

# **INTRODUCTION**

## **Module – 1**

### **Introduction**

Databases and database technology have a major impact on the growing use of computers. It is fair to say that databases play a critical role in almost all areas where computers are used, including business, electronic commerce, engineering, medicine, genetics, law, education, and library science.

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

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

For example, 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:

- It represents some aspect of the real world, sometimes called the miniworld or the universe of discourse (UoD). Changes to the miniworld are reflected in the database.
- It is a logically coherent collection of data, to which some meaning can be attached.
- It 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.

To summarize: a database has some source (i.e., the miniworld) from which data are derived, some degree of interaction with events in the represented miniworld and an audience that is interested in using it.

**Size/Complexity:** 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. It contains data for over 20 million books, CDs, videos, DVDs, games, electronics, apparel, and other items.

**Computerized vs. manual:** A database may be generated and maintained manually or it may be computerized. For example, simple database like telephone directory may be created and maintained manually. Huge and complex database may be created and maintained either by a group of application programs written specifically for that task or by a database management system.

## Database Management System (DBMS)

A database management system (DBMS) is a collection of programs enabling users to create and maintain a database. More specifically, The DBMS is a general-purpose software system that facilitates the processes of defining, constructing, manipulating, and sharing databases among various users and applications.

- **Defining** a database involves specifying the data types, structures, and constraints of the data to be stored in the database. The database definition or descriptive information is 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.

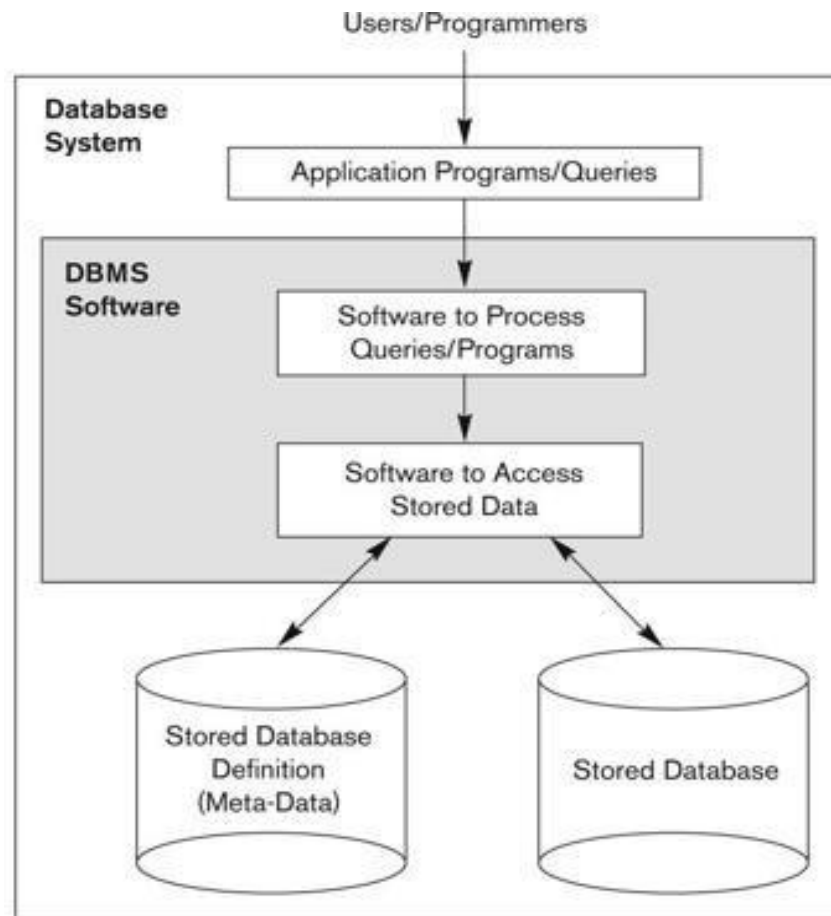
A **query** typically causes some data to be retrieved; a transaction may cause some data to be read and some data to be written into the database.

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.

A database together with the DBMS software is referred to as a **database system**.

## A simplified database system environment



## **An Example**

Consider a UNIVERSITY database for maintaining information concerning students, courses, and grades in a university environment. The database is organized as five files, each of which stores data records of the same type.

1. STUDENT file: stores data on each student.
2. COURSE file: stores data on each course.
3. SECTION file: stores data on each section of a course.
4. GRADE\_REPORT file: stores the grades that students receive in the various sections they have completed.
5. PREREQUISITE file: stores the prerequisites of each course.

**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

### Defining a UNIVERSITY database

- Specify the structure of the records of each file - data elements to be stored in each record. For example: each STUDENT record includes data to represent the student's Name, Student\_number and Class Major. Similarly each COURSE record includes data to represent the Course\_name, Course\_number, Credit\_hours, and Department.
- Specify a data type for each data element within a record. For example: student's Name is a string of alphabetic characters Student\_number is an integer.

### Constructing the UNIVERSITY database

- To construct the UNIVERSITY database, we store data to represent each student, course, section, grade report, and prerequisite as a record in the appropriate file.
- Records in the various files may be related. For example, the record for Smith in the STUDENT file is related to two records in the GRADE\_REPORT file that specify Smith's grades in two sections. Similarly, each record in the PREREQUISITE file relates two course records: one representing the course and the other representing the prerequisite.

### Manipulating a UNIVERSITY database

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

- Retrieve the transcript—a list of all courses and grades—of 'Smith'
- List the names of students who took the section of the 'Database' course offered in fall 2008 and their grades in that section
- List the prerequisites of the 'Database' course

Examples of updates include the following:

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

These informal queries and updates must be specified precisely in the query language of the DBMS before they can be processed.

As with software in general, 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**

### **Database approach vs. File Processing approach**

Consider an organization that is organized as a collection of departments/offices. Each department has certain data processing "needs", many of which are unique to it.

In the file processing approach, each department would control a collection of relevant data files and software applications to manipulate that data. 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.

In the database approach, a single repository maintains data that is defined once and then accessed by various users. In file systems, each application is free to name data elements independently. In contrast, in a database, the names or labels of data are defined once, and used repeatedly by queries, transactions, and applications.

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**

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 meta-data (i.e., data about data) is stored in the so-called system catalog, which contains a description of the structure of each file, the type and storage format of each field, and the various constraints on the data (i.e., conditions that the data must satisfy).

The system catalog is used not only by users but also by the DBMS software, which certainly needs to "know" how the data is structured/organized in order to interpret it in a manner consistent with that structure.

An example of a database catalog for the database is shown below,

**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**

**Program-Data Independence:** In traditional file processing, the structure of the data files accessed by an application is "hard-coded" in its source code. If, for some reason, we decide to change the structure of the data, every application in which a description of that file's structure is hard-coded must be changed!

In contrast, DBMS access programs, in most cases, do not require such changes, because the structure of the data is described separately from the programs that access it and those programs consult the catalog in order to ascertain the structure of the data so that they interpret that data properly.

In other words, the DBMS provides a conceptual or logical view of the data to application programs, so that the underlying implementation may be changed without the programs being modified. (This is referred to as program-data independence.)

**Program-operation independence:** In object-oriented and object-relational systems, users can define operations on data as part of the database definitions. 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**.

## Data abstraction

The characteristic that allows program-data independence and program-operation independence is called **data abstraction**. A DBMS provides users with a conceptual representation of data that does not include many of the details of how the data is stored or how the operations are implemented. Informally, a data model is a type of data abstraction that is used to provide this conceptual representation. The data model uses logical concepts, such as objects, their properties, and their interrelationships, that may be easier for most users to understand than computer storage concepts. Hence, the data model hides storage and implementation details that are not of interest to most database users.

## 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. 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. For example, when several reservation agents try to assign a seat on an airline flight, the DBMS should ensure that each seat can be accessed by only one agent at a time for assignment to a passenger. These types of applications are generally called online transaction processing (OLTP) applications. A fundamental role of multiuser DBMS software is to ensure that concurrent transactions operate correctly and efficiently.

The concept of a transaction has become central to many database applications. 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.

## Database Users

Users may be divided into

- Those who actually use and control the database content, and those who design, develop and maintain database applications called **“Actors on the Scene”**



- Those who design and develop the DBMS software and related tools, and the computer systems operators called “**Workers Behind the Scene**”

### **Actors on the Scene**

**Database Administrator (DBA):** chief administrator, who oversees and manages the database system (including the data and software). Duties include authorizing users to access the database, coordinating/monitoring its use, acquiring hardware/software for upgrades, etc. The DBA is accountable for problems such as security breaches and poor system response time. In large organizations, the DBA might have a support staff.

**Database Designers:** responsible for identifying the data to be stored and for choosing an appropriate way to organize it. 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. The final database design must be capable of supporting the requirements of all user groups.

**End Users:** These are persons who access the database for querying, updating, and report generation. There are several categories of end users:

- **Casual end users:** use database occasionally, needing different information each time; use query language to specify their requests; typically middle- or high-level managers.
- **Naive/Parametric end users:** biggest group of users; frequently query/update the database using standard canned transactions that have been carefully programmed and tested in advance.

Examples:

- Bank tellers check account balances, post withdrawals/deposits
- Reservation clerks for airlines, hotels, etc., check availability of seats/rooms and make reservations.
- **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:** maintain personal databases by using ready-made program packages that provide easy-to-use menu-based or graphics-based interfaces.

Example: user of a tax package that stores a variety of personal financial data for tax purposes.

### **System Analysts and 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**

**DBMS system designers and implementers:** design and implement the DBMS modules and interfaces as a software package. A DBMS is a very complex software system that consists of many components, or modules, including modules for implementing the catalog, query language processing, interface processing, accessing and buffering data, controlling concurrency, and handling data recovery and security.

**Tool developers:** design and implement tools that facilitate database modeling and design, database system design, and improved performance.

**Operators and maintenance personnel (system administration personnel):** 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**

Data redundancy such as tends to occur in the "file processing" approach leads to wasted storage space, duplication of effort and a higher likelihood of the introduction of inconsistency.

In the database approach, the views of different user groups are integrated during database design. This is known as data normalization, and it ensures consistency and saves storage Space. However, it is sometimes necessary to use controlled redundancy to improve the performance of queries. For example, we may store Student\_name and Course\_number redundantly in a GRADE\_REPORT file because whenever we retrieve a GRADE\_REPORT record, we want to retrieve the student name and course number along with the grade, student number, and section identifier.

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

### **Restricting Unauthorized Access**

When multiple users share a large database, it is likely that most users will not be authorized to access all information in the database. For example, financial data is often considered confidential and only authorized persons are allowed to access such data. In addition, some users may only be permitted to retrieve data, whereas others are allowed to retrieve and update. Hence, the type of access operation—retrieval or update—must also be controlled. A DBMS should provide a security and authorization subsystem, which the DBA uses to create accounts, to specify account restrictions and enforce these restrictions automatically.

### **Providing Persistent Storage for Program Objects**

The values of program variables or objects are discarded once a program terminates, unless the programmer explicitly stores them in permanent files, which often involves converting these complex structures into a format suitable for file storage. Object-oriented database systems make it easier for complex runtime objects to be saved in secondary storage so as to survive beyond program termination and to be retrievable at a later time.

Object-oriented database systems are compatible with programming languages such as C++ and Java, and the DBMS software automatically performs any necessary conversions.

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

DBMS maintains indexes that are utilized to improve the execution time of queries and updates. DBMS has a buffering or caching module that maintains parts of the database in main memory buffers. The query processing and optimization module is responsible for choosing an efficient query execution plan for each query submitted to the system.

### **Providing Backup and Recovery**

The backup and recovery subsystem of the DBMS is responsible for recovery. For example, if the computer system fails in the middle of a complex update transaction, the recovery subsystem is responsible for making sure that the database is restored to the state it was in before the transaction started executing. Disk backup is also necessary in case of a catastrophic disk failure.

### **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. These include

- Query languages for casual users
- Programming language interfaces for application programmers
- Forms and command codes for parametric users
- Menu-driven interfaces and natural language interfaces for standalone users.

### **Representing Complex Relationships among Data**

A database may include numerous varieties of data that are interrelated in many ways. For example each section record is related to one course record and to a number of GRADE\_REPORT records—one for each student who completed that section. 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.

### **Enforcing Integrity Constraints**

Most database applications are such that the semantics of the data require that it satisfy certain restrictions in order to make sense. The simplest type of integrity constraint involves specifying a data type for each data item.

For example, in student table we specified that the value of Name must be a string of no more than 30 alphabetic characters.

More complex type of constraint is referential integrity involves specifying that a record in one file must be related to records in other files. For example, in university database, we can specify that every section record must be related to a course record.

Another type of constraint specifies uniqueness on data item values, such as every course record must have a unique value for Course\_number. This is known as a key or uniqueness constraint.

It is the responsibility of the database designers to identify integrity constraints during database design.

### **Permitting Inferencing and Actions Using Rules**

In a deductive database system, one may specify declarative rules that allow the database to infer new data. For example, 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. In today's relational database systems, it is possible to associate triggers with tables.

### **Additional Implications of Using the Database Approach**

**Potential for Enforcing Standards:** database approach permits the DBA to define and enforce standards among database users in a large organization which facilitates communication and cooperation among various departments, projects, and users within the organization. Standards can be defined for names and formats of data elements, display formats, report structures and so on.

**Reduced Application Development Time:** once a database is up and running, substantially less time is generally required to create new applications using DBMS facilities. Development time using a DBMS is estimated to be one-sixth to one-fourth of that for a traditional file system.

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

**Availability of Up-to-Date Information:** DBMS makes the database available to all users. Availability of up-to-date information is essential for many transaction-processing applications, such as reservation systems or banking databases.

**Economies of Scale:** DBMS approach permits consolidation of data and applications, to overlap between activities of data-processing in different projects or departments. This enables the whole organization to invest in more powerful processors, storage devices, or communication gear, rather than having each department purchase its equipment thus reducing overall costs of operation and management.

## **History of Database Applications**

### **Early Database Applications Using Hierarchical and Network Systems**

Early database applications maintained records in large organizations such as corporations, universities, hospitals, and banks. In many of these applications, there were large numbers of records of similar structure. There were also many types of records and many interrelationships among them.

Problems with the early database systems

- Lack of data abstraction and program-data independence capabilities
- Provided only programming language interfaces. This made it time-consuming and expensive to implement new queries and transactions, since new programs had to be written, tested, and debugged.

### **Providing Data Abstraction and Application Flexibility with Relational Databases**

Relational databases were originally proposed to separate the physical storage of data from its conceptual representation and to provide a mathematical foundation for data representation and querying. The relational data model also introduced high-level query languages that provided an alternative to programming language interfaces, making it much faster to write new queries. Hence, data abstraction and program-data independence were much improved when compared to earlier systems.

### **Object-Oriented Applications and the Need for More Complex Databases**

Object-oriented databases (OODBs) mainly used in specialized applications, such as engineering design, multimedia publishing, and manufacturing systems. In addition, many object-oriented concepts were incorporated into the newer versions of relational DBMSs, leading to object-relational database management systems, known as ORDBMSs.

### **Interchanging Data on the Web for E-Commerce Using XML**

The World Wide Web provides a large network of interconnected computers. Users can create documents using a Web publishing language, such as HyperText Markup Language (HTML), and store these documents on Web servers where other users (clients) can access them. Documents can be linked through hyperlinks, which are pointers to other documents.

Currently, eXtended Markup Language (XML) is considered to be the primary standard for interchanging data among various types of databases and Web pages. XML combines concepts from the models used in document systems with database modeling concepts.

### **Extending Database Capabilities for New Applications**

The success of database systems in traditional applications encouraged developers of other types of applications to attempt to use them. The following are some examples of these applications:

- Scientific applications that store large amounts of data resulting from Scientific experiments in areas such as high-energy physics, the mapping of the human genome, and the discovery of protein structures.
- Storage and retrieval of images, including scanned news or personal photographs, satellite photographic images, and images from medical procedures such as x-rays and MRIs (magnetic resonance imaging).
- Storage and retrieval of videos, such as movies, and video clips from news or personal digital cameras.
- Data mining applications that analyze large amounts of data searching for the occurrences of specific patterns or relationships, and for identifying unusual patterns in areas such as credit card usage.
- Spatial applications that store spatial locations of data, such as weather information, maps used in geographical information systems, and in automobile navigational systems.
- Time series applications that store information such as economic data at regular points in time, such as daily sales and monthly gross national product figures.

### **Databases versus Information Retrieval**

Database technology is heavily used in manufacturing, retail, banking, insurance, finance, and health care industries, where structured data is collected through forms, such as invoices or patient registration documents. An area related to database technology is Information Retrieval (IR), which deals with books, manuscripts, and various forms of library-based articles. Data is indexed, cataloged, and annotated using keywords. IR is concerned with searching for material based on these keywords, and with the many problems dealing with document processing and free-form text processing.

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

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 use regular files under the following circumstances:

- Simple, well-defined database applications that are not expected to change at all
- Stringent, real-time requirements for some application programs that may not be met because of DBMS overhead
- Embedded systems with limited storage capacity, where a general-purpose DBMS would not fit
- No multiple-user access to data.

## **Overview of Database Languages and Architectures**

### **Introduction**

The architecture of DBMS packages has evolved from the early monolithic systems, where the whole DBMS software package was one tightly integrated system. Modern DBMS packages are modular in design, with a client/server system architecture. In a basic client/server DBMS architecture, the system functionality is distributed between two types of module. A client module is designed to run on a user workstation or personal computer. The client module handles user interaction and provides the user-friendly interfaces such as forms- or menu-based GUIs. The other kind of module, called a server module handles data storage, access, search, and other functions.

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

#### **Data Model**

A data model is a collection of concepts that can be used to describe the structure of a database. By structure of a database we mean 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. Data model provides the necessary means to achieve abstraction.

#### **Categories of Data Models**

Data models can be categorized according to the types of concepts they use to describe the database structure.

**High-level or conceptual data models:** provide concepts that are close to the way many users perceive data. Conceptual data models use concepts such as entities, attributes, and relationships.

**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. Representational data models hide many details of data storage on disk but can be implemented on a computer system directly. Representational or implementation data models are the models used most frequently in traditional commercial DBMSs. These include the widely used relational data model, as well as the so-called legacy data models—the network and hierarchical models. Representational data models represent data by using record structures and hence are sometimes called record-based data models.

**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. Physical data models describe how data is stored as files in the computer by representing information such as record formats, record orderings, and access paths.

## Database 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

A displayed schema is called a schema diagram. 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.

Example: Schema diagram for the University database is shown below.

### 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 is called schema construct. For example student or course.

## Database state or snapshot

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.

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.

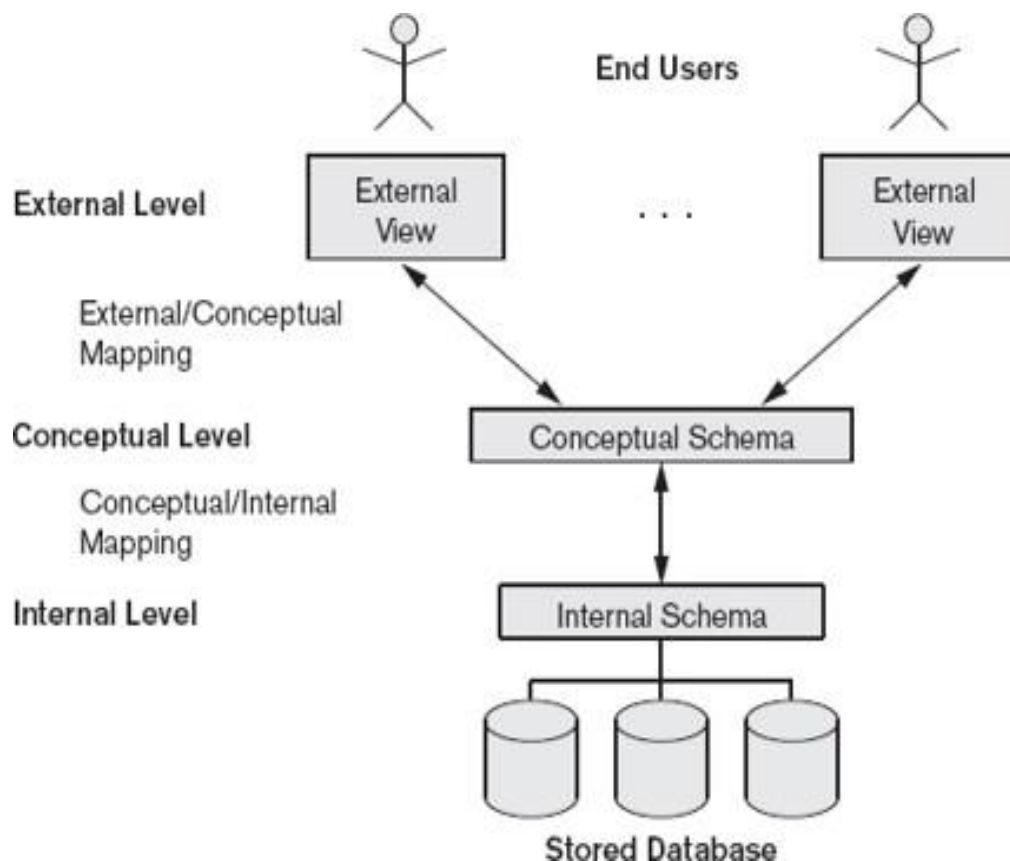
The DBMS stores the descriptions of the schema constructs and constraints—also called the **meta-data**—in the DBMS catalog.

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

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:

**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.

**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.

**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 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. We may change the conceptual schema to expand the database, to change constraints, or to reduce the database. Only the view definition and the mappings need to be changed in a DBMS that supports logical data independence.
2. **Physical data independence** is the capacity to change the internal schema without having to change the conceptual schema. Hence, the external schemas need not be changed as well. Changes to the internal schema may be needed because some physical files were reorganized—for example, by creating additional access structures—to improve the performance of retrieval or update.

Data independence occurs because when the schema is changed at some level, the schema at the next higher level remains unchanged; only the mapping between the two levels is changed.

## **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)**

The data definition language (DDL) is used by the DBA and by database designers to define both schemas when no strict separation of levels is maintained. The DBMS will have a DDL compiler whose function is to process DDL statements in order to identify descriptions of the schema constructs and to store the schema description in the DBMS catalog.

#### **Storage Definition Language (SDL)**

Storage definition language is used when clear separation is maintained between the conceptual and internal levels, the DDL is used to specify the conceptual schema only. The storage definition language (SDL), is used to specify the internal schema. The mappings between the two schemas may be specified in either one of these languages.

#### **View Definition Language (VDL)**

View definition language is used to specify user views and their mappings to the conceptual schema. In relational DBMSs, SQL is used in the role of VDL to define user or application views as results of predefined queries.

#### **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**

User-friendly interfaces provided by a DBMS may include the following,

**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. There is no need for the user to memorize the specific commands and syntax of a query language. Pull-down menus are a very popular technique in Web- based user interfaces.

**Forms-Based Interfaces:** A forms-based interface displays a form to each user. Users can fill out all of the form entries to insert new data, or they can fill out only certain entries, in which case the DBMS will retrieve matching data for the remaining entries. Forms are usually designed and programmed for naive users as interfaces to canned transactions.

**Graphical User Interfaces:** A GUI typically displays a schema to the user in diagrammatic form. The user then can specify a query by manipulating the diagram. In many cases, GUIs utilize both menus and forms. Most GUIs use a pointing device, such as a mouse, to select certain parts of the displayed schema diagram.

**Natural Language Interfaces:** These interfaces accept requests written in English or some other language and attempt to understand them. A natural language interface usually has its own schema, which is similar to the database conceptual schema, as well as a dictionary of important words. The natural language interface refers to the words in its schema, as well as to the set of standard words in its dictionary, to interpret the request.

If the interpretation is successful, the interface generates a high-level query corresponding to the natural language request and submits it to the DBMS for processing; otherwise, a dialogue is started with the user to clarify the request.

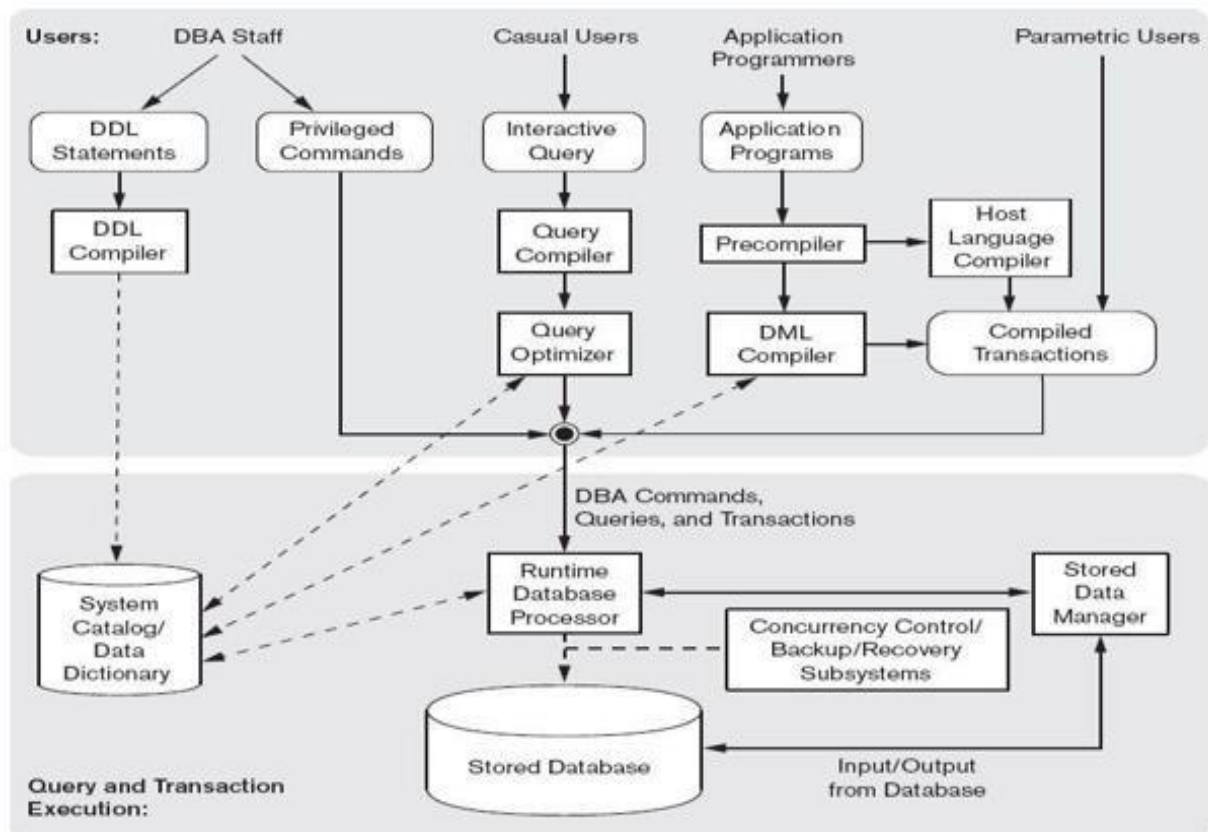
**Speech Input and Output:** Applications with limited vocabularies such as inquiries for telephone directory, flight arrival/departure, and credit card account information are allowing speech for input and output to enable customers to access this information. The speech input is detected using a library of predefined words and used to set up the parameters that are supplied to the queries. For output, a similar conversion from text or numbers into speech takes place.

**Interfaces for Parametric Users:** Parametric users, such as bank tellers, often have a small set of operations that they must perform repeatedly. For example, a teller is able to use single function keys to invoke routine and repetitive transactions such as account deposits or withdrawals, or balance inquiries. Usually a small set of abbreviated commands is included, with the goal of minimizing the number of keystrokes required for each request.

**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



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:

1. 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)
2. 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

### 3. Database storage reorganization

- Used to reorganize a set of database files into different file organizations to improve performance

### 4. 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 environment** 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**.

## **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 mode**

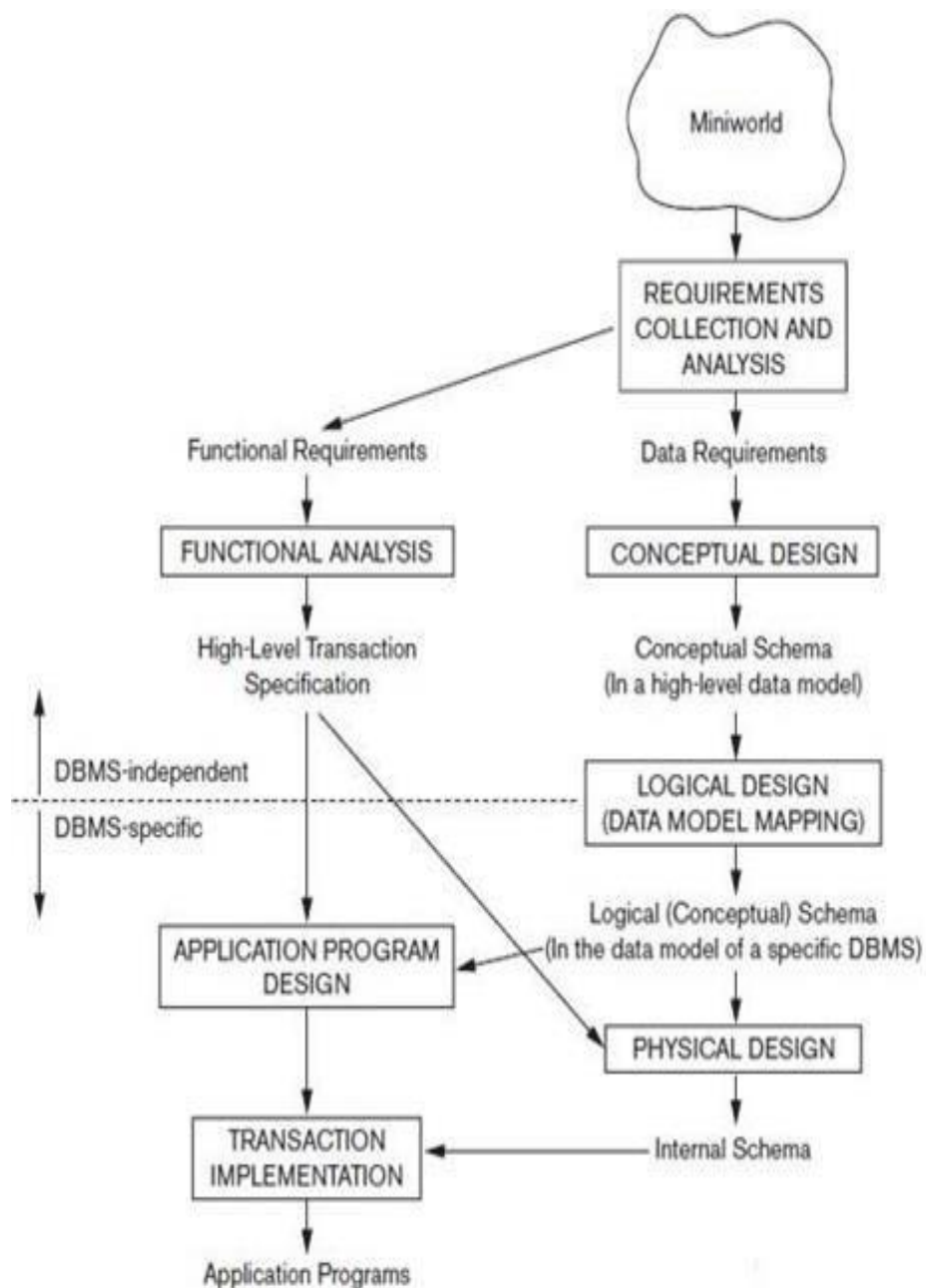
- Represents data as hierarchical tree structures.
- Each hierarchy represents a number of related records.

## Conceptual Data Modelling Using Entities and Relationships

### Introduction

Conceptual modeling is a very important phase in designing a successful database application. Entity-Relationship (ER) model is a popular high-level conceptual data model. This model and its variations are frequently used for the conceptual design of database applications, and many database design tools employ its concepts.

### Using High-Level Conceptual Data Models for 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/implementation 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.

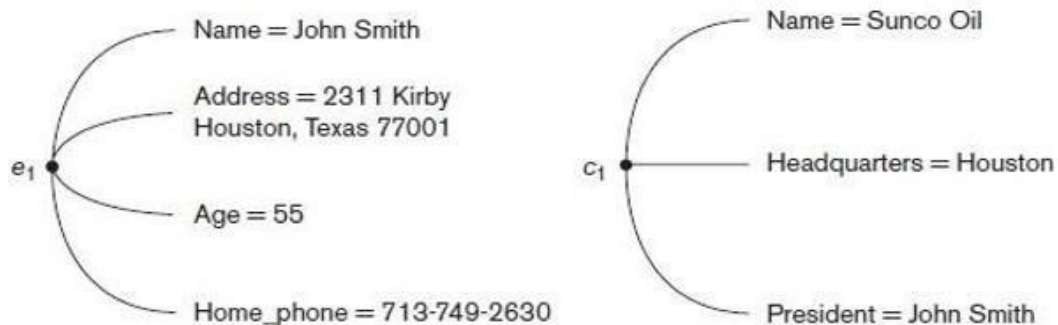
### Entity Types, Entity Sets, Attributes and Keys

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

#### Entities and Attributes

**Entity:** a thing 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 it may be an object with a conceptual existence (for instance, a company, a job, or a university course).

**Attributes:** Particular properties that describe entity. For example, an EMPLOYEE entity may be described by the employee's name, age, address, salary, and job.



Two entities, EMPLOYEE e1, and COMPANY c1, and their attributes are shown above in the figure.

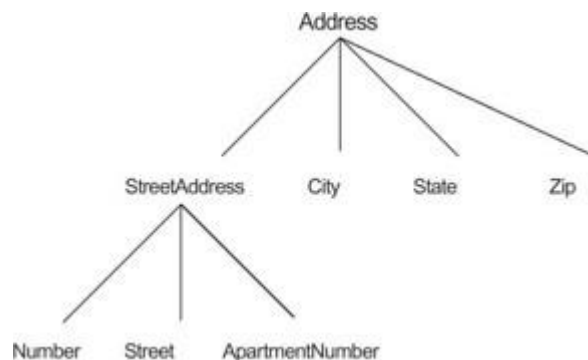
#### Types of attributes

- Composite versus Simple (Atomic) Attributes
- Single-valued versus multivalued
- Stored versus derived
- NULL values
- Complex attributes

### Composite versus Simple (Atomic) Attributes

Composite Attributes can be divided into smaller subparts, which represent more basic attributes with independent meanings. For example, the Address attribute of the EMPLOYEE entity can be subdivided into Street\_address, City, State, and Zip.

Composite attributes can form a hierarchy. For example, Street\_address can be further subdivided into three simple component attributes: Number, Street, and Apartment\_number. The value of a composite attribute is the concatenation of the values of its component simple attributes. A hierarchy of composite attributes is shown figure below,



Attributes that are not divisible are called **simple or atomic attributes**. Example SSN of an employee.

### Single-Valued versus Multivalued Attributes

Attributes that have a single value for a particular entity are called **single-valued**. For example, Age is a single-valued attribute of a person.

Attributes that can have a set of values for a particular entity are called **Multivalued Attributes**. For example Colors attribute for a car, or a College\_degrees attribute for a person. A multivalued attribute may have lower and upper bounds to constrain the number of values allowed for each individual entity. For example, the Colors attribute of a car may be restricted to have between one and three values, if we assume that a car can have three colors at most.

### Stored versus Derived Attributes

An attribute, which cannot be derived from other attribute are called **stored attribute**. For example, Birth\_Date of an employee

Attributes derived from other stored attribute are called **derived attribute**. For example age of an employee can be determined from the current (today's) date and Date of Birth.

## Null Value Attribute (Optional Attribute)

In some cases, a particular entity may not have an applicable value for an attribute. For example, 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. Similarly, a College\_degrees attribute applies only to people with college degrees. For such situations, a special value called **NULL** is created. An address of a single-family home would have NULL for its Apartment\_number attribute, and a person with no college degree would have NULL for College\_degrees. NULL can also be used if we do not know the value of an attribute for a particular entity.

## 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.

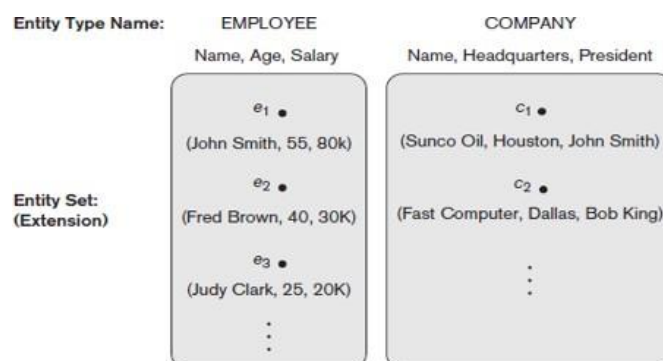
## Entity Types, Entity Sets, Keys, and Value Sets

### Entity Types

An entity type defines a collection (or set) of entities that have the same attributes. Each entity type in the database is described by its name and attributes. For example, a company employing hundreds of employees may want to store similar information concerning each of the employees. These employee entities share the same attributes, but each entity has its own value(s) for each attribute.

### Entity Sets

The collection of all entities of a particular entity type in the database at any point in time is called an entity set; the entity set is usually referred to using the same name as the entity type. For example, EMPLOYEE refers to both a type of entity as well as the current set of all employee entities in the database.



Two entity types, EMPLOYEE and COMPANY, and some member entities of each are shown in the above figure.

- 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

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 example, the Name attribute is a key of the COMPANY entity because no two companies are allowed to have the same name.

In ER diagrammatic notation, each key attribute has its name underlined inside the oval. Some entity types have more than one key attribute. For example, each of the Vehicle\_id and Registration attributes of the entity type CAR is a key in its own right.

Example: The CAR entity type with two key attributes, Registration and Vehicle\_id.

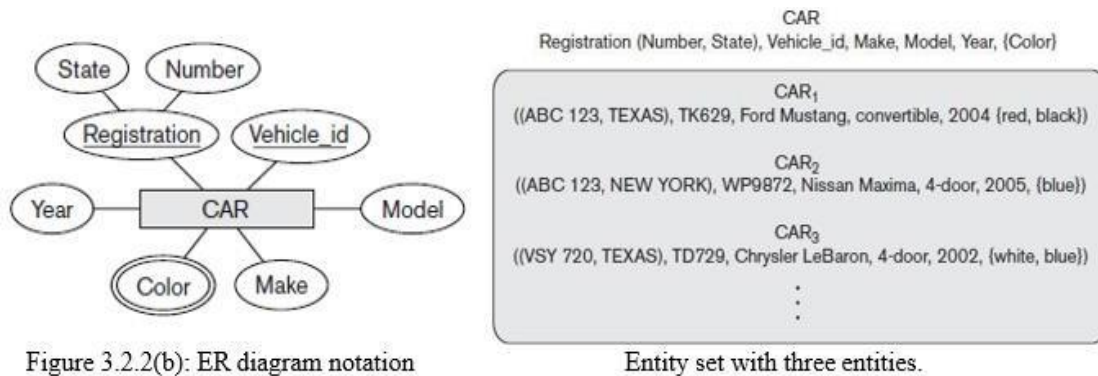


Figure 3.2.2(b): ER diagram notation

Entity set with three entities.

### **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. For example, 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.

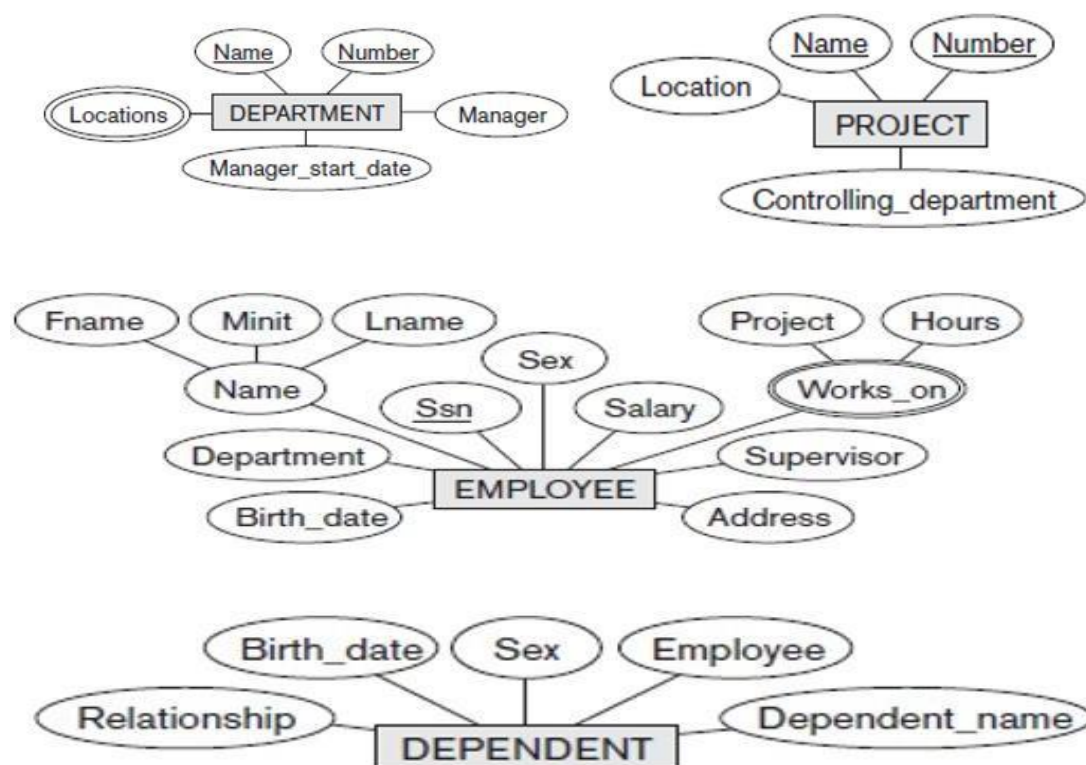
Value sets are not displayed in ER diagrams, and are specified using the basic data types available in most programming languages, such as integer, string, Boolean, float, enumerated, type, subrange and so on.

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  $A(e)$ . A NULL value is represented by the empty set.

## A Sample Database Application

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

- After the requirements collection and analysis phase, the database designers provide the following description of the miniworld:
- The company is organized into departments.
- Each department has a unique name, a unique number, and a particular employee who manages the department. We keep track of the start date when that employee began managing the department. A department may have several locations.
- A department controls a number of projects, each of which has a unique name, a unique number, and a single location.
- We store each employee's name, Social Security number, address, salary, gender, and birth date.
- An employee is assigned to one department, but may work on several projects, which are not necessarily controlled by the same department.
- We keep track of the current number of hours per week that an employee works on each project. We also keep track of the direct supervisor of each employee (who is another employee).
- We want to keep track of the dependents of each employee for insurance purposes. We keep each dependent's first name, gender, birth date, and relationship to the employee.



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

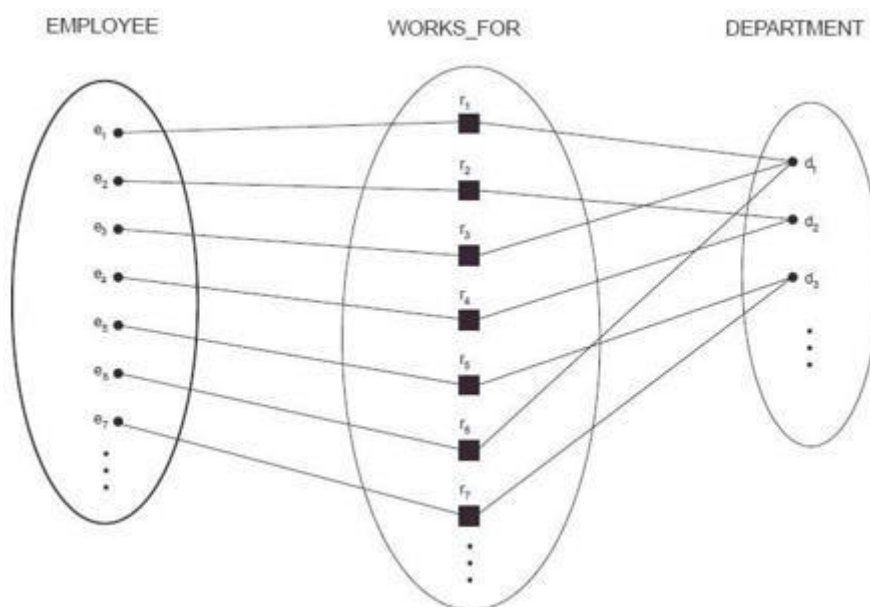
There are several implicit relationships among the various entity types. Whenever an attribute of one entity type refers to another entity type, some relationship exists. For example

- 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
- The attribute Supervisor of EMPLOYEE refers to another employee -the one who supervises this employee
- The attribute Department of EMPLOYEE refers to the department for which the employee works

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  $r_i = (e_1, e_2, \dots, e_n)$ .



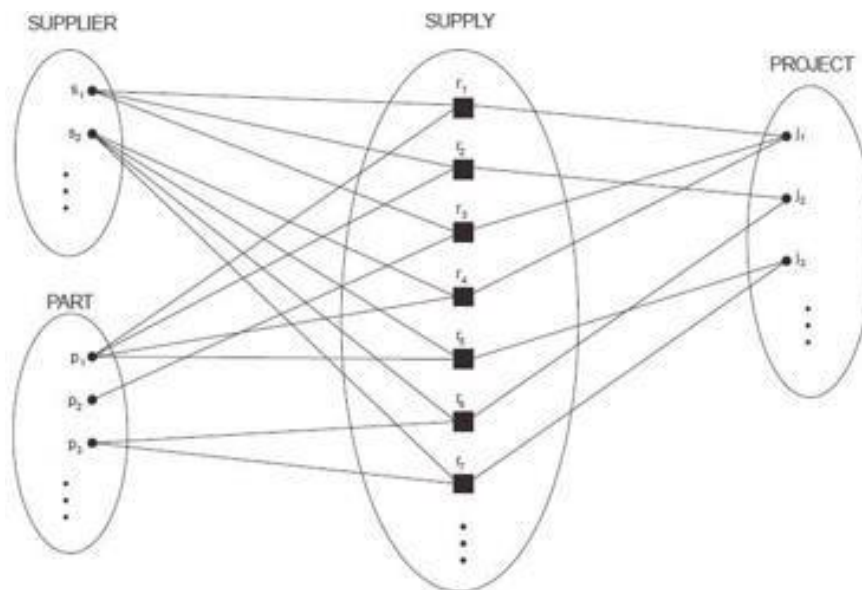


- Consider a relationship type WORKS\_FOR between the two entity types EMPLOYEE and DEPARTMENT, which associates each employee with the department for which the employee works. Each relationship instance in the relationship set WORKS\_FOR associates one EMPLOYEE entity and one DEPARTMENT entity.
- The employee's e1, e3, and e6 work for department d1
- The employee's e2 and e4 work for department d2; and the employee's 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

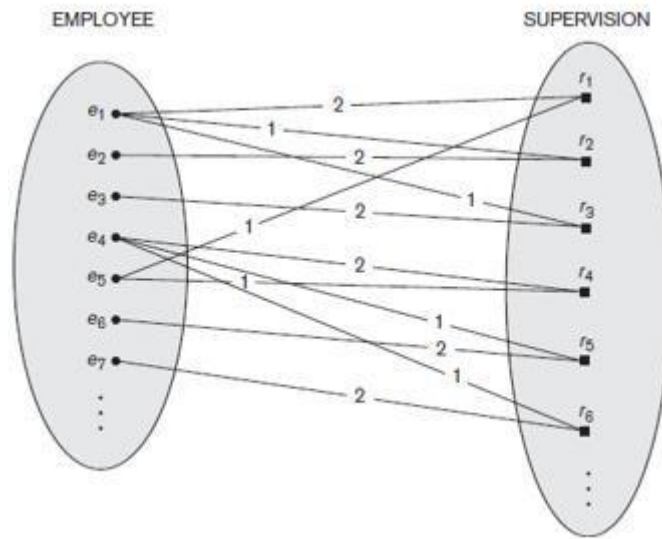
- Hence, the WORKS\_FOR relationship is of degree two.
- A relationship type of degree two is called binary, and one of degree three is called ternary
- An example of a ternary relationship is SUPPLY, shown in Figure, where each relationship instance  $r_i$  associates three entities—a supplier  $s$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



A recursive relationship SUPERVISION between EMPLOYEE in the supervisor role (1) and EMPLOYEE in the subordinate role is shown in the above figure.

### 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:

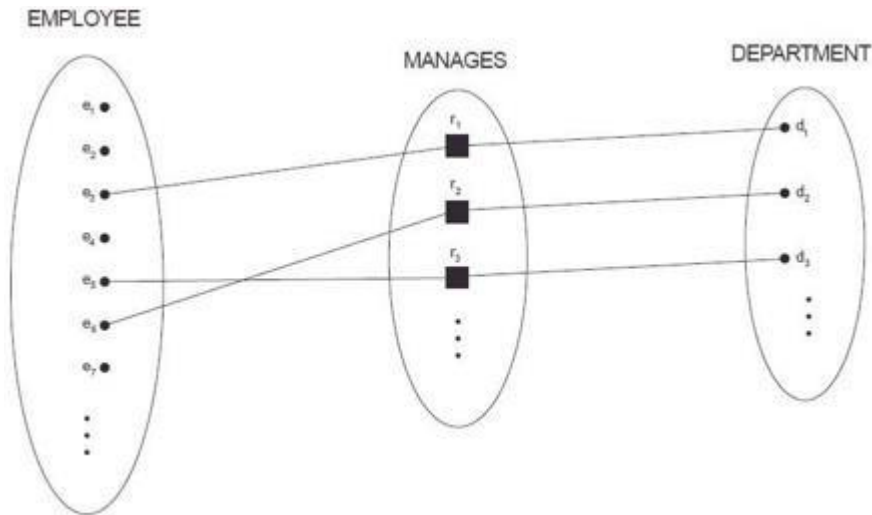
Cardinality ratio	}	Structural constraints
Participation constraint		

### Cardinality Ratios for Binary Relationships

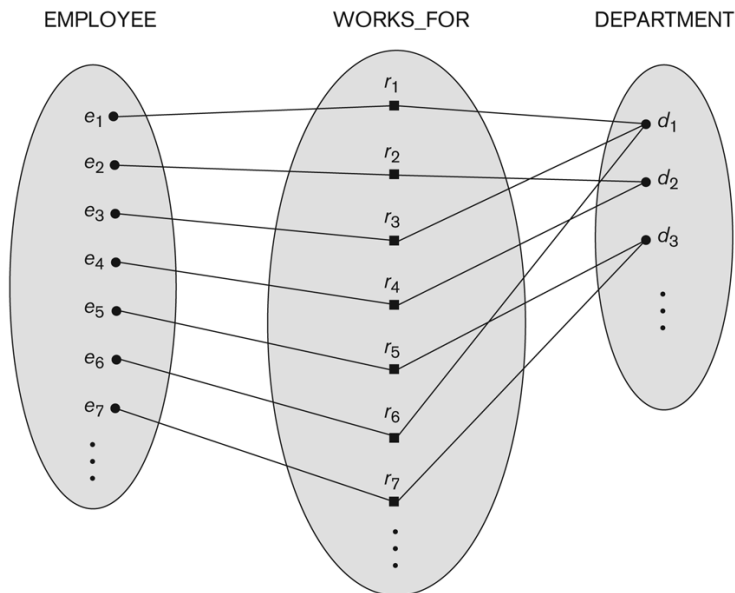
The cardinality ratio for a binary relationship specifies the maximum number of relationship instances that an entity can participate in. For example, in the WORKS\_FOR binary relationship type, DEPARTMENT:EMPLOYEE is of cardinality ratio 1:N, meaning that each department can be related to any number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

### Example of a 1:1 binary relationship

- MANAGES which relates a department entity to the employee who manages that department.
- This represents the miniworld constraints that—at any point in time—an employee can manage one department only and a department can have one manager only.

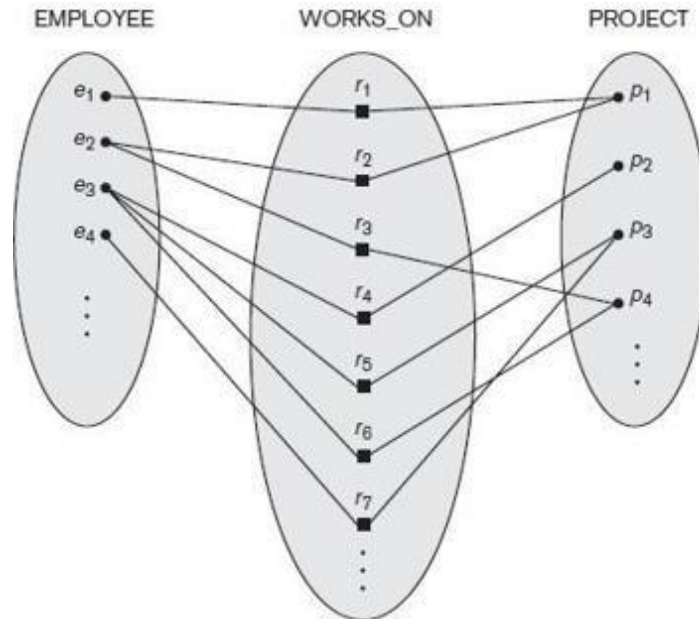


### One-to-Many(1:N) & Many-to-one (N:1) Relationship



### Example of a M:N binary relationship

- The relationship type **WORKS\_ON** is of cardinality ratio M:N, because the mini-world rule is that 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.



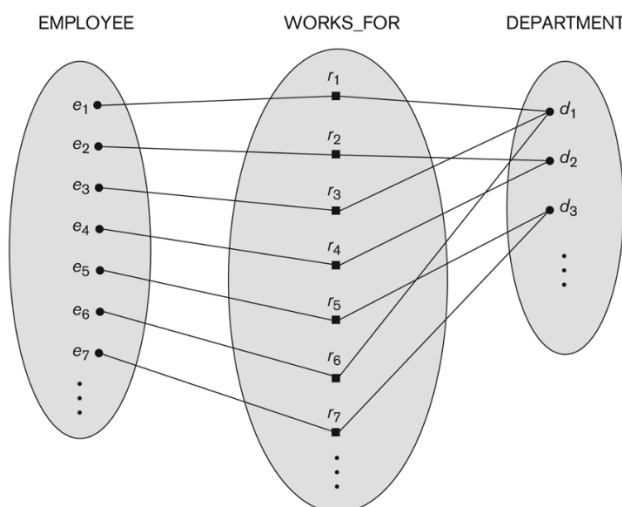
### Participation Constraints and Existence Dependencies

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
- Partial

#### Total participation

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.

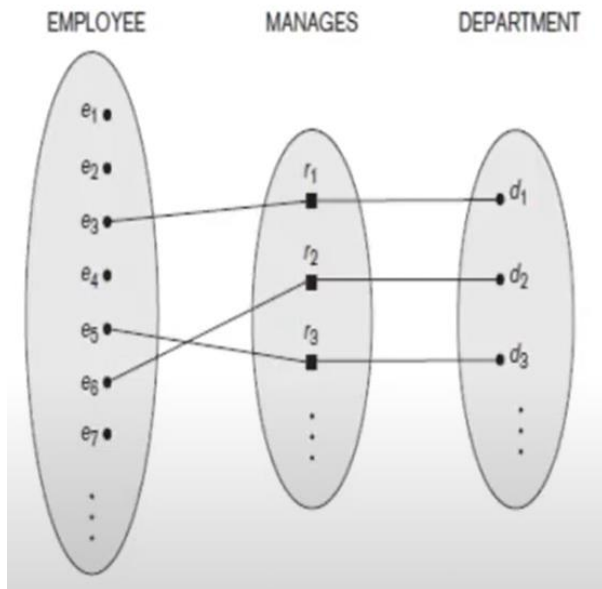


#### Partial participation

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 is displayed as a double line connecting the participating entity type to the relationship, whereas partial participation is represented by a single line.

cardinality ratio + participation constraints = structural constraints of a relationship type.



### **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.

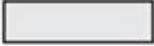





### **Weak Entity Types**


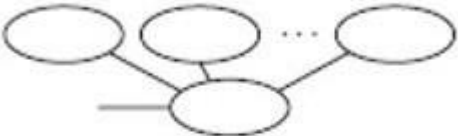
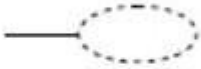
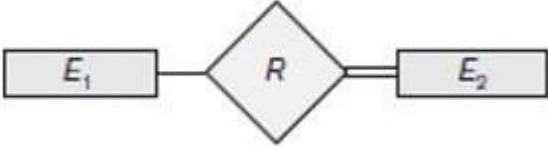
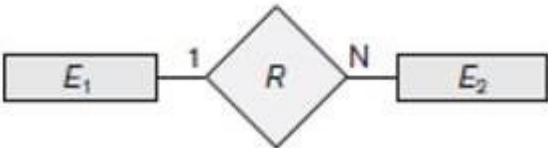
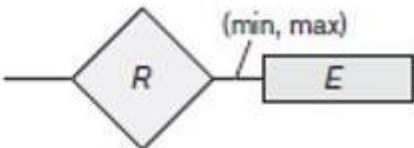
- Entity types that do not have key attributes of their own are called weak entity types
- In contrast, regular entity types that do have a key attribute—are called strong entity types
- Entities belonging to a weak entity type are identified by being related to specific entities from another entity 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**

### **Summary of Notation for ER Diagrams**

Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Identifying Relationship
	Attribute
	Key Attribute

Symbol	Meaning
	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 $R$

### 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**

In general, the schema design process should be considered an iterative refinement process, where an initial design is created and then iteratively refined until the most suitable design is reached. Some of the refinements that are often used include the following:

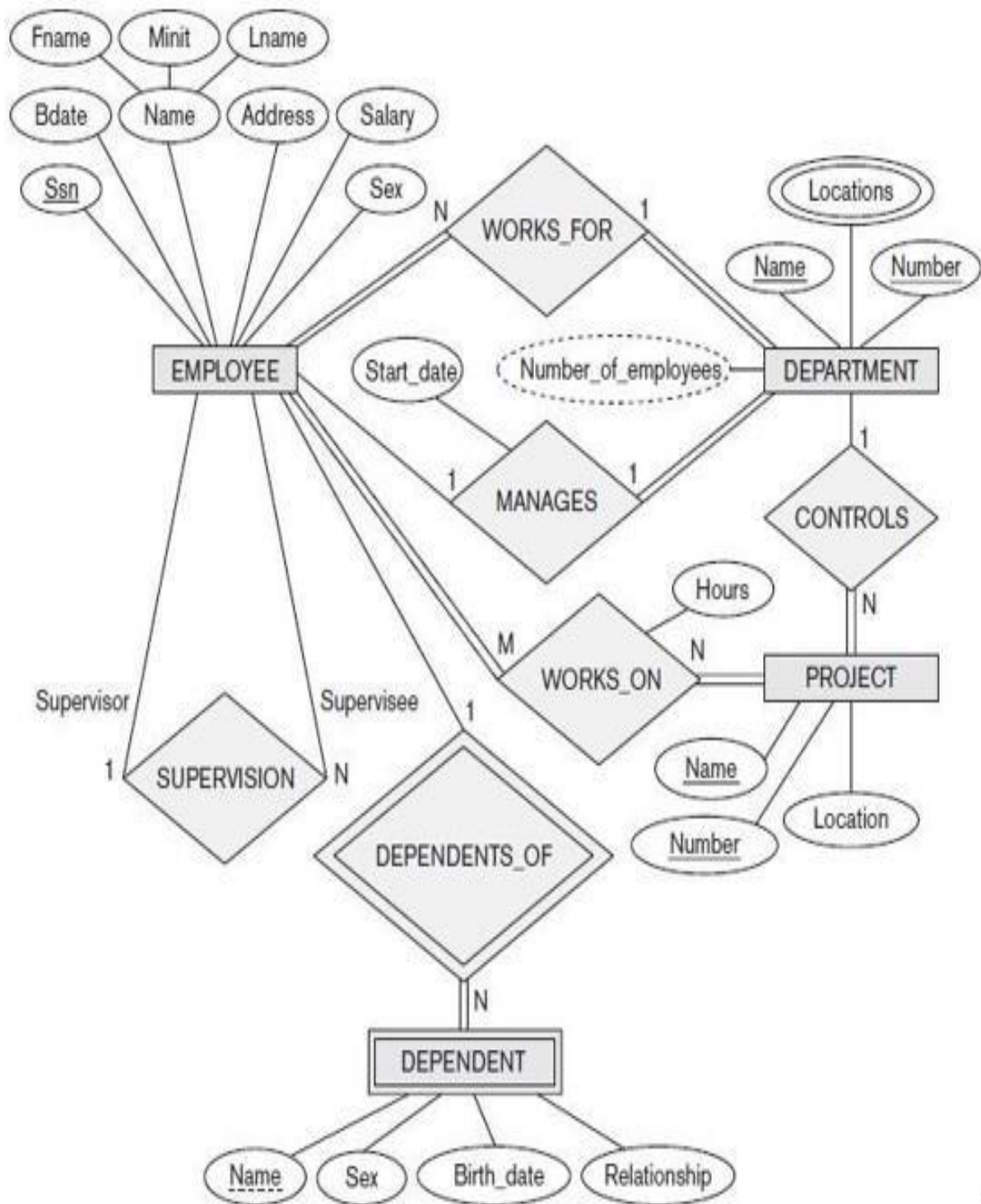
- A concept may be first modeled as an attribute and then refined into a relationship because it is determined that the attribute is a reference to another entity type. It is often the case that a pair of such attributes that are inverses of one another are refined into a binary relationship.
- Similarly,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

## **Alternative Notations for ER Diagrams**

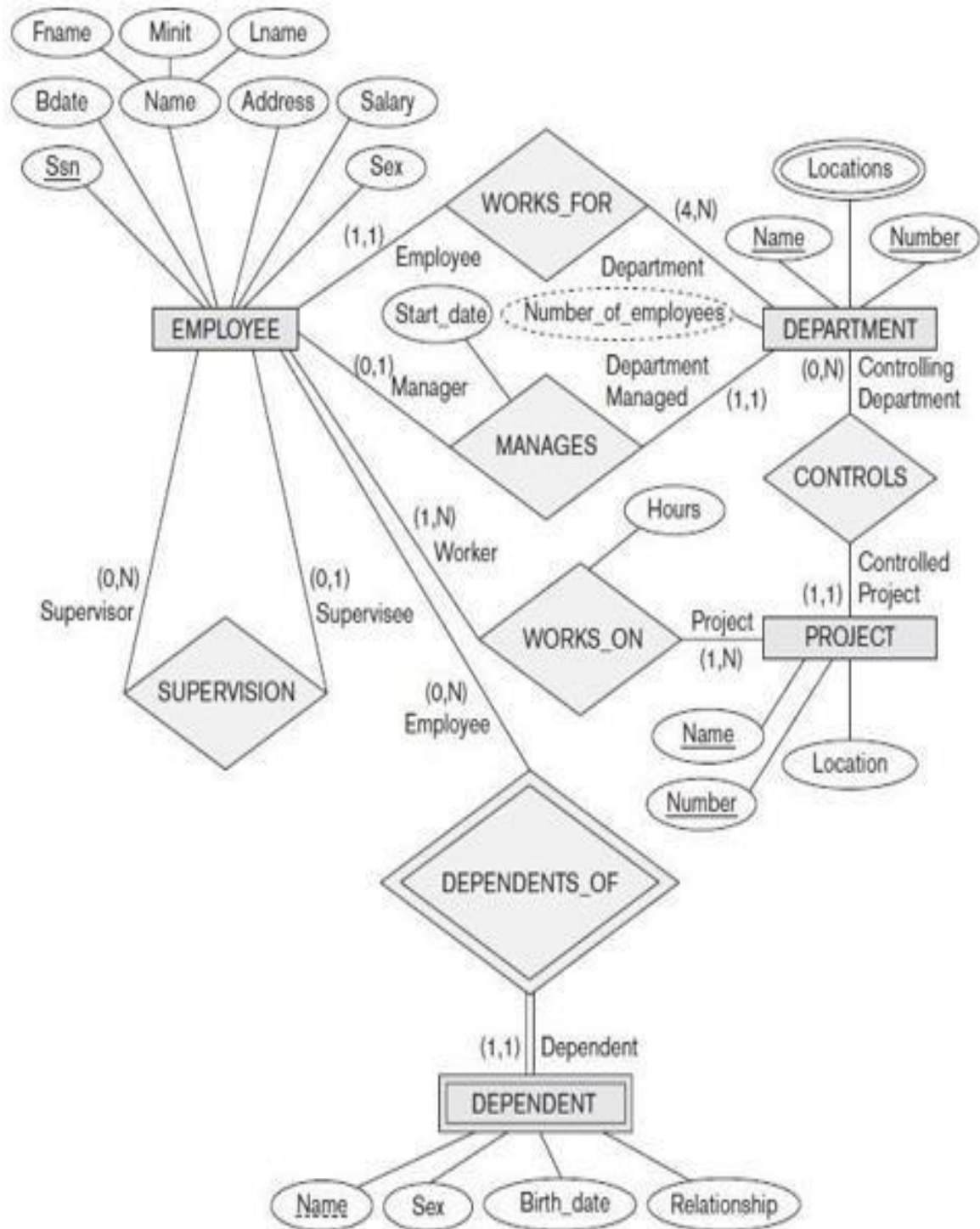
There are many alternative diagrammatic notations for displaying ER diagrams. One alternative ER notation for specifying structural constraints on relationships, which replaces the cardinality ratio (1:1, 1:N, M:N) and single/double line notation for participation constraints. This notation involves associating a pair of integer numbers (min, max) with each participation of an entity type E in a relationship type R, where  $0 \leq \min \leq \max$  and  $\max \geq 1$ .

The numbers mean that for each entity e in E, e must participate in at least min and at most max relationship instances in R at any point in time. In this method, min = 0 implies partial participation, whereas min > 0 implies total participation.





ER diagram for Company Database is shown above.



ER diagram for Company Database is shown above (using alternative notation).