Introduction

Chapter-1

- A **database-management system (DBMS)** is a collection of programs that enables users to create and maintain a database. It facilitates the processes of defining, constructing, manipulating, and sharing databases among various users and applications.
- The collection of data, usually referred to as the **database**, contains information relevant to an enterprise.
- ❏ The primary goal of a DBMS is to provide a way to store and retrieve database information that is both *convenient* and *efficient*.
- ❏ Database systems are designed to manage large bodies of information. Management of data involves both defining structures for storage of information and providing mechanisms for the manipulation of information.
- ❏ In addition, the database system must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access. If data are to be shared among several users, the system must avoid possible anomalous results.

Defining a database:

- It involves specifying the data types, structures, and constraints of the data to be stored in the database.
- The database definition or descriptive information is also stored by the DBMS in the form of a database catalog or dictionary; it is called meta-data.

Constructing the database: 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

Sharing a database: allows multiple users and programs to access the database simultaneously.

1.1 Database-System Applications

Databases are widely used. Here are some representative applications:

• **Enterprise Information**

◦ Sales: For customer, product, and purchase information.

◦ Accounting: For payments, receipts, account balances and other accounting information.

◦ Human resources: For information about employees, salaries.

◦ Manufacturing: For management of the supply chain and for tracking production of items in factories.

◦ Online retailers: For sales data and online order tracking.

• **Banking and Finance**

◦ Banking: For customer information, accounts, loans, and banking transactions.

◦ Credit card transactions: For purchases on credit cards and generation of monthly statements.

◦ Finance: For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds.

• **Universities:** For student information, course registrations, and grades and other information.

• **Airlines:** For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner.

• **Telecommunication:** For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.

1.2 Characteristics of Database Approach

- In the database approach, a single repository maintains data that is defined once and then accessed by various users whereas In traditional file processing, each user defines and implements the files needed for a specific software application as part of programming the application.
- The main characteristics of the database approach versus the file-processing approach are the following:
  - Self-describing nature of a database system
    - ❖ A 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.
  - 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 catalogue separately from the access programs.
    - ❖ This property is called **program-data independence.**

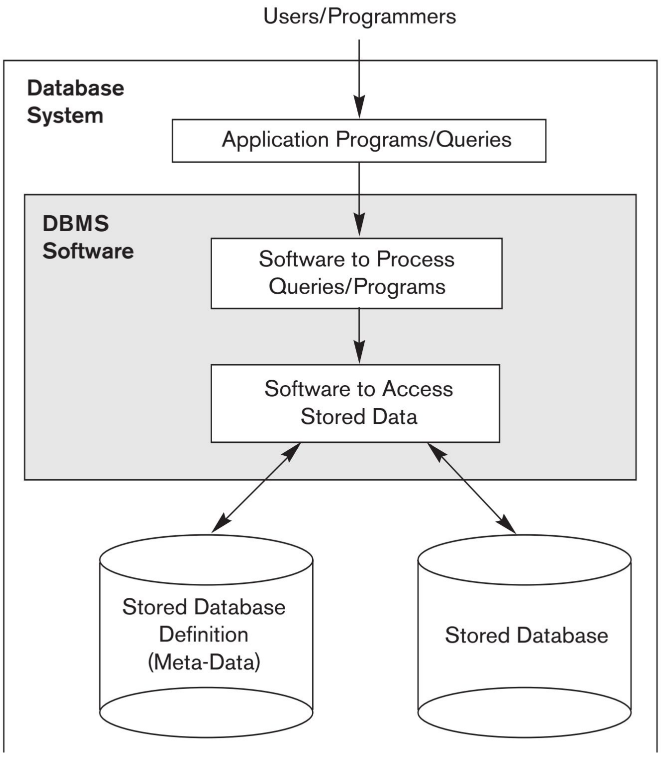
    - ❖ The implementation (or method) of the operation is specified separately and can be changed without affecting the interface.
    - ❖ User application programs can operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented.
    - ❖ It is known as **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

- Sharing of data and multi user transaction processing

  - A multi user 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.


Users/Programmers
Database System
Application Programs/Queries
DBMS Software
Software to Process Queries/Programs
Software to Access Stored Data
Stored Database Definition (Meta-Data)
Stored Database

1.3 Purpose of Database Systems

The typical **file-processing system** is supported by a conventional operating system. The system stores permanent records in various files, and it needs different application programs to extract records from, and add records to, the appropriate files. Before database management systems (DBMSs) were introduced, organizations usually stored information in such systems. Keeping organizational information in a file-processing system has a number of major *disadvantages*:

- **Data redundancy and inconsistency** The same information may be duplicated in several places (files). This *redundancy* leads to higher storage and access cost. In addition, it may lead to *data inconsistency*; that is, the various copies of the same data may no longer agree. For example, a changed student address may be reflected in the Music department records but not elsewhere in the system.

- **Difficulty in accessing data** Suppose that one of the university clerks needs to find out the names of all students who live within a particular postal-code area. Because there is no application program on hand to meet it, the university clerk has now two choices: either obtain the list of all students and extract the needed information manually or ask a programmer to write the necessary application program. Both alternatives are obviously unsatisfactory.
  The point here is that conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner. More responsive data-retrieval systems are required for general use.

- **Data isolation** Because data is scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

- **Integrity problems** The data values stored in the database must satisfy certain types of *consistency constraints*. Suppose the university maintains an account for each department, and records the balance amount in each account. Suppose also that the university requires that the account balance of a department may never fall below zero. Developers enforce these constraints in the system by adding appropriate code in the various application programs. However, when new constraints are added, it is difficult to change the programs to enforce them.

- **Atomicity problems** A computer system, like any other device, is subject to failure. Consider a program to transfer $500 from the account balance of department A to the account balance of department B. If a system failure occurs during the execution of the program, it is possible that the $500 was removed from the balance of department A but was not credited to the balance of department B, resulting in an inconsistent database state. Clearly, it is essential to database consistency that either both the credit and debit occur, or that neither occur. That is, the funds transfer must be atomic—it must happen in its entirety or not at all. It is difficult to ensure atomicity in a conventional file-processing system.

- **Concurrent-access anomalies**
- **Security problems**

1.4 View of Data

A major purpose of a database system is to provide users with an abstract view of the data. That is, the system hides certain details of how the data are stored and maintained.

1.4.1 Data Abstraction

For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database. Since many database-system users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify users' interactions with the system:

- **Physical level** The lowest level of abstraction describes how the data are actually stored. The physical level describes complex low-level data structures in detail.

- **Logical level** The next-higher level of abstraction describes what data are stored in the database, and what relationships exist among those data. The logical level thus describes the entire database in terms of a small number of relatively simple structures. Although implementation of the simple structures at the logical level may involve complex physical-level structures, the user of the logical level does not need to be aware of this complexity. This is referred to as **physical data independence**. Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.

- **View level** The highest level of abstraction describes only part of the entire database. Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database. Many users of the database system do not need all this information; instead, they need to access only a part of the database. The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database.

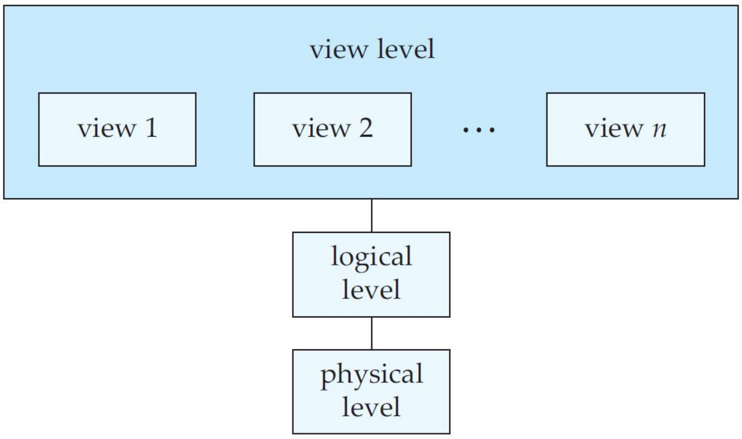


Figure 1.1 The three levels of data abstraction.

Programmers using a programming language work at a logical level of abstraction. Similarly, database administrators usually work at this level of abstraction.

1.4.2 Data Models

Data model: a collection of concepts that can be used to describe the structure of a database to achieve the data abstraction

The structure of a database includes data types, relationships, and constraints that apply to the data.

- **Categories of Data Models**
  - High Level or Conceptual Data Model
    - close to the way many users perceive data
    - Use concepts such as entities, attributes, and relationships.
    - Entity: An entity represents a real-world object or concept described in the database
    - Attribute: An attribute represents some property of interest that further describes an entity
    - Relationship: A relationship among two or more entities represents an association among the entities
    - Entity-Relationship (ER) model: a popular high-level conceptual data model
  - Representational (or implementation) data models
    - Provide concepts that may be easily understood by end users
    - Hides many details of data storage on disk but can be implemented on a computer system directly.
    - 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**.
    - Object Data Management Group (ODMG) Object data model is a higher-level implementation data model that is closer to conceptual data models.
  - Low Level or Physical Data Model
    - Describes the details of how data is stored on the computer storage media
    - Describe how data is stored as files in the computer by representing information such as record formats, record orderings, and access paths
    - Access path: is a structure that makes the search for particular database records efficient.
    - Index: index is an example of an access path that allows direct access to data using an index term or a keyword

1.4.3 Instances and Schemas

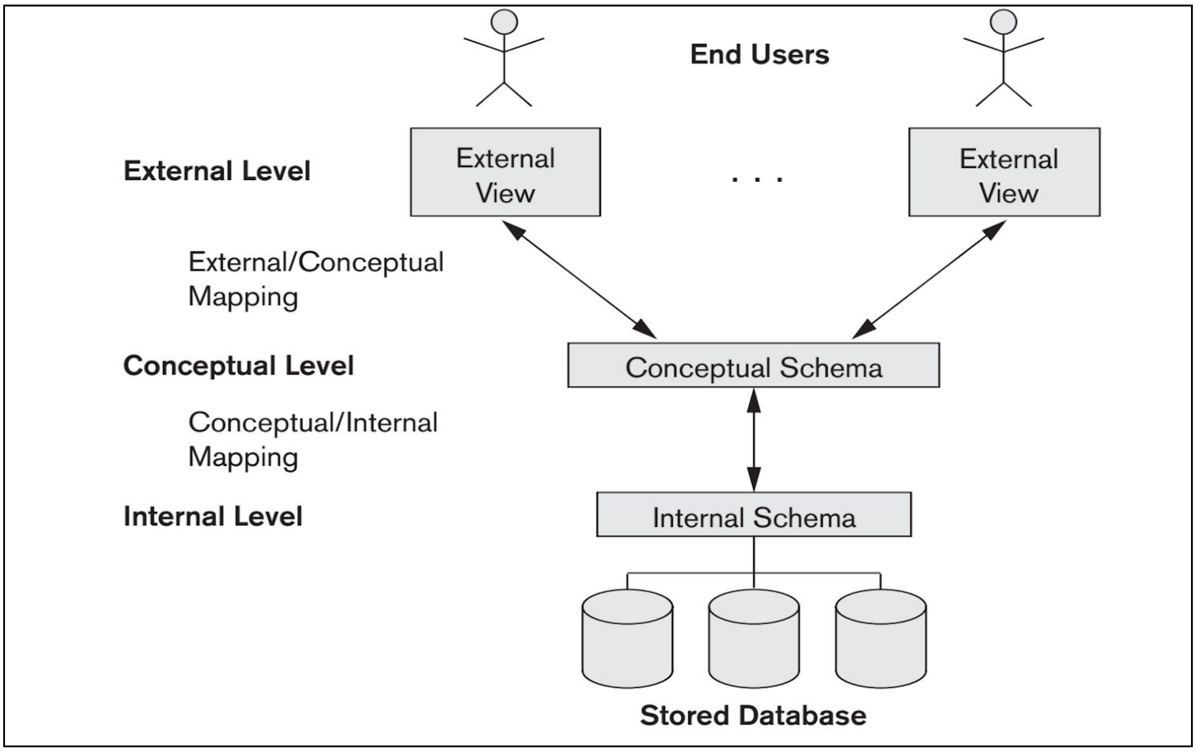
Database State: The data in the database at a particular moment in time is called a database state or snapshot. It is also called the current set of occurrences or **instances** in the database

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

The schema is sometimes called the *intension*, and a database state is called an *extension* of the schema.

- **Three Schema Architecture**

  - The **internal level** has an internal schema, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.
  - The **conceptual level** has a conceptual schema, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints.
  - 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.



The processes of transforming requests and results between levels are called mappings.

- **Data Independence**

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

1.5 Database Languages

A database system provides a **data-definition language** to specify the database schema and a **data-manipulation language** to express database queries and updates. In practice, the data-definition and data-manipulation languages are not two separate languages; instead they simply form parts of a single database language, such as the widely used SQL language.

1.5.1 Data-Manipulation Language

A data-manipulation language (DML) is a language that enables users to access or manipulate data as organized by the appropriate data model. The types of access are:

- Retrieval of information stored in the database
- Insertion of new information into the database
- Deletion of information from the database
- Modification of information stored in the database

There are basically two types:

- **Procedural DMLs** require a user to specify what data are needed and how to get those data.
- **Declarative DMLs** (also referred to as nonprocedural DMLs) require a user to specify what data are needed without specifying how to get those data.

Declarative DMLs are usually easier to learn and use than are procedural DMLs. However, since a user does not have to specify how to get the data, the database system has to figure out an efficient means of accessing data. A **query** is a statement requesting the retrieval of information. The portion of a DML that involves information retrieval is called a query language. Although technically incorrect, it is common practice to use the terms query language and data-manipulation language synonymously.

Highlevel DMLs
can retrieve many records in a single DML statement; therefore, they are called set-at-a-time or set-oriented DMLs

Low-level DMLs are also called record-at-a-time typically retrieves individual records or objects from the database and processes each separately.

Note: 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**.

1.5.2 Data-Definition Language

We specify a database schema by a set of definitions expressed by a special language called a data-definition language (DDL). The DDL is also used to specify additional properties of the data.
We specify the storage structure and access methods used by the database system by a set of statements in a special type of DDL called a data storage and definition language. These statements define the implementation details of the database schemas, which are usually hidden from the users.

- Used by the DBA and by database designers to define conceptual and internal schemas
- DDL compiler is used to process DDL statements in order to identify descriptions of the schema constructs and to store the schema description in the DBMS catalog

1.5.3 Storage Definition Language (SDL)

- Used to specify the internal schema
- DDL is used to specify the conceptual schema

1.5.4 View definition language (VDL)

- Needed to specify user views and their mappings to the conceptual schema
- In RDBMS, SQL is used in the role of VDL to define user or application views as results of predefined queries

The data values stored in the database must satisfy certain *consistency constraints*:

1.Domain Constraints A domain of possible values must be associated with every attribute (for example, integer types, character types, date/time types). Declaring an attribute to be of a particular domain acts as a constraint on the values that it can take.

2.Referential Integrity There are cases where we wish to ensure that a value that appears in one relation for a given set of attributes also appears in a certain set of attributes in another relation (referential integrity).

3.Assertions An assertion is any condition that the database must always satisfied. Domain constraints and referential-integrity constraints are special forms of assertions. When an assertion is created, the system tests it for validity. If the assertion is valid, then any future modification to the database is allowed only if it does not cause that assertion to be violated.

4.Authorization We may want to differentiate among the users as far as the type of access they are permitted on various data values in the database. These differentiations are expressed in terms of authorization, the most common being: read authorization, which allows reading, but not modification, of data; insert authorization, which allows insertion of new data, but not modification of existing data; update authorization, which allows modification, but not deletion, of data; and delete authorization, which allows deletion of data. We may assign the user all, none, or a combination of these types of authorization.

The DDL, just like any other programming language, gets as input some instructions (statements) and generates some output. The output of the DDL is placed in the **data dictionary**,which contains **metadata**—that is, data about data. The data dictionary is

considered to be a special type of table that can only be accessed and updated by the database system itself (not a regular user). The database system consults the data dictionary before reading or modifying actual
data.

1.6 Database Interface

- Menu-Based Interfaces for Web Clients or Browsing
- Forms-Based Interfaces
- Graphical User Interfaces:
    - A GUI typically displays a schema to the user in diagrammatic form. The user then can specify a query by manipulating the diagram
- Natural Language Interfaces:
    - These interfaces accept requests written in English or some other language and attempt to understand them.
- Speech Input and Output
    - Limited use of speech as an input query and speech as an answer to a question or result of a request is becoming commonplace
    - Applications with limited vocabularies such as inquiries for telephone directory, flight arrival/departure, and credit card account information are allowing speech for input and output to enable customers to access this information.
- Interfaces for Parametric Users.
    - Parametric users, such as bank tellers, often have a small set of operations that they must perform repeatedly.
    - function keys in a terminal can be programmed to initiate various commands which reduces the number of keystrokes for parametric users
- Interfaces for the DBA.
    - 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

1.7 Database Architecture

We are now in a position to provide a single picture (Figure 1.5) of the various components of a database system and the connections among them.

The architecture of a database system is greatly influenced by the underlying computer system on which the database system runs. Database systems can be centralized, or client-server, where one server machine executes work on behalf of multiple client machines. Database systems can also be designed to exploit parallel computer architectures. Distributed databases span multiple geographically separated machines.

Most users of a database system today are not present at the site of the database system, but connect to it through a network. We can therefore differentiate between **client** machines, on which remote database users work, and **server** machines, on which the database system runs.

Database applications are usually partitioned into two or three parts, as in Figure 1.6. In a **two-tier architecture**, the application resides at the client machine, where it invokes database system functionality at the server machine through query language statements.

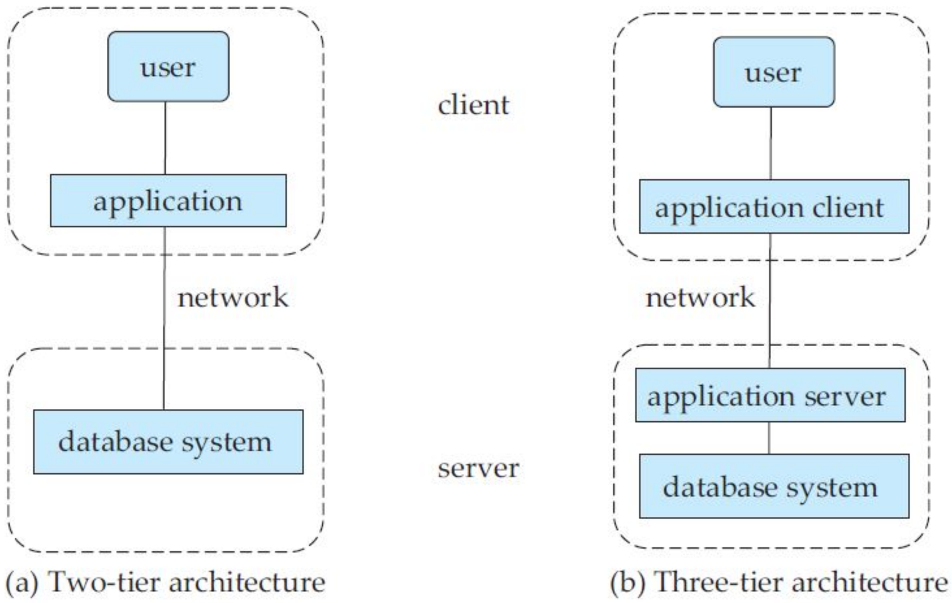


Figure 1.6 Two-tier and three-tier architectures.

Application program interface standards like ODBC and JDBC are used for interaction between the client and the server.

In contrast, in a **three-tier architecture**, the client machine acts as merely a front end and does not contain any direct database calls. Instead, the client end communicates with an application server, usually through a forms interface. The **application server** in turn communicates with a database system to access data. The **business logic** of the application, which says what actions to carry out under what conditions, is embedded in the application server, instead of being distributed across multiple clients. Three-tier applications are more appropriate for large applications, and for applications that run on the WorldWideWeb.

1.7.1 Database Access from Application Programs

SQL is not as powerful as a universal Turing machine; that is, there are some computations that are possible using a general-purpose programming language but are not possible using SQL. SQL also does not support actions such as input from users, output to displays, or communication over the network. Such computations and actions must be written in a host language, such as C, C++, or Java, with embedded SQL queries that access the data in the database. Application programs are programs that are used to interact with the database in this fashion.

1.7.2 Client/Server Architecture

- A client module is typically designed so that it will run on a user workstation or personal computer
- A server module, typically handles data storage, access, search, and other functions.

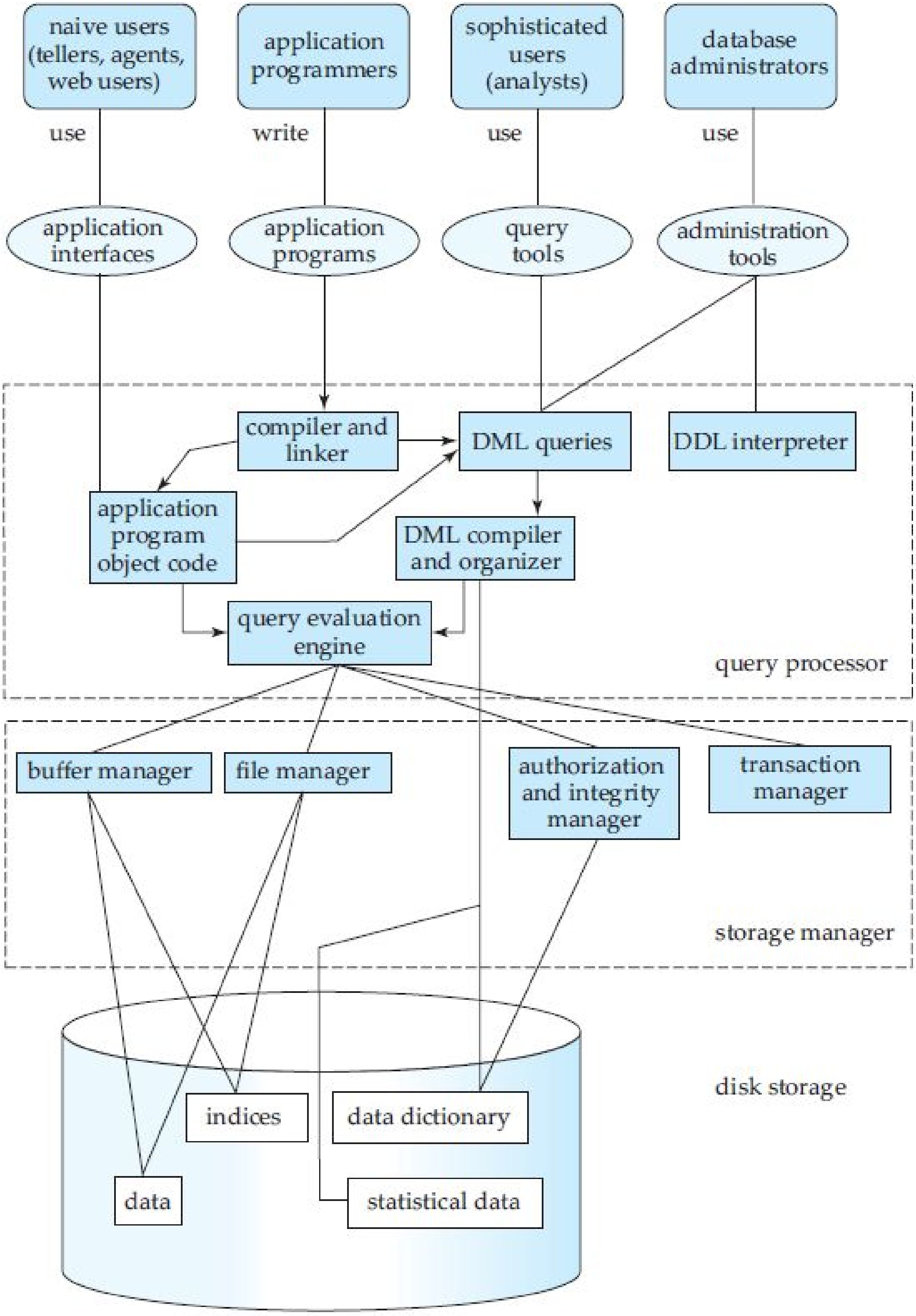


Figure 1.5 System structure.

1.7 Database Users and Administrators

A primary goal of a database system is to retrieve information from and store new information into the database. People who work with a database can be categorized as *database users* or *database administrators*.

There are different types of database-system users, differentiated by the way they expect to interact with the system. Different types of user interfaces have been designed for the different types of users.

- **Na¨ıve users/Parametric End Users** are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously. For example, a clerk in the university who needs to add a new instructor to department A invokes a program called new hire. This program asks the clerk for the name of the new instructor, her new ID, the name of the department (that is, A), and the salary.
  The typical user interface for na¨ıve users is a forms interface, where the user can fill in appropriate fields of the form. Na¨ıve users may also simply read reports generated from the database.
- **Application programmers** are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces. Rapid application development (RAD) tools are tools that enable an application programmer to construct forms and reports with minimal programming effort.
- **Sophisticated users** interact with the system without writing programs. Instead, they form their requests either using a database query language or by using tools such as data analysis software. Analysts who submit queries to explore data in the database fall in this category.
- **Specialized users** are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework. Among these applications are computer-aided design systems, knowledgebase and expert systems, systems that store data with complex data types (for example, graphics data and audio data), and environment-modeling systems.
- **Casual Users / Temporary Users** Casual Users are the users who occasionally use/access the database but each time when they access the database they require the new information, for example, Middle or higher level manager.

1.7.2 Database Administrator

One of the main reasons for using DBMSs is to have central control of both the data and the programs that access those data. A person who has such central control over the system is called a database administrator (DBA). The functions of a DBA include:

• **Schema definition** The DBA creates the original database schema by executing a set of data definition statements in the DDL.
• **Storage structure and access-method definition**
• **Schema and physical-organization modification** TheDBAcarries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical organization to improve performance.

• **Granting of authorization for data access** By granting different types of authorization, the database administrator can regulate which parts of the database various users can access. The authorization information is kept in a special system structure that the database system consults whenever someone attempts to access the data in the system.
• **Routine maintenance** Examples of the database administrator’s routine
maintenance activities are:
◦ Periodically backing up the database, either onto tapes or onto remote servers, to prevent loss of data in case of disasters such as flooding.
◦ Ensuring that enough free disk space is available for normal operations, and upgrading disk space as required.
◦ Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.

1.7.3 Users: Database Designers

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

1.8 DBMS: Varieties

- Traditional Database applications - most of the information that is stored and accessed is either textual or numeric.
- Multimedia Databases - can store images, audio clips, and video streams digitally
- Geographic information systems (GIS) - can store and analyze maps, weather data, and satellite images
- Data warehouses and online analytical processing (OLAP) systems - To extract and analyze useful business information from very large databases to support decision making.
- Real-time and active database technology - Used to control industrial and manufacturing processes

1.9 DBMS Advantages

- Controlled Redundancy
- Restricting Unauthorized Access
- Persistent Storage for Program Objects
- Storage Structures and Search techniques for Efficient Query Processing
- Backup and Recovery
- Multiple User Interfaces
- Representing Complex Relationships among Data

1.10 DBMS Disadvantages

- 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 function

1.11 Database Applications: History

- Hierarchical and Network Systems
  - Hierarchical model organizes data in a tree-like structure while network model organizes data in a graph structure and relational database model organizes data in tables.
  - Hierarchical model represents one to many relationship whereas network model represents many to many relationships
  - Hierarchical model uses parent child relationships whereas in Network model multiple records can be linked to the same owner file.

- Object Oriented Databases (OODB)
  - Store and share complex, structured object
- Interchanging Data on the Web for E-Commerce Using XML
  - Extended Markup Language (XML) is considered to be the primary standard for interchanging data among various types of databases and Web pages.
- Extending Database Capabilities for New Applications
  - Storing large amounts of data resulting from scientific experiments
  - Storage and retrieval of images
  - Storage and retrieval of videos,
  - Data mining applications that analyze large amounts of data
  - Spatial applications that store spatial locations of data
  - Time series applications that store information such as economic data at
  - regular points in time

Notes:

1. Types of relationships – One-One, Many-One, One-Many, Many-Many
2. SDLC(Software Development Life Cycle) – Analysis(understanding requirements), Design(includes database design), Coding, Testing.

Entity Relationship Model

Chapter-2

2.1 Entity

- An **entity** is 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.

2.2 Attributes

The particular properties that describe an entity.
For example, an EMPLOYEE entity may be described by the employee’s name, age, address, salary, and job.

Types of attributes:

- Simple versus composite,
- Single valued versus multivalued,
- stored versus derived

2.2.1 Composite versus Simple (Atomic) Attributes

- Composite attributes can be divided into smaller subparts, which represent more basic attributes with independent meanings.
- Attributes that are not divisible are called simple or atomic attributes.
- Composite attributes can form a hierarchy
- The value of a composite attribute is the concatenation of the values of its component simple attributes.

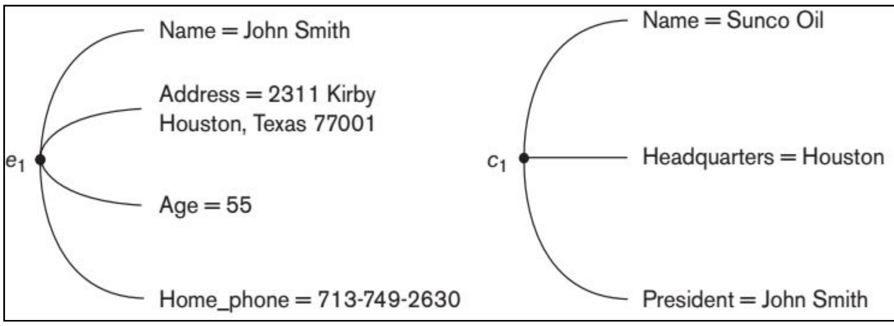


Figure 7.3
Two entities, EMPLOYEE e_1, and COMPANY c_1, and their attributes.

2.2.2 Single-Valued versus Multivalued Attributes

- Most attributes have a single value for a particular entity; such attributes are called single-valued

- If an attribute can have a set of values for the same entity, it is known as multivalued attributes
- A multivalued attribute may have lower and upper bounds to constrain the number of values allowed for each individual entity.

2.2.3 Stored versus Derived Attributes.

- Some attribute values can be derived from related entities; those are known as derived attributes
- For a particular person entity, the value of Age can be determined from the current (today's) date and the value of that person's Birth_date. The Age attribute is hence called a derived attribute and is said to be derivable from the Birth_date attribute, which is called a stored attribute.

2.2.4 Complex Attributes

- Composite and multivalued attributes can be nested arbitrarily
- It can be represented as the grouping components of a composite attribute between parentheses () and separating the components with commas, and by displaying multivalued attributes between braces { }
- For example, if a person can have more than one residence and each residence can have a single address and multiple phones can be represented as:
  {Address_phone(
  {Phone(Area_code,Phone_number)},Address(Street_address(Number,Street,Apartment_number),City,State,Zip))}

2.3 Entity Type

- Defines a collection (or set) of entities that have the same attributes
- Described by its name and attributes.
- 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
- 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*.

2.4 Representation

- An entity type is represented in ER diagrams as a rectangular box by enclosing the entity type name.
- Attribute names are enclosed in ovals and are attached to their entity type by straight lines.
- Composite attributes are attached to their component attributes by straight lines.
- Multivalued attributes are displayed in double ovals.
- Each key attribute has its name underlined inside the oval.

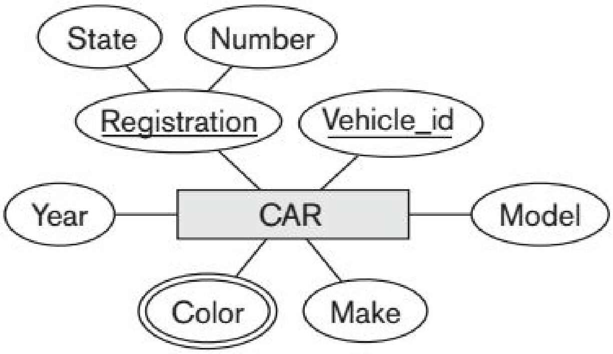
Notes-

- ➔ ENTITY is generally like **objects** of OOPs.
- ➔ ENTITY TYPE(represented by rectangle) is generally like a **class** of OOPs. Also called entity set.
- ➔ 1st design of DBMS is the designing of ER model
- ➔ Prepare Attributes for MCQs
  - ◆ Attributes representation in ER diagram– simple, single, stored are represented by **single oval**; composite is represented by **hierarchical oval**; derived is by **dotted-oval** and multi-valued by double oval.
  - ◆ In SCHEMA multivalued is represented by {} and component of composite attribute by ().
  - ◆ Key attributes(which are unique and can't be left blank) are represented by **underline**.
  - ◆ A single attribute cannot be attached to any other attribute but only with an entity.
  - ◆ SSN(Social Security no.) as an Aadhar card in India which is unique no.

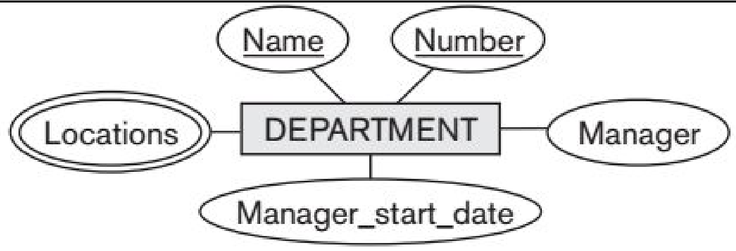
- ➔ Entity are of two types:
  - ◆ Strong Entity – having key attributes.
  - ◆ Weak Entity – no key attribute.
- ➔ Recursive relationship is important for short ques/ans
- ➔ No entity type can exist without attribute.
- ➔ To build a good ER Diagram we should ask us following questions:
  - ◆ What is the type of entity (weak or strong)?
  - ◆ What are the types of attributes?
  - ◆ Which one is the key attribute?
  - ◆ What is the type of relationship (1-1, many-1, 1-many, many-many)
  - ◆ Participation is total or partial?

2.5 Examples

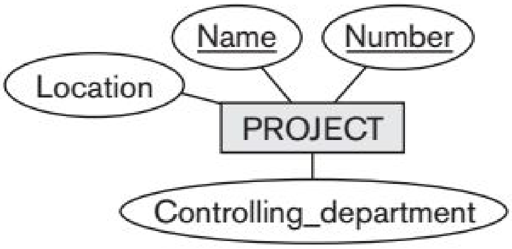
1. CAR - Entity



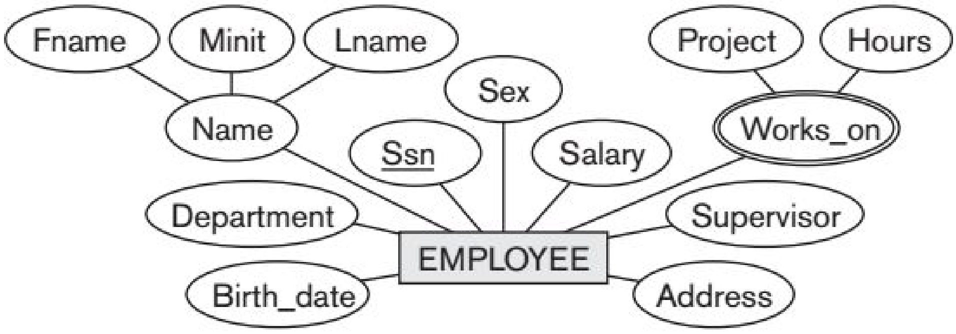
2. An entity type DEPARTMENT with attributes Name, Number, Locations, Manager, and Manager_start_date. Locations is the only multivalued attribute.
   We can specify that both Name and Number are (separate) key attributes because each was specified to be unique.



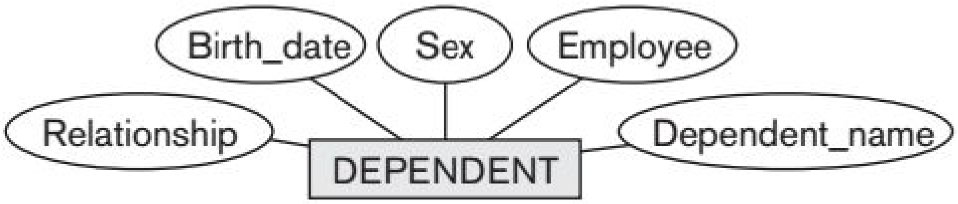
3. An entity type PROJECT with attributes Name, Number, Location, and Controlling_department. Both Name and Number are (separate) key attributes



4. An entity type EMPLOYEE with attributes Name, Ssn, Sex, Address, Salary, Birth_date, Department, and Supervisor.
   Both Name and Address may be composite attributes; however, this was not specified in the requirements.
   We must go back to the users to see if any of them will refer to the individual components of Name—First_name, Middle_initial, Last_name—or of Address.

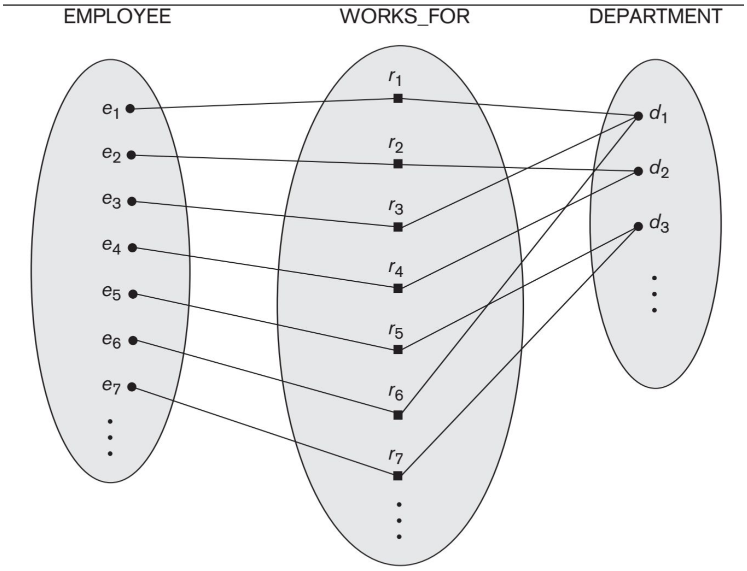


5. An entity type DEPENDENT with attributes Employee, Dependent_name, Sex, Birth_date, and Relationship (to the employee).



2.6 Relationship Types, Sets, and Instances

A relationship type R among n entity types E 1 , E 2 , ..., E n defines a set of associations—or a relationship set—among entities from these entity types.

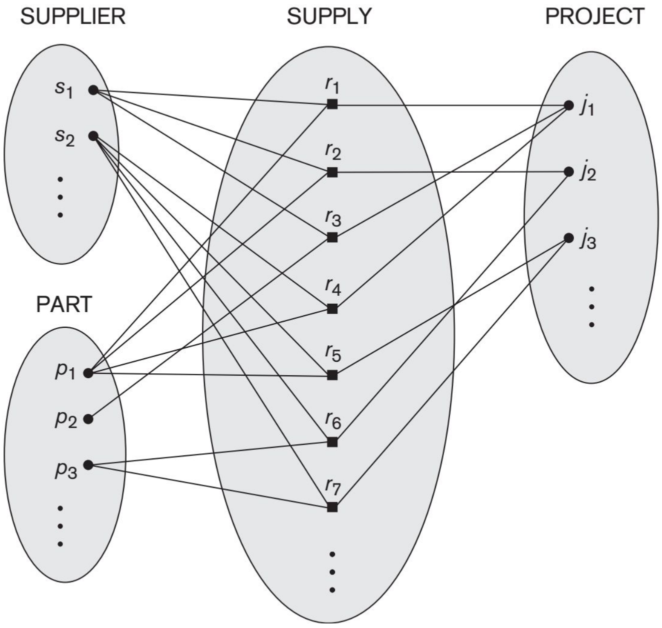


Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

2.6.1 Relationship Degree

- Degree of a Relationship Type
  - The degree of a relationship type is the number of participating entity types.
  - WORKS_FOR relationship is of degree two.
- A relationship type of degree two is called binary, and one of degree three is called ternary.

Ternary Relationship



Some relationship instances in the SUPPLY ternary relationship set.

2.6.2 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 helps to explain what the relationship means.
- In WORKS_FOR relationship type, EMPLOYEE plays the role of employee or worker and DEPARTMENT plays the role of department or employer.

2.6.3 Recursive Relationship

- If 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.
- Eg: 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).

2. 7 Constraints on Binary Relationship Types

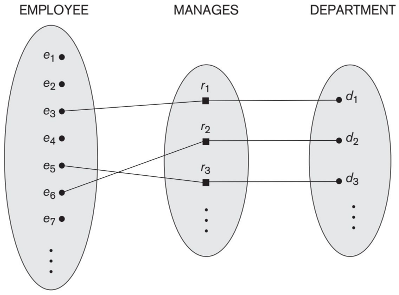
- Relationship types usually have certain constraints that limit the possible combinations of entities that may participate in the corresponding relationship set.
- Eg: if the company has a rule that each employee must work for exactly one department, it can be represented as shown in Works_for relationship

2.8 Cardinality Ratios for Binary Relationships

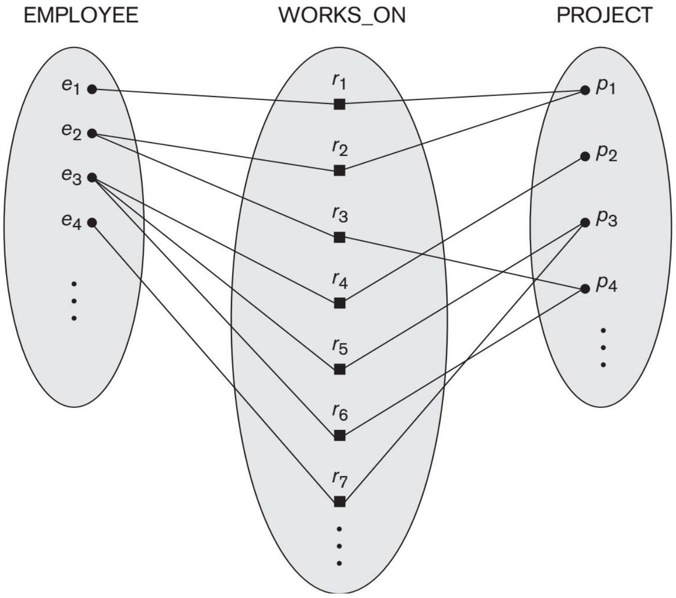
- The cardinality ratio for a binary relationship specifies the maximum number of relationship instances that an entity can participate in.
- Eg: WORKS_FOR binary relationship type, DEPARTMENT:EMPLOYEE is of cardinality ratio 1:N, meaning that each department can be related to (that is, employs) any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

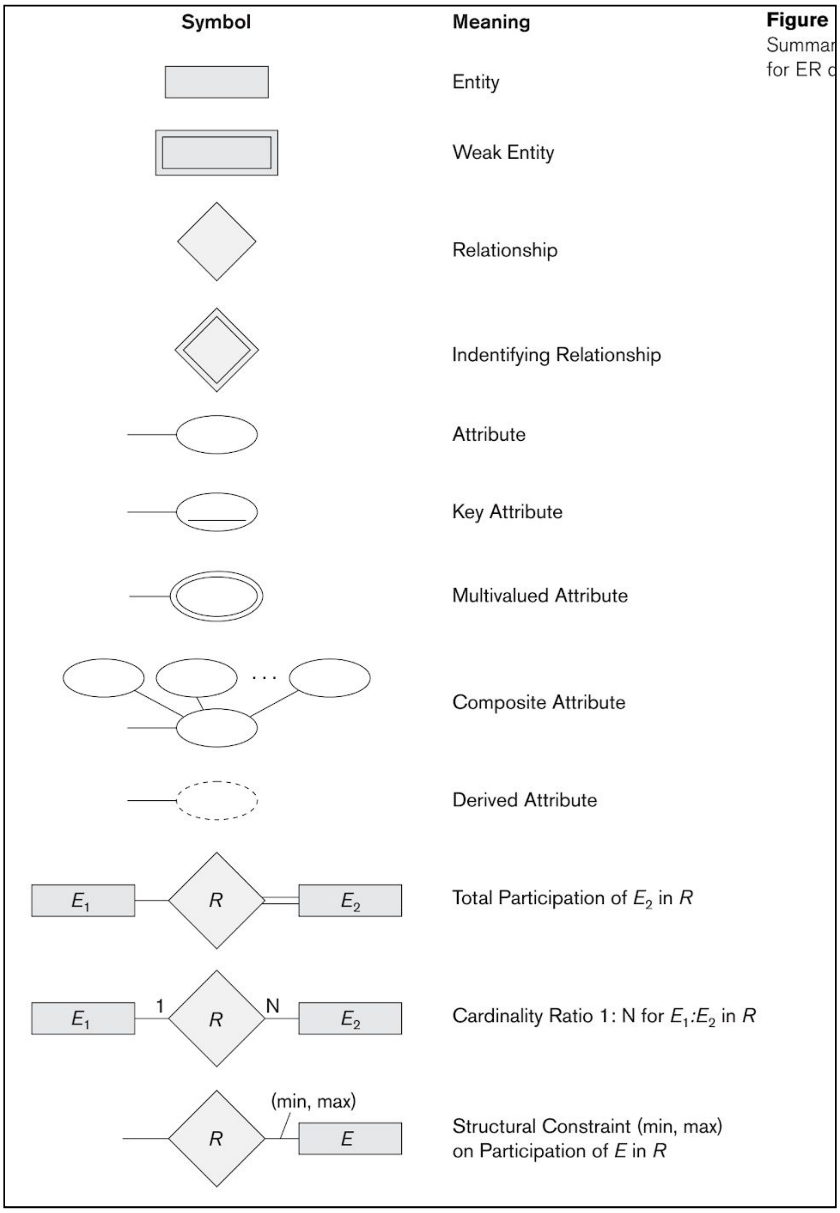
2.8.1 1:1 Relationships

Eg: An employee can manage one department only and a department can have one manager only.



2.8.2 M:N Relationships





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

Figure
Summar
for ER d