relationship type	2 Foreign keys
N-ary relationship type	'n' Foreign keys
Simple attributes	Attribute
Composite attributes	Set of simple composite attributes
Multi-valued attributes	Relation & foreign key
Value set	Domain
Key attribute	Primary / Secondary Key