

## **Module 1**

# **Introduction to Databases**

- Introduction to Database
- Characteristics of database approach
- Advantages of using the DBMS approach
- History of database applications

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

- Data Models
- Schemas and Instances
- Three schema architecture and data independence
- Database languages and interfaces
- The Database System environment

### **Conceptual Data Modeling using Entities and Relationships**

- Entity types
- Entity sets
- Attributes, roles, and structural constraints
- Weak entity types
- ER diagrams and examples
- Specialization and Generalization.

**Textbook 1: Chapter 1.1 to 1.8, 2.1 to 2.6, 3.1 to 3.10**

# Introduction to Databases

## Introduction

Databases and database systems are an essential component of life in modern society: most of us encounter several activities every day that involve some interaction with a database.

Databases and database technology have had 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, social media, engineering, medicine, genetics, law, education, and library science.

## Database

A **database** is a collection of related data. Or it is an organized collection of related data, stored and accessed electronically.

**Data** means known facts that can be recorded and that have implicit meaning.

A database has the implicit properties:

- A database represents some aspect of the real world(**miniworld**).
- Changes to the miniworld are reflected in the database.
- A database is a logically coherent collection of data with some inherent meaning.
- A database is designed, built, and populated with data for a specific purpose.
- It has an intended group of users and some preconceived applications in which these users are interested.

A database has some source from which data is derived, some degree of interaction with events in the real world, and an audience that is actively interested in its contents. It can be of any size and complexity. A database may be generated and maintained manually or it may be computerized.

## Database management system (DBMS)

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

**Definition:** “The DBMS is a *general-purpose software system* that facilitates the processes of *defining, constructing, manipulating, and sharing* databases among various users and applications”.

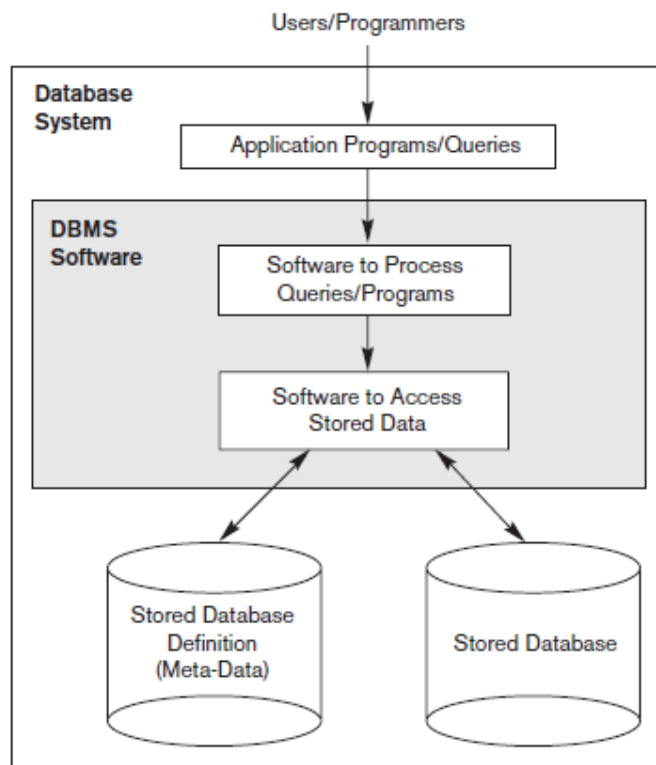
**Defining** a database involves specifying the data types, structures, and constraints of the data to be stored in the database.

**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.

- The database definition or descriptive information is also stored by the DBMS in the form of a **database catalog or dictionary**; it is called **meta-data**.
- 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.
- The DBMS also *protects* 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 lifecycle 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.
- In the database approach, a single repository maintains data that is defined once and then accessed by various users repeatedly through queries, transactions, and application programs.
- The database and DBMS software together form a **database system**.



A simplified database system environment

### An Example

UNIVERSITY database for maintaining information concerning students, courses, and grades in a university environment.

#### 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

**GRADE\_REPORT**

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

## Characteristics of the Database Approach

### ■ Self-describing nature of a database system

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

**RELATIONS**

Relation_name	No_of_columns
STUDENT	4
COURSE	4
SECTION	5
GRADE_REPORT	3

**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
....	....	....

An example of a database catalog for the UNIVERSITY database.

### ■ 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. This property is called **program-data independence**.

In database systems, users can define operations on data as part of the database definitions. 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**. 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.

#### ■ 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.

#### ■ Sharing of data and multiuser transaction processing

A multiuser DBMS 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.

A **transaction** is an *executing program* or *process* that includes one or more database accesses, such as reading or updating of database records. Each transaction is supposed to execute a logically correct database access if executed in its entirety without interference from other transactions. 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.

#### File System vs DBMS – Difference between File System and DBMS

File Management System	Database Management System
File System is a general, easy-to-use system to store general files which require less security and constraints.	Database management system is used when security constraints are high.
Data Redundancy is more in file management system.	Data Redundancy is less in database management system.

Data Inconsistency is more in file system.	Data Inconsistency is less in DBMS
Centralization is hard	Centralization is achieved
User locates the physical address of the files to access data in File Management System.	User is unaware of physical address data.
Security is low	Security is high
Stores unstructured data as isolated data files/entities.	Stores structured data which have well defined constraints and interrelation.

## Actors on the Scene

### Database Administrators

Database Administrators responsible for

- Administering primary resource database itself, and the secondary resource DBMS and related software.
- 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.

### 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
- To communicate with all prospective database users in order to understand their requirements and to create a design that meets these requirements.
- To interact with each potential group of users and develop **views** of the database that meet the data and processing requirements of these groups.
- Each view is then analyzed and *integrated* with the views of other user groups. The final database design must be capable of supporting the requirements of all user groups.

### End Users

End users are the people, whose jobs require access to the database for querying, updating, and generating reports; the database primarily exists for their use. There are several categories of end users:

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

■ **Naive or parametric end users** make up a sizable portion of database end users. They constantly querying and updating the database, using standard types of queries and update called **canned**

**transactions**—that have been carefully programmed and tested. They need to learn very little about the facilities provided by the DBMS.

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. They try to learn most of the DBMS facilities.

■ **Standalone users** maintain personal databases by using ready-made program packages that provide easy-to-use menu-based or graphics-based interfaces. They typically become very proficient in using a specific software package.

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

### **Workers behind the Scene**

These persons are typically not interested in the database itself. They are associated with the design, development and operation and maintenance of DBMS software and system environment. They are called as workers behind the scene.

Includes: DBMS Designers and Implementers, Tool Developers (Packages for database design, performance monitoring, natural language, or GUI, prototyping, simulation, and test data generation), Operators and maintenance personnel.

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

- Controlled redundancy
- Restricted unauthorized access ( Security at various levels)
- Providing Persistent Storage for Program Objects
- Providing Storage Structures and Search Techniques for Efficient Query Processing
- Providing multiple users and interfaces
- Representing complex relationships among data
- Enforcing integrity constraints
- Providing backup and recovery
- Representing Complex Relationships among Data
- Handling large volume of data and allowing multiple views

### **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. This facilitates communication and cooperation among various departments, projects, and users within the organization.

**Reduced Application Development Time.**

Once a database is up and running, substantially less time is generally required to create new applications using DBMS facilities.

**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.

**Economies of Scale.**

Consolidation of data and applications, reduces the amount of wasteful overlap between activities of data-processing personnel. This enables the whole organization to invest in more powerful processors, storage devices, or networking gear, rather than having each department purchase its own equipment. This reduces overall costs of operation and management.

## History of Database Applications

### 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 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

### **Application of Database System**

- **Railway Reservation System**
- **Library Management System**
- **Banking**
- **Universities and colleges**
- **Credit card transactions**
- **Social Media Sites**
- **Telecommunications**
- **Finance**
- **Military**
- **Online Shopping**
- **Human Resource Management**
- **Manufacturing**
- **Airline Reservation system**

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

The architecture of DBMS packages has evolved from the early monolithic systems, where the whole DBMS software package was one tightly integrated system, to the modern DBMS packages that are modular in design, with a client /server system architecture.

## Data Models, Schemas, and Instances

One fundamental characteristic of the database approach is that it provides some level of data abstraction. **Data abstraction** generally refers to the suppression of details of data organization and storage, and the highlighting of the essential features for an improved understanding of data. So that different users can perceive data at their preferred level of detail.

A **data model**—a collection of concepts that can be used to describe the structure of a database—provides the necessary means to achieve data abstraction. By *structure of a database* means the data types, relationships, and constraints that apply to the data.

Data models also include a set of **basic operations** for specifying retrievals and updates on the database.

### Categories of Data Models

- **High-level or conceptual data models**
- **Representational or implementation data models**
- **Low-level or physical data models**

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

Example an **Entity–relationship model**. An **entity** represents a real-world object or concept, such as an employee or project. An **attribute** represents some property of interest that further describes an entity, such as the employee's name or salary. A **relationship** among two or more entities represents an association among the entities, for example, a works-on relationship between an employee and a project.

Representational or implementation data models Provide concepts that fall between the above two, used by many commercial DBMS implementations. Used most frequently in traditional commercial DBMSs. These include the widely used **relational data model**, **network** and **hierarchical models**—that have been widely used in the past. Here data is represented 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. Concepts provided are generally meant for computer specialists, not for end users.

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.

## Schemas, Database State or Instances

- **Database Schema:** The description of a database that includes descriptions of the database structure, data types, and the constraints on the database.
- **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.

STUDENT

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

COURSE

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

SECTION

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

GRADE REPORT

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

STUDENT

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

Schema diagram for the UNIVERSITY database

Example of a database state for STUDENT

- **Schema construct:** Each object in the schema—such as STUDENT or COURSE—is called a schema construct.
- **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. 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 database schema changes very infrequently. The database state changes every time the database is updated.
- The schema is sometimes called the **intension**, and a database state is called an **extension** of the schema.

## Three-Schema Architecture

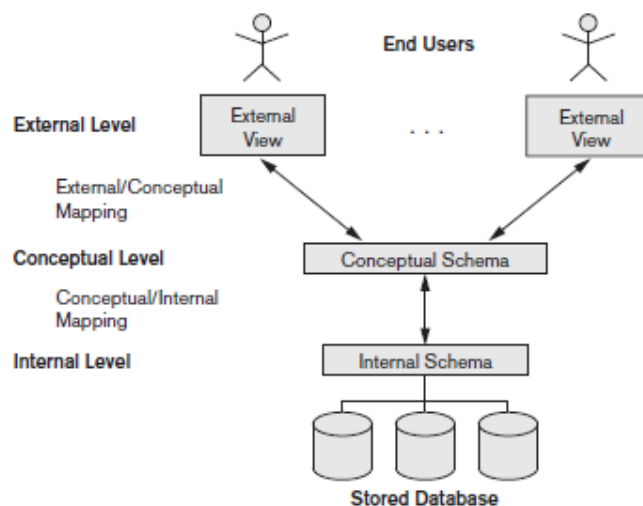
The goal of the three-schema architecture is to separate the user applications from the physical database. Here schemas can be defined at the following three levels:

- **Internal level**
- **Conceptual level**
- **External or view level**

The **internal level** has an **internal schema**, which describes the physical storage structure of the database. It 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**, which describes the structure of the whole database for a community of users. It hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. Usually, a representational data model is used to describe the conceptual schema.

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. Typically implemented using a representational data model.



**The three-schema architecture**

- The user can easily visualize the schema levels in a database system.
- Most DBMSs do not separate the three levels completely and explicitly.
- Here, 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 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 is the capacity to change the schema at one level of a database system without having to change the schema at the next higher level.

Two types of data independence:

**Logical data independence** is the capacity to change the conceptual schema without having to change external schemas or application programs. The conceptual schema may be changed to expand the database, to change constraints, or to reduce the database.

After the conceptual schema undergoes a logical reorganization, application programs that reference the external schema constructs must work as before.

**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.

If the same data as before remains in the database, we should not have to change the conceptual schema.

## DBMS Languages

A DBMS has appropriate languages and interfaces to express database queries and updates. Database languages can be used to read, store and update the data in the database.

### Types of DBMS languages:

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Control Language (DCL)
- Transaction Control Language (TCL)
- Storage definition language (SDL)
- View definition language (VDL)

### Data Definition Language (DDL)

- DDL is used to define the database schema.
- It is used to create schema, tables, indexes, constraints etc. in the database.
- Using the DDL statements, you can create the skeleton of the database.
- 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.

Here are some tasks that come under DDL:

- **Create:** It is used to create the database or its objects (like table, index, function, views, store procedure and triggers).
- **Alter:** It is used to alter the structure of the database.
- **Drop:** It is used to delete objects from the database.
- **Truncate:** It is used to remove all records from a table, including all spaces allocated for the records are removed.
- **Rename:** It is used to rename an object.
- **Comment:** It is used to comment to the data dictionary.

These commands are used to update the database schema that's why they come under Data definition language.

### Data Manipulation Language (DML)

DML is used for accessing and manipulating data in a database. Like retrieval, insertion, deletion, and modification of the data. It handles user requests. There are two main types of DMLs.

- A high-level or nonprocedural DML statements either to be entered interactively from a display monitor or terminal or to be embedded in a general-purpose programming language. Also called **set-at-a-time** or **set-oriented** DMLs.
- A low-level or procedural DML must be embedded in a general-purpose programming language. This type of DML typically retrieves individual records or objects from the database and processes each separately. Also called as **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**. On the other hand, a high-level DML used in a standalone interactive manner is called a **query language**. In

Here are some tasks that come under DML:

- **Select**: It is used to retrieve data from a database.
- **Insert**: It is used to insert data into a table.
- **Update**: It is used to update existing data within a table.
- **Delete**: It is used to delete all records from a table.
- 

### Data Control Language (DCL)

- DCL mainly deals with the rights, permissions and other controls of the database system.

Here are some tasks that come under DCL:

- **Grant**: It is used to give user access privileges to a database.
- **Revoke**: It is used to take back permissions from the user.

There are the following operations which has authorization of Revoke: CONNECT, INSERT, USAGE, EXECUTE, DELETE, UPDATE and SELECT.

### Transaction Control Language (TCL)

TCL is used to run the changes made by the DML statement. TCL commands deals with the transaction within the database. Here are some tasks that come under TCL:

- **Commit**: It is used to save the transaction on the database.
- **Rollback**: It is used to restore the database to original since the last Commit.
- **SAVEPOINT**—sets a savepoint within a transaction.
- **SET TRANSACTION**—specify characteristics for the transaction.

### Storage definition language (SDL)

Storage definition language (SDL) is used to specify the internal schema. In most relational DBMSs today, there *is no specific language* that performs the role of SDL. Instead, the internal schema is specified by a combination of functions, parameters, and specifications related to storage of files.

### View definition language (VDL)

View definition language (VDL), to specify user views and their mappings to the conceptual schema, but in most DBMSs *the DDL is used to define both conceptual and external schemas*. In relational DBMSs, SQL is used in the role of VDL to define user or application **views** as results of predefined queries

## 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.

**Apps for Mobile Devices.** These interfaces present mobile users with access to their data. For example, banking, reservations, and insurance companies.

**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. Example: SQL\*Forms, Oracle Forms

**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.

**Natural Language Interfaces.** These interfaces accept requests written in English or some other language and attempt to *understand* them. It usually has its own *schema*, as well as a dictionary of important words.

**Keyword-based Database Search.** It accepts strings of natural language (like English or Spanish) words and match them with documents at specific sites (for local search engines) or Web pages on the Web at large (for engines like Google or Ask).

**Speech Input and Output** 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. Applications with limited vocabularies, such as inquiries for telephone directory, flight arrival/departure.

**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.

**Interfaces for the DBA.** Used only by the DBA staff to execute privileged commands. These include commands for creating accounts, setting system parameters, granting account authorization, changing a schema, and reorganizing the storage structures of a database.

## DBMS Programming Language Interfaces

Programmer interfaces for embedding DML in programming languages:

1. Embedded Approach: e.g embedded SQL (for C, C++, etc.), SQLJ (for Java)
2. Procedure Call Approach: e.g. JDBC for Java, ODBC for other programming languages

3. Database Programming Language Approach: e.g. ORACLE has PL/SQL, a programming language based on SQL; language incorporates SQL and its data types as integral components

## The Database System Environment

A DBMS is a complex software system. The typical DBMS components are divided into two parts:

The various users of the database environment and their interfaces.

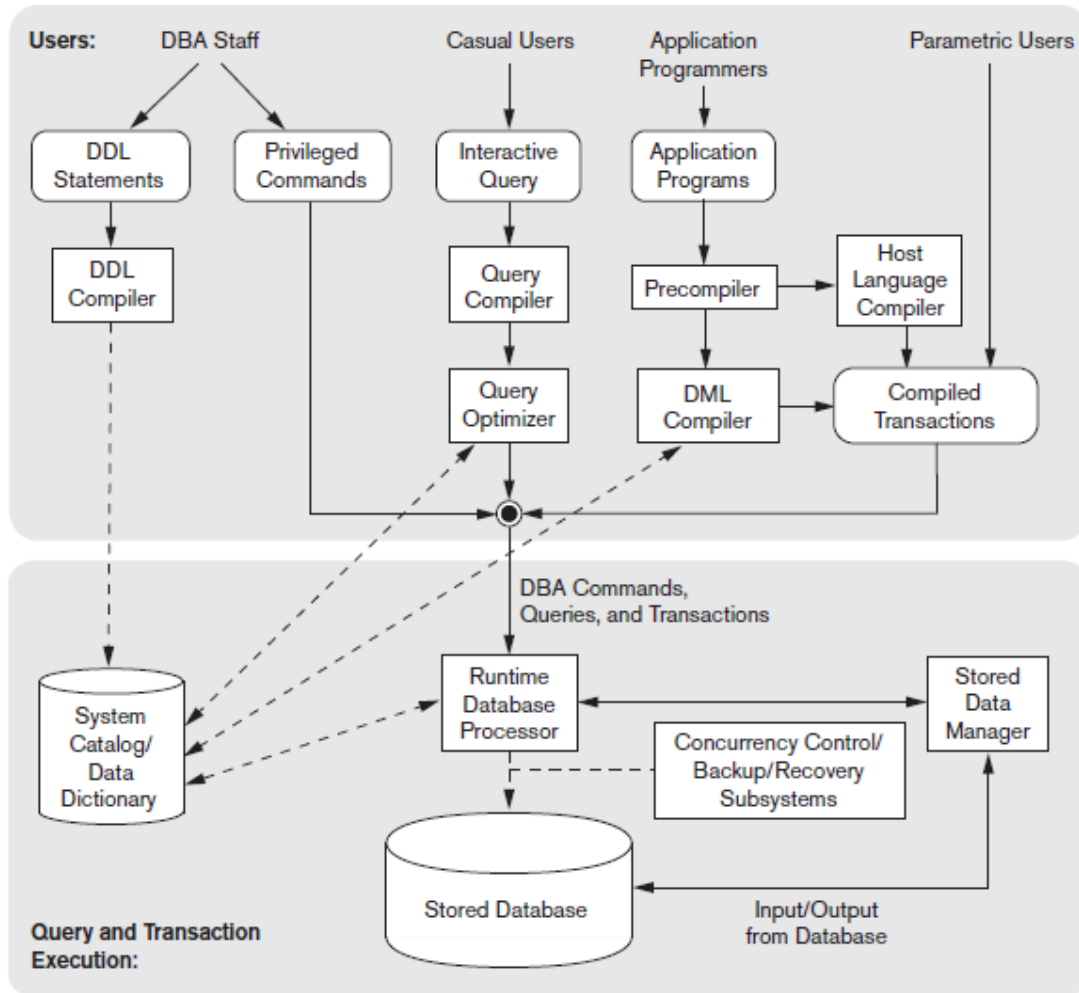
The internal modules of the DBMS responsible for storage of data and processing of transactions.

- 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.
- 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.

### Database Users

- **DBA staff:** The DBA staff works on defining the database and tuning it by making changes to its definition using the DDL and other privileged commands. The DDL compiler processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta-data) in the DBMS catalog.
- **Casual users** and persons with occasional need for information from the database interact using the interactive query interface or to access canned transactions. These queries are parsed and validated for correctness of the query syntax by a **query compiler** that compiles them into an internal form and then is subjected to query optimization. The **query optimizer** is concerned with the rearrangement and possible reordering of operations, elimination of redundancies, and use of efficient search algorithms during execution.
- **Application programmers** write programs in host languages such as Java, C, or C++ that are submitted to a precompiler, which extracts DML commands from an application program. 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.
- **Parametric users** who do data entry work by supplying parameters to predefined transactions.





Component modules of a DBMS and their interactions

1. 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. In addition, the catalog stores many other types of information that are needed by the DBMS modules, which can then look up the catalog information as needed.
  - **Runtime database processor** executes (1) the privileged commands, (2) the executable query plans, and (3) the canned transactions with runtime parameters. It works with the **system catalog** and may update it with statistics.
  - Runtime database processor also works with the **stored data manager**, which in turn uses basic operating system services for carrying out low-level input/output (read/write) operations between the disk and main memory.
  - The runtime database processor handles other aspects of data transfer, such as management of buffers in the main memory.
  - **concurrency control** and **backup and recovery systems modules** are used for transaction management.

## Database System Utilities

Most DBMSs have **database utilities** that help the DBA manage the database system.

- **Loading:** A loading utility is used to load existing data files—such as text files or sequential files—into the database.
- **Backup.** A backup utility creates a backup copy of the database, usually by dumping the entire database onto tape or other mass storage medium.
- **Database storage reorganization.** This utility can be used to reorganize a set of database files into different file organizations and create new access paths to improve performance.
- **Performance monitoring.** Such a utility monitors database usage and provides statistics to the DBA. The DBA uses the statistics in making decisions such as whether or not to reorganize files or whether to add or drop indexes to improve performance.
- Other utilities may be available for sorting files, handling data compression, monitoring access by users, interfacing with the network, and performing other functions.

## Other Tools

### Data dictionary / repository:

- Used to store schema descriptions and other information such as design decisions, application program descriptions, user information, usage standards, etc.
- Active data dictionary is accessed by DBMS software and users/DBA.
- Passive data dictionary is accessed by users/DBA only.

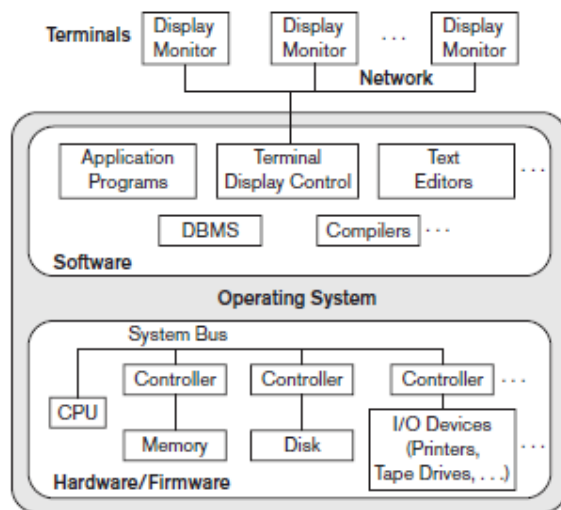
Application Development Environments and CASE (computer-aided software engineering) tools: Examples:

- PowerBuilder (Sybase)
- JBuilder (Borland)
- JDeveloper 10G (Oracle)

## Centralized and Client/Server Architectures for DBMSs

### Centralized DBMSs Architecture

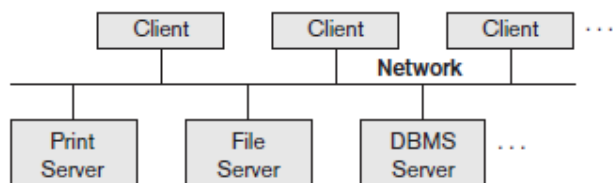
- Older architectures used mainframe computers to provide the main processing for all system functions, including user application programs and user interface programs, as well as all the DBMS functionality.
- Most users accessed the DBMS via computer terminals that did not have processing power and only provided display capabilities.
- Therefore, all processing was performed remotely on the computer system housing the DBMS, and only display information and controls were sent from the computer to the display terminals, which were connected to the central computer via various types of communications networks.



### Basic Client/Server Architectures

- The **client/server architecture** was developed to deal with computing environments in which a large number of PCs, workstations, file servers, printers, database servers, Web servers, e-mail servers, and other software and equipment are connected via a network.
- The idea is to define **specialized servers** with specific functionalities. For example, it is possible to connect a number of PCs or small workstations as clients to a **file server** that maintains the files of the client machines.
- The resources provided by specialized servers can be accessed by many client machines.

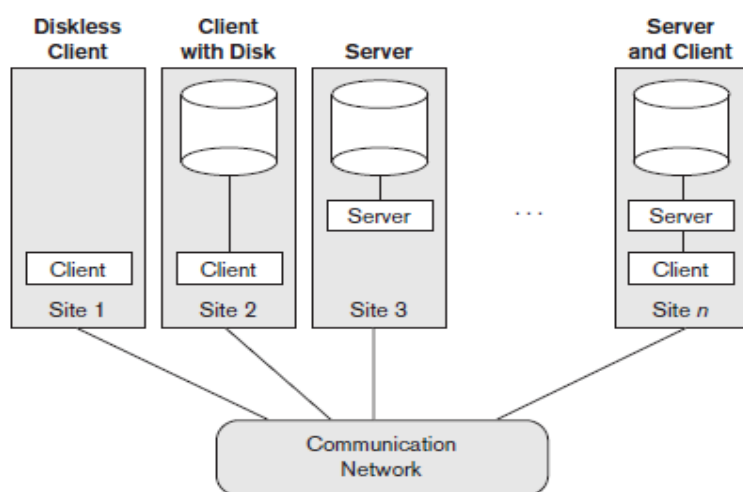
- The **client machines** provide the user with the appropriate interfaces to utilize these servers, as well as with local processing power to run local applications.
- A **server** is a system containing both hardware and software that can provide services to the client machines, such as file access, printing, archiving, or database access.



Logical two-tier client/server architecture.

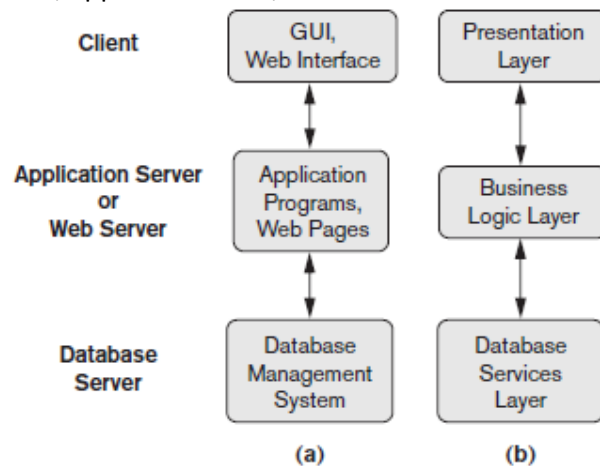
### Two-Tier Client/Server Architectures for DBMSs

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



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

- The three-tier architecture, which adds an intermediate layer between the client and the database server.
- This intermediate layer or middle tier is called the application server or the Web server, runs application programs and storing business rules that are used to access data from the database server.
- Improves database security by checking a client's credentials before forwarding a request to the database server.
- Clients contain user interfaces and Web browsers.
- The intermediate server accepts requests from the client, processes the request and sends database queries and commands to the database server, and then acts as a conduit for passing (partially) processed data from the database server to the clients, where it may be processed further and filtered to be presented to the users.
- Thus, the user interface, application rules, and data access act as the three tiers.



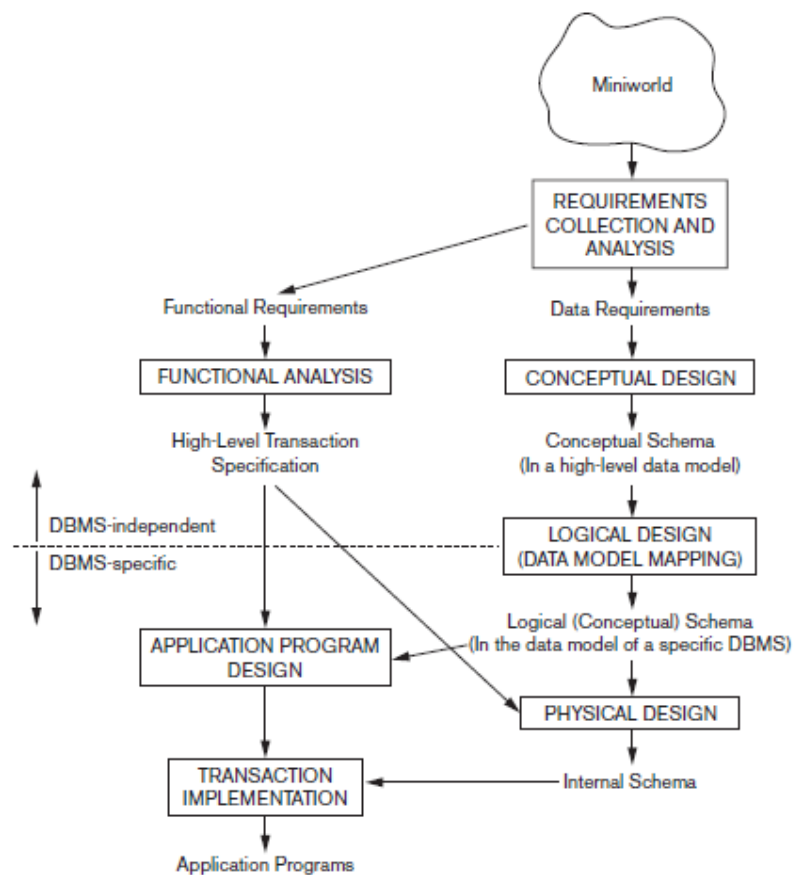
Logical three-tier client/server architecture

## Conceptual Data Modeling using Entities and Relationships

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

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

- The database design process begins with **requirements collection and analysis**.
- During this step, the database designers interview prospective database users to understand and document their **data requirements**.
- In parallel with this, specify the known **functional requirements** of the application, like user defined **operations** (or **transactions**) that will be applied to the database, including both retrievals and updates.
- The next step is to create a **conceptual schema** for the database, using a high-level conceptual data model. 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;
- During or after the conceptual schema design, the basic data model operations can be used to specify the high-level user queries and operations identified during functional analysis and confirm all the identified functional requirements are met.
- Modifications to the conceptual schema can be introduced if required.
- The next step is the actual implementation of the database, using a commercial DBMS, using 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, resulting in a database schema.
- This step is called **logical design** or **data model mapping**
- Data model mapping is often automated or semi automated within the database design tools.
- The last step is the **physical design** phase, during which the internal storage structures, file organizations, indexes, access paths, and physical design parameters for the database files are specified.
- In parallel with these activities, application programs are designed and implemented as database transactions corresponding to the high-level transaction specifications.

## The ER model Concepts

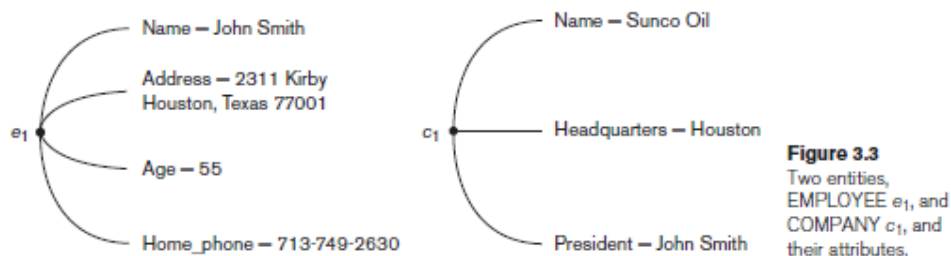
The ER model describes data as **entities**, **relationships**, and **attributes**.

### Entities and Attributes

**Entity:** The basic concept that the ER model represents is an entity, which is a thing or object in the real world with an independent existence.

An entity may be an object with a physical existence (for example, a particular person, car, house, or employee) or it may be an object with a conceptual existence (for instance, a company, a job, or a university course).

**Attributes:** Each entity has attributes. The attributes are particular properties that describe entity. For example, an EMPLOYEE entity may be described by the employee's name, age, address, salary, and job. The attribute values that describe each entity become a major part of the data stored in the database.



### Types of attributes

**Composite attributes** can be divided into smaller subparts, which represent more basic attributes with independent meanings.

Example, the Address attribute of the EMPLOYEE entity can be subdivided into Street\_address, City, State, and Zip,3 with the values '2311 Kirby', 'Houston', 'Texas', and '77001'.

**Simple (Atomic) Attributes** that are not divisible are called simple or atomic attributes.

Example: the age of the EMPLOYEE entity.

**Single-Valued** attributes have a single value for a particular entity.

Example: Age is a single-valued attribute of a person.

**Multivalued Attributes** can have a set of values for the same entity.

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.

**Stored and Derived Attributes.** In some cases, two (or more) attribute values are related—for example, the Age and Birth\_date attributes of a person. For a particular person entity, the value of Age can be determined from the current date and the value of that person's Birth\_date.



The Age attribute is hence called a **derived** attribute and is said to be derivable from the Birth\_date attribute, which is called a **stored** attribute.

Some attribute values can be derived from related entities; for example, an attribute Number\_of\_employees of a DEPARTMENT entity can be derived by counting the number of employees related to (working for) that department.

**Complex Attributes.** The composite and multivalued attributes can be nested arbitrarily. 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.

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 can be specified. Both Phone and Address are themselves composite attributes.

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

**NULL Values.** A special value called NULL is used for situations where attribute is **not applicable or unknown**.

NULL can be used if

1. A particular entity **may not have an applicable value** for an attribute.

**Example,** Apartment\_number attribute of an address applies only to addresses that are in apartment buildings and not to other types of residences, such as single-family homes.

College\_degrees attribute applies only to people with college degrees.

2. We **do not know the value of an attribute** for a particular entity, example, if we do not know the home phone number of 'John Smith'. The unknown category of NULL can be further classified into two cases.
  - when it is **known that the attribute value exists but is missing**—for instance, if the Height attribute of a person is listed as NULL.
  - when it is **not known whether the attribute value exists**—for example, if the Home\_phone attribute of a person is NULL.

## 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.

A database usually contains groups of entities that are similar. For example, a company employing hundreds of employees and these employee entities share the same attributes, but each entity has its *own value(s)* for each attribute. Figure shows two entity types: EMPLOYEE and COMPANY, and a list of some of the attributes for each.

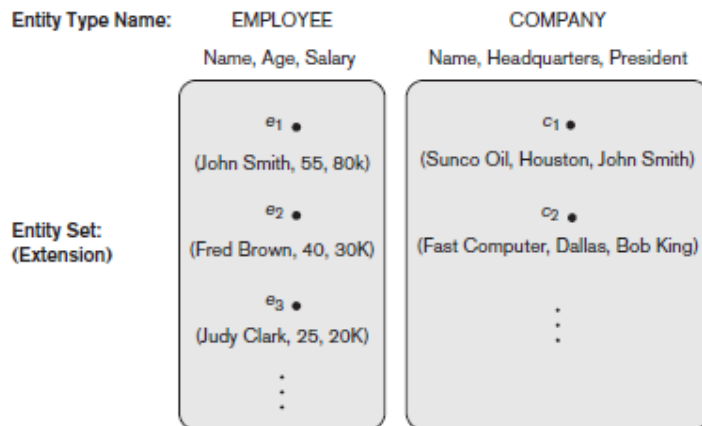
An entity type is represented in ER diagrams as a **rectangular box** enclosing the entity type name.

**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** or **entity collection**; 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 collection of *all employee entities* in the database.

An entity type describes the **schema** or **intension** for a *set of entities* that share the same structure.

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



Attribute names are enclosed in ovals and are attached to their entity type by straight lines. Composite attributes are attached to their component attributes by straight lines. Multivalued attributes are displayed in double ovals.

### Key Attributes of an Entity Type

An 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.

Example: For the STUDENT entity type, a typical key attribute is USN.

- Sometimes several attributes together form a key, that is *combination* of the attribute values must be distinct for each entity. If a set of attributes possesses this property, then define a *composite attribute* and designate it as a key attribute of the entity type.
- Notice that such a composite key must be *minimal*;
- Superfluous attributes must not be included in a key.
- In ER diagrammatic notation, each key attribute has its name **underlined** inside the oval.
- Uniqueness property must hold for *every entity set* of the entity type.
- This key constraint is derived from the constraints of the miniworld that the database represents.

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

There are several implicit relationships among the various entity types. In fact, whenever an attribute of one entity type refers to another entity type, some relationship exists. For example, the attribute Manager of DEPARTMENT refers to an employee who manages the department; the attribute Controlling\_department of PROJECT refers to the department that controls the project;

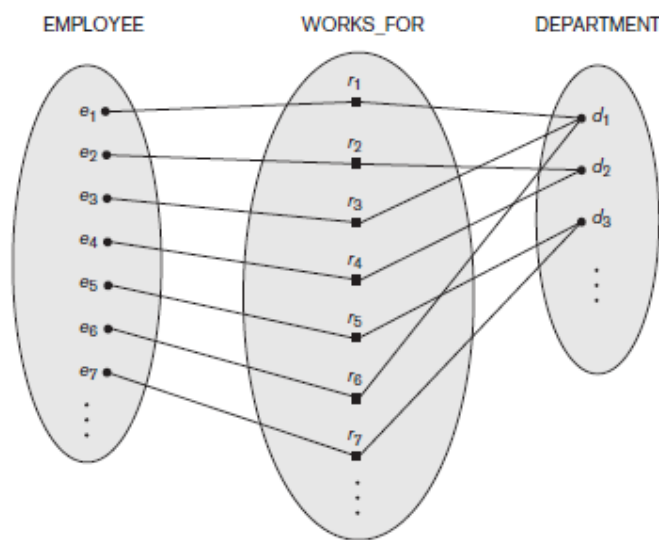
In the ER model, these references should not be represented as attributes but as relationships.

## Relationship Types

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.

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  $E_j$ ,  $1 \leq j \leq n$ .

- Informally, each relationship instance  $r_i$  in  $R$  is an association of entities, where the association includes exactly one entity from each participating entity type.
- For example, 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.
- 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



### 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 as shown in above figure.

A relationship type of degree two is called **binary**, and one of degree three is called **ternary**.

Relationships can generally be of any degree, but the ones most common are binary relationships.

### Relationships as Attributes

It is sometimes convenient to think of a binary relationship type in terms of attributes. Consider the **WORKS\_FOR** relationship type. One can think of an attribute called **Department** of the **EMPLOYEE** entity

type, where the value of Department for each EMPLOYEE entity is (a reference to) the DEPARTMENT entity for which that employee works. Hence, the value set for this Department attribute is the set of *all* DEPARTMENT entities, which is the DEPARTMENT entity set.

### 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.

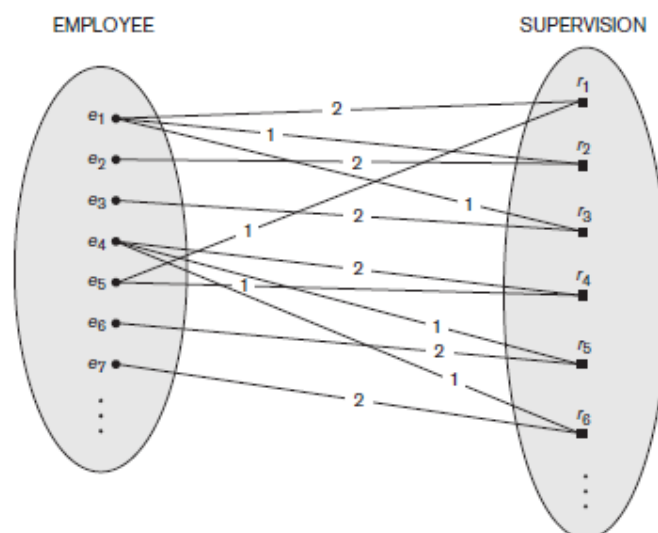
Example: In the WORKS\_FOR relationship type, EMPLOYEE plays the role of *employee* or *worker* and DEPARTMENT plays the role of *department* or *employer*.

### 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**.

Example: The SUPERVISION relationship type relates an employee to a supervisor, where both employee and supervisor entities are members of the same EMPLOYEE entity set.

Hence, the EMPLOYEE entity type *participates twice* in SUPERVISION: once in the role of *supervisor* (or *boss*), and once in the role of *supervisee* (or *subordinate*).



### 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.

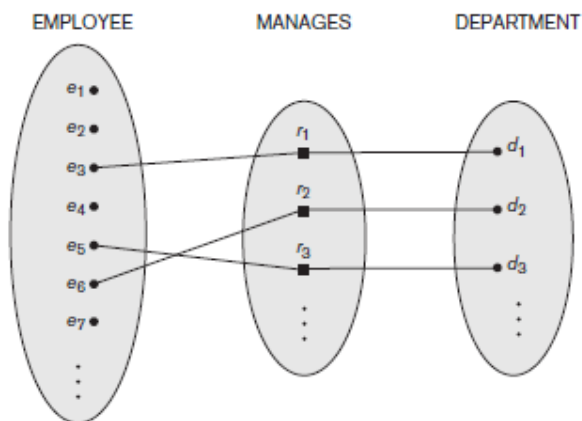
- These constraints are determined from the miniworld situation that the relationships represent.  
Example: if the company has a rule that each employee must work for exactly one department, then we would like to describe this constraint in the schema.
- Two main types of binary relationship constraints: **cardinality ratio** and **participation**.
- The cardinality ratio and participation constraints, taken together, as the **structural constraints** of a relationship type.

### Cardinality Ratios for Binary Relationships:

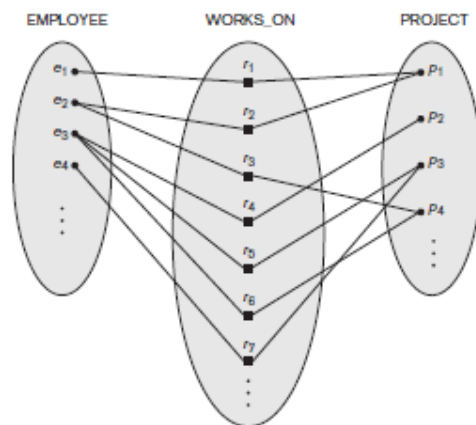
The **cardinality ratio** for a binary relationship specifies the *maximum* number of relationship instances that an entity can participate in. The possible cardinality ratios for binary relationship types are 1:1, 1:N, N:1, and M:N.

Example 1: In the WORKS\_FOR binary relationship type, DEPARTMENT:EMPLOYEE is of cardinality ratio 1:N, that is each department can have any number of employees (N), but an employee can work for at most one department (1).

Example 2: binary relationship MANAGES is of cardinality ratio 1:1, an employee can manage at most one department and a department can have at most one manager.



Example 3: The relationship type WORKS\_ON is of cardinality ratio M:N, because the miniworld 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. Also *maximum number* on participation can be specified, such as 4 or 5.

### 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. It specifies the *minimum* number of relationship instances that each entity can participate in and thus is also called the **minimum cardinality constraint**.

There are two types of participation constraints

**Total participation constraints:** If every instance of entity type E participate in at least one relationship instance *r*, the participation is said to be total. Total participation is also called **existence dependency**.

Example: 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.

**Partial participation constraints:** If only some of the instances of the entity type E participate in the relationship, the participation is said to be partial.

Example: Every employee is not expected 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**.

### 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.
- This other entity type is called the **identifying** or **owner entity type**, and the relationship type that relates a weak entity type to its owner is called 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*.
- The partial key attribute is underlined with a dashed or dotted line.

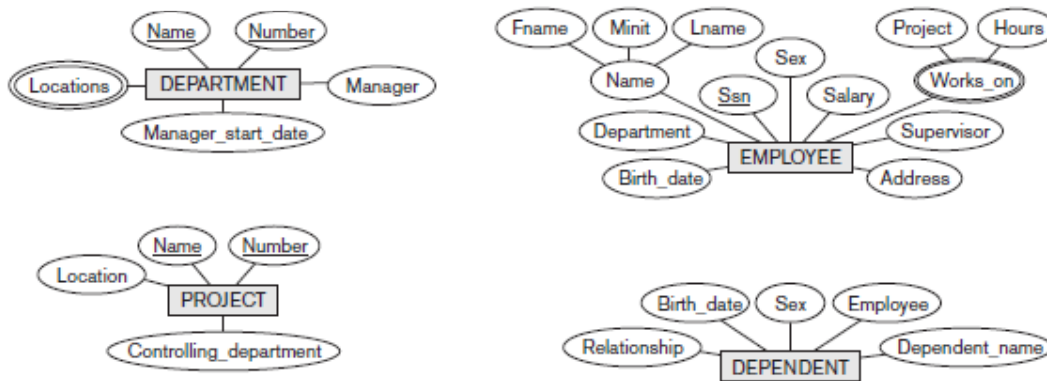
### A Sample Database Application: COMPANY DATABASE

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

- *The company is organized into departments. Each department has a unique name, a unique number, and a particular employee who manages the department. We keep track of the start date when that employee began managing the department. A department may have several locations.*
- *A department controls a number of projects, each of which has a unique name, a unique number, and a single location.*
- *The database will store each employee's name, Social Security number,2 address, salary, sex (gender), and birth date. An employee is assigned to one department, but may work on several projects, which are not necessarily controlled by the same department. It is required to keep track of the current number of hours per week that an employee works on each project, as well as the direct supervisor of each employee (who is another employee).*
- *The database will keep track of the dependents of each employee for insurance purposes, including each dependent's first name, sex, birth date, and relationship to the employee.*

### Initial Design of Entity Types for the COMPANY Database Schema

- Based on the requirements, we can identify four initial entity types in the COMPANY database: DEPARTMENT, PROJECT, EMPLOYEE, and DEPENDENT. Their initial attributes are derived from the requirements description. Initial design is shown in the figure



**Preliminary design of entity types for the COMPANY database. Some of the shown attributes will be refined into relationships.**

### **Refining the COMPANY database schema by introducing relationships**

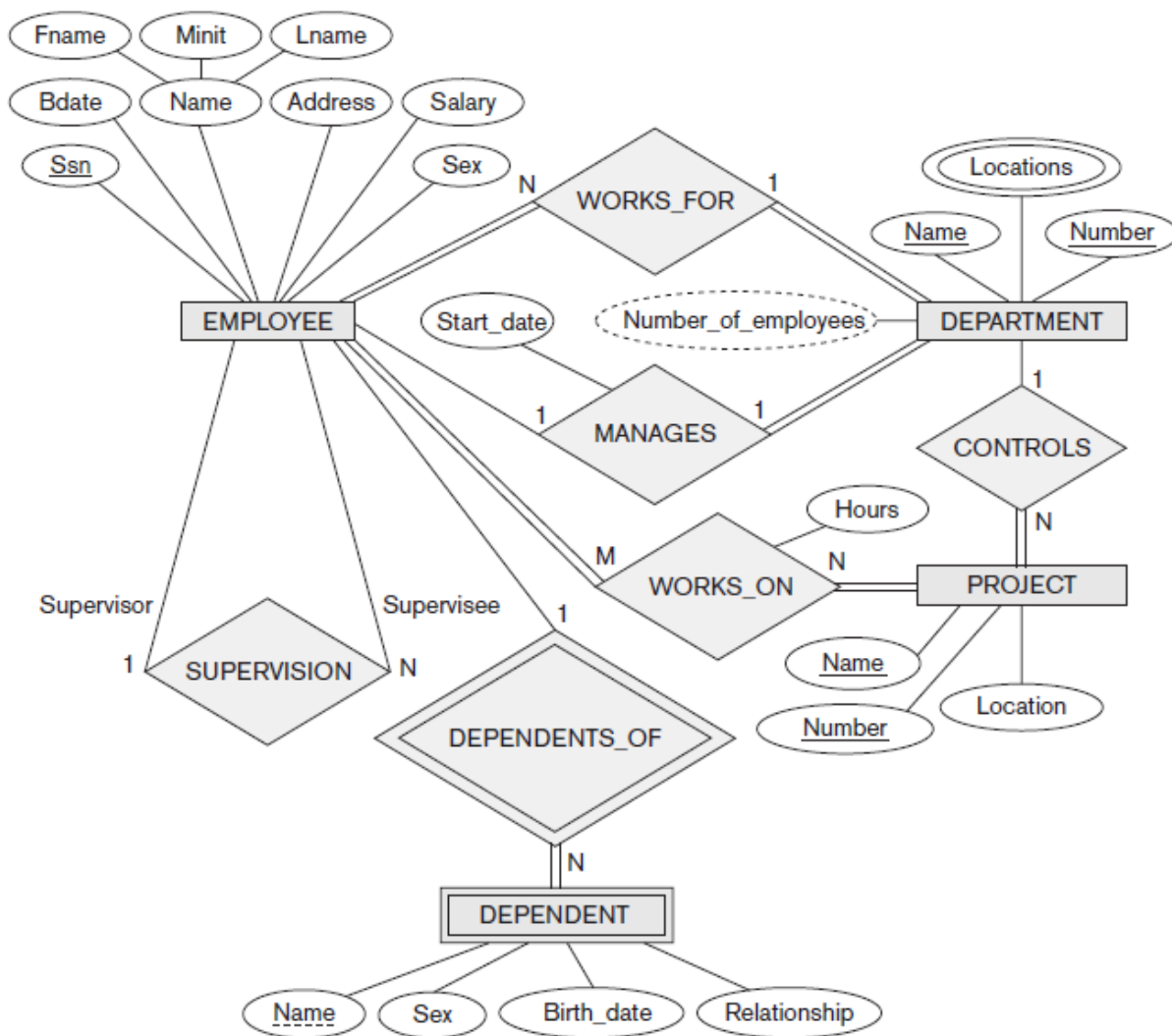
By refining the requirements, six relationship types are identified. All are binary relationships (degree 2). Listed below with their participating entity types:

- WORKS\_FOR (between EMPLOYEE, DEPARTMENT)
- MANAGES (also between EMPLOYEE, DEPARTMENT)
- CONTROLS (between DEPARTMENT, PROJECT)
- WORKS\_ON (between EMPLOYEE, PROJECT)
- SUPERVISION (between EMPLOYEE (as subordinate), EMPLOYEE (as supervisor))
- DEPENDENTS\_OF (between EMPLOYEE, DEPENDENT)

**In the refined design, some attributes from the initial entity types are refined into relationships:**

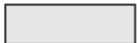
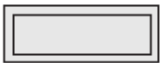


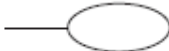
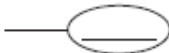

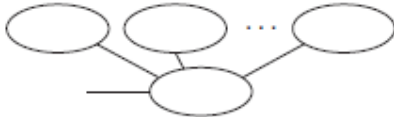
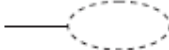
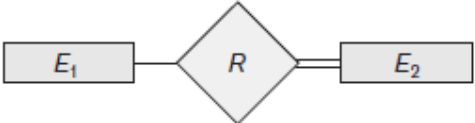

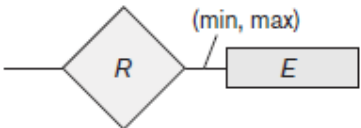
- Manager of DEPARTMENT -> MANAGES
- Works\_on of EMPLOYEE -> WORKS\_ON
- Department of EMPLOYEE -> WORKS\_FOR





An ER schema diagram for the COMPANY database.

## **Summary of the notation for ER diagrams**

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

## **Question Bank**

1. Define the following terms:  
*data, database, DBMS, database system, database catalog, program-data independence, user view, DBA, end user, canned transaction and meta-data.*
2. Discuss the main characteristics of the database approach and how it differs from traditional file systems.
3. What are the responsibilities of the DBA and the database designers?
4. What are the different types of database end users? Discuss the main activities of each.
5. Discuss the capabilities that should be provided by a DBMS.
6. Discuss the differences between database systems and File system.
7. Discuss the advantages and disadvantages of DBMS.
8. Discuss applications of database system.
9. Define the following terms:  
Data model, database schema, database state, internal schema, conceptual schema, external schema, data independence, DDL, DML, SDL, VDL, query language, host language, data sublanguage.
10. Discuss the main categories of data models.
11. What is the difference between a database schema and a database state?
12. Describe the three-schema architecture. Why do we need mappings among schema levels?
13. What is the difference between logical data independence and physical data independence? Which one is harder to achieve? Why?
14. Explain database Languages.
15. Discuss the different types of user-friendly interfaces and the types of users who typically use each.
16. Explain the important component modules of a DBMS and their interactions with diagram.
17. Discuss database system architectures. What is the difference between the two-tier and three-tier client/server architectures?
18. Discuss the use of high-level conceptual data model in the database design process.
19. List the various cases where use of a NULL value would be appropriate.
20. Define the following terms:  
Entity, attribute, attribute value, relationship instance, composite attribute, multivalued attribute, derived attribute, complex attribute, key attribute, and domain
21. What is an entity type? What is an entity set? Explain the differences among an entity, an entity type, and an entity set.
22. What is a relationship type? Explain the differences among a relationship type, and a relationship set.
23. What is a role name? When is it necessary to use role names in the description of relationship types?
24. Describe the two alternatives for specifying structural constraints on relationship types. What are the advantages and disadvantages of each?
25. Under what conditions can an attribute of a binary relationship type be migrated to become an attribute of one of the participating entity types?
26. What is meant by a recursive relationship type? Give some examples of recursive relationship types.

27. When is the concept of a weak entity used in data modeling? Define the terms owner entity type, weak entity type, identifying relationship type, and partial key.
28. Can an identifying relationship of a weak entity type be of a degree greater than two? Give examples to illustrate your answer.
29. Discuss the naming conventions used for ER schema diagrams.