21CS53: DATABASE MANAGEMENT SYSTEMS

(Effective from the academic year 2018 -2019)

Semester: 05 **Credits: 03**
Number of Contact Hours/Week 3:2:0 **CIE Marks: 50**
Total Number of Hours: 40 **Exam Marks: 50**

Course Learning Objectives: This course (18CS53) will enable students to:

COs	Description
18CS53.1	Demonstrate the basics of database, architecture, and data modelling.
18CS53.2	Apply the basics of database and formulate the problems based on mathematical models using relational algebra concepts.
18CS53.3	Formulate solutions to a given problem using Structured Query Language (SQL).
18CS53.4	Analyze the applicapability of normalization level to a given problem in databases.
18CS53.5	Demonstrate the knowledge of building applications to interact with databases, perform transactions, and provide security and recovery protocols.

MODULE – I

Introduction to Databases
Introduction, Characteristics of database approach,
Advantages of using the DBMS approach,
History of database applications.

Overview of Database Languages and Architectures
Data Models, Schemas, and Instances.
Three schema architecture and data independence,
Database languages, and interfaces,
The Database System environment.

Conceptual Data Modelling using Entities and Relationships
Entity types, Entity sets,
Attributes, roles, and structural constraints,
Weak entity types,
ER diagrams,
Examples

Introduction

- Databases and database systems are an essential component of life in modern society.
- For example,
  * In the bank to deposit or withdraw funds.
  * If we make a hotel, airline or railway reservation.
  * If we access a computerized library catalog to search for a bibliographic item.
  * When we are purchasing items at a supermarket often automatically updates the database that holds the inventory of grocery items
  * If we purchase something online—such as a book, toy, or computer—chances are that our activities will involve someone or some computer program accessing a database..
- These interactions are examples of what we may call **traditional database applications**, in which most of the information that is stored and accessed is either *textual or numeric*.
- Advances in technology have led to exciting new applications of database systems. New media technology has made it possible to store images, audio clips, and video streams digitally.
  * These types of files are becoming an important component of **multimedia databases.**
  * **Geographic information systems (GIS)** can store and analyze maps, weather data, and satellite images.
  * **Data warehouses and online analytical processing (OLAP)** systems are used in many companies to extract and analyze useful business information from very large databases to support decision making.
  * Real-time and active database technology is used to control industrial and manufacturing processes.
  * Database search techniques are being applied to the World Wide Web to improve the search for information that is needed by users browsing the Internet.

1.1 Introduction

- Databases play a critical role in almost all areas where computers are used, including business, electronic commerce, engineering, medicine, genetics, law, education, and library science.
- ***DATABASE: "A database is a collection of related data."***
- By **data**, we mean **known facts that can be recorded and that have implicit meaning**.
- For example,
  * Consider the names, telephone numbers, and addresses of the people you know. This can be recorded as data in an indexed address book or on a hard drive or using software such as Microsoft Access or Excel. This collection of related data with an implicit meaning is a database.

Properties of a database:

- A database has the following **implicit properties**:
  * A database represents some aspect of the **real world**, sometimes called the miniworld or the universe of discourse (UoD). Changes to the miniworld are reflected in the database. (**Mini-world:** Some part of the real world about which data is stored in a database. For example, student grades and transcripts at a university.)
  * A database is a **logically coherent collection of data with some inherent meaning**. A random assortment of data cannot correctly be referred to as a database.

  - A database is designed, built, and populated with data for a **specific purpose**. It has an intended group of users and some preconceived applications in which these users are interested.
- In order for a database to be **accurate and reliable** at all times, it must be a true reflection of the miniworld that it represents; therefore, **changes must be reflected** in the database as soon as possible.
- A database can be of **any size and complexity**.
- For example, (simple to complex)
  - The list of names, phone numbers and addresses may consist of only a few hundred records, each with a simple structure.
  - The computerized catalog of a large library may contain half a million entries organized under different categories—by primary author's last name, by subject, by book title—with each category organized alphabetically.
  - A database of even greater size and complexity is maintained by the **Internal Revenue Service (IRS)** to monitor **tax forms filed by U.S. taxpayers**. If we assume that there are **100 million taxpayers** and each taxpayer files an average of five forms with approximately 400 characters of information per form, we would have a database of $100 \times 10^6 \times 400 \times 5$ characters (bytes) = 2×10^{11} of information. If the IRS keeps the past three returns of each taxpayer in addition to the current r
  - eturn(3+1=4), we would have a database of $2 \times 10^{11} \times 4 = 8 \times 10^{11}$ bytes (800 gigabytes). This huge amount of information must be organized and managed so that users can search for, retrieve, and update the data as needed.
  - An example of a large commercial database is **Amazon.com**. It contains data for over **20 million books**, CDs, videos, DVDs, games, electronics, apparel, and other items. The database occupies over **2 terabytes** (a terabyte is 10^{12} bytes worth of storage) and is **stored on 200 different computers** (called servers). About **15 million visitors** access Amazon.com each day and use the database to make purchases. The database is continually updated as new books and other items are added to the inventory and stock quantities are updated as purchases are transacted. About 100 people are responsible for keeping the Amazon database up-to-date.

- **DBMS**: A **database management system** is a collection of programs that enables users to create and maintain a database. The DBMS is a **general-purpose software system** that facilitates the processes of **defining, constructing, manipulating, and sharing databases** among various users and applications.
  - **Defining** a database involves specifying the data types, structures, and constraints of the data to be stored in the database. The **database definition** or descriptive information is also stored by the DBMS in the form of a **database catalog or dictionary**; it is called **meta-data**.
  - **Constructing** the database is the process of **storing the data on** some **storage medium** that is controlled by the DBMS.
  - **Manipulating** a database includes functions such as **querying** the database to retrieve specific data, **updating** the database to reflect changes in the miniworld, and **generating reports** from the data.
  - **Sharing** a database allows **multiple users and programs** to **access** the database **simultaneously**.

- Other important functions provided by the DBMS include ***protecting*** the database and ***maintaining*** it over a long period of time.
    * **Protection** includes system protection against hardware or software malfunction (or crashes) and security protection against unauthorized or malicious access.
    * A typical large database may have a life cycle of many years, so the DBMS must be able to **maintain** the database system by allowing the system to evolve as requirements change over time.
- 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 database and DBMS software together forms a database system.** Figure 1.1 illustrates some of the concepts we have discussed so far.

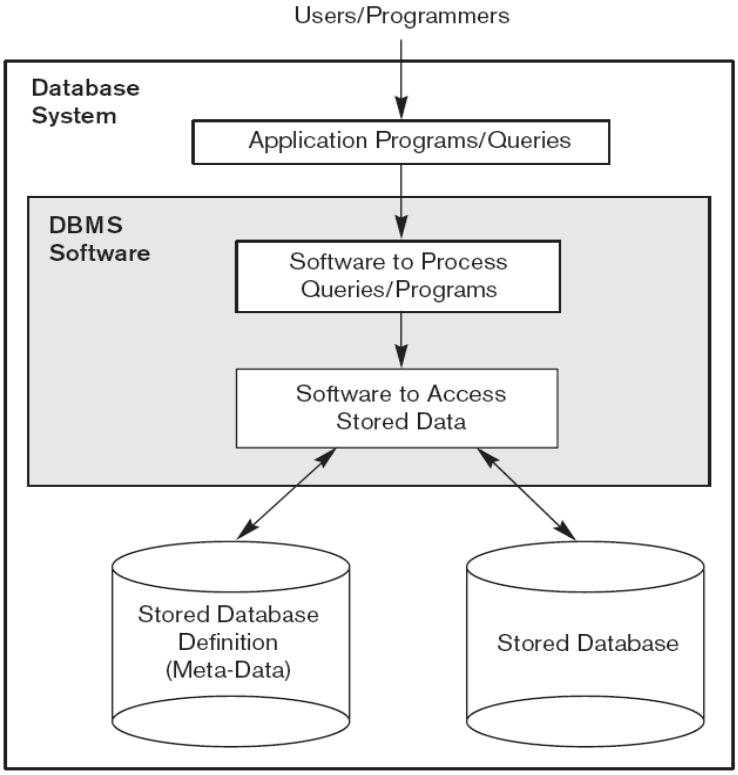


Figure 1.1
A simplified database system environment.

1.2 An Example

- Consider a simple example of a UNIVERSITY database for maintaining information concerning students, courses and grades in a university environment.
- Figure 1.2 shows the database structure and a few sample data for such a database. The database is organized as five files, each of which stores data records of the same type.
    * STUDENT file stores data on each student
    * COURSE file stores data on each course
    * SECTION file stores data on each section of a course
    * GRADE_REPORT file stores the grades that student receive in the various sections they have completed, and

* PREREQUISITE file stores the prerequisites of each course.

STUDENT

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

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Figure 1.2
A database that stores student and course information.

- To define this database, we must specify the structure of the records of each file by specifying the different types of data elements to be stored in each record.
- We must also specify a data type for each data element within a record. For example,
  * We can specify that Name of STUDENT is a string of alphabetic characters.
  * Student_number of STUDENT is an integer.
  * Grade of GRADE_REPORT is a single character from the set {‘A’, ‘B’, ‘C’, ‘D’, ‘F’, ‘I’}.

    * We may also use a coding scheme to represent the values of a data item. For example, in Figure 1.2 we represent the Class of a STUDENT as 1 for freshman, 2 for sophomore, 3 for junior, 4 for senior, and 5 for graduate student.
- Records in the various files may be related.
- For example, the record for Smith in the STUDENT file is related to two records in the GRADE_REPORT file that specify Smith's grades in two sections.
- Similarly, each record in the PREREQUISITE file relates two course records: one representing the course and the other representing the prerequisite.
- Most medium-size and large databases include many types of records and have many relationships among the records.
- Database manipulation involves querying and updating. Examples of queries are as follows:
    * Retrieve the transcript—a list of all courses and grades—of 'Smith'
    * List the names of students who took the section of the 'Database' course offered in fall 2008 and their grades in that section
    * List the prerequisites of the 'Database' course
- Examples of updates include the following:
    * Change the class of 'Smith' to sophomore
    * Create a new section for the 'Database' course for this semester
    * Enter a grade of 'A' for 'Smith' in the 'Database' section of last semester
- Design of a new application for an existing database or design of a brand new database starts off with a phase called **requirements specification and analysis**.
- These requirements are documented in detail and transformed into a **conceptual design** that can be represented and manipulated using some computerized tools so that it can be easily maintained, modified, and transformed into a database implementation. (We will introduce a model called the **Entity-Relationship model in Chapter 7** that is used for this purpose.)
    * **Some mini-world *relationships*:**
        - SECTIONs *are of specific* COURSEs
        - STUDENTs *take* SECTIONs
        - COURSEs *have prerequisite* COURSEs
        - INSTRUCTORs *teach* SECTIONs
        - COURSEs *are offered by* DEPARTMENTs
        - STUDENTs *major in* DEPARTMENTs
- The design is then translated to a **logical design** that can be expressed in a data model implemented in a commercial DBMS. (In this book we will emphasize a data model known as the **Relational Data Model from Chapter 3** onward. This is currently the most popular approach for designing and implementing databases using relational DBMSs.)
- The final stage is **physical design**, during which further specifications are provided for storing and accessing the database. The database design is implemented, populated with actual data, and continuously maintained to reflect the state of the mini world.

1.3 Characteristics of the Database Approach

- In traditional file processing, each user defines and implements the files needed for a specific software application as part of programming the application.

  - For example, one user, the grade reporting office, may keep files on students and their grades. Programs to print a student's transcript and to enter new grades are implemented as part of the application.
  - A second user, the accounting office, may keep track of students' fees and their payments. Although both users are interested in data about students, each user maintains separate files — and programs to manipulate these files—because each requires some data not available from the other user's files. This redundancy in defining and storing data results in wasted storage space and in redundant efforts to maintain common up-to-date data.
- In the database approach, a single repository maintains data that is defined once and then accessed by various users.
- In file systems, each application is free to name data elements independently.
- In contrast, in a database, the names or labels of data are defined once, and used repeatedly by queries, transactions, and applications.
- **The main characteristics of the database approach versus the file-processing approach are the following:**
  - Self-describing nature of a database system
  - Insulation between programs and data, and data abstraction
  - Support of multiple views of the data
  - Sharing of data and multiuser transaction processing
  -
- Data is stored in flat files and can be accessed using any programming language. The file based approach suffers following problems:
  - Dependency of program on physical structure of data
  - Complex process to retrieve data
  - Loss of data on concurrent access
  - Inability to give access based on record (Security)
  - Data redundancy

- **Self-Describing Nature of a Database System**
- A fundamental characteristic of the database approach is that the database system contains not only the database itself but also a complete definition or description of the database structure and constraints. This definition is stored in the **DBMS catalog**, which contains information such as the **structure of each file**, **the type** and **storage format** of each data item, and various **constraints on the data**. The information stored in the catalog is called **meta-data**, and it describes the structure of the **primary database** (Figure 1.1).
- The catalog is used by the DBMS software and also by database users who need information about the database structure.
- The DBMS software must work equally well with any number of database applications—for example, a university database, a banking database, or a company database—as long as the database definition is stored in the catalog.
- In traditional file processing, data definition is typically part of the application programs themselves. Hence, these programs are constrained to work with only one specific database, whose structure is declared in the application programs. For example, an application program written in C++ may have struct or class declarations to define its files.

- Whereas file-processing software can access only specific databases, DBMS software can access diverse databases by extracting the database definitions from the catalog and using these definitions.
- For the example shown in Figure 1.2, the DBMS catalog will store the definitions of all the files shown. Figure 1.3 shows some sample entries in a database catalog.
- Whenever a request is made to access, say, the Name of a STUDENT record, the DBMS software refers to the catalog to determine the structure of the STUDENT file and the position and size of the Name data item within a STUDENT record.

RELATIONS

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

COLUMNS

Column_name	Data_type	Belongs_to_relation
Name	Character (30)	STUDENT
Student_number	Character (4)	STUDENT
Class	Integer (1)	STUDENT
Major	Major_type	STUDENT
Course_name	Character (10)	COURSE
Course_number	XXXXNNNN	COURSE
….	….	…..
….	….	…..
….	….	…..
Prerequisite_number	XXXXNNNN	PREREQUISITE

Note: Major_type is defined as an enumerated type with all known majors.
XXXXNNNN is used to define a type with four alpha characters followed by four digits.

Figure 1.3
An example of a database catalog for the database in Figure 1.2.

1.3.1 Insulation between Programs and Data, and Data Abstraction

- In traditional file processing, the structure of data files is embedded in the application programs, so any changes to the structure of a file may require changing all programs that access that file. By contrast, DBMS access programs do not require such changes in most cases. The structure of data files is stored in the DBMS catalog separately from the access programs.We call this property **program-data independence**.
    * For example, a file access program may be written in such a way that it can access only STUDENT records of the structure shown in Figure 1.4. If we want to add another piece of data to each STUDENT record, say the Birth_date, such a program will no longer work and must be changed. By contrast, in a DBMS environment, we only need to change the description of STUDENT records in the catalog (Figure 1.3) to reflect the inclusion of the

new data item Birth_date; no programs are changed. The next time a DBMS program refers to the catalog, the new structure of STUDENT records will be accessed and used.

- In some types of database systems, such as **object-oriented and object-relational systems** (see Chapter 11), users can define operations on data as part of the database definitions. An operation (also called a function or method) is specified in two parts. The **interface** (or signature) of an operation includes the operation name and the data types of its arguments (or parameters). The **implementation** (or method) of the operation is specified separately and can be changed without affecting the interface. User application programs can operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented. This may be termed **program-operation independence**.
- The characteristic that allows program-data independence and program-operation independence is called **data abstraction**.
- The internal implementation of a file may be defined by its record length—the number of characters (bytes) in each record—and each data item may be specified by its starting byte within a record and its length in bytes. The STUDENT record would thus be represented as shown in Figure 1.4.

Data Item Name	Starting Position in Record	Length in Characters (bytes)
Name	1	30
Student_number	31	4
Class	35	1
Major	36	4

Figure 1.4
Internal storage format for a STUDENT record, based on the database catalog in Figure 1.3.

1.3.2 Support of Multiple Views of the Data

- A database typically has many 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**.
- Some users may not need to be aware of whether the data they refer to is stored or derived. A multiuser DBMS whose users have a variety of distinct applications must provide facilities for defining multiple views.
    * For example, one user of the database of Figure 1.2 may be interested only in accessing and printing the transcript of each student; the view for this user is shown in Figure 1.5(a).
    * A second user, who is interested only in checking that students have taken all the prerequisites of each course for which they register, may require the view shown in Figure 1.5(b).

(a)

TRANSCRIPT

Student_name	Student_transcript				
	Course_number	Grade	Semester	Year	Section_id
Smith	CS1310	C	Fall	08	119
	MATH2410	B	Fall	08	112
Brown	MATH2410	A	Fall	07	85
	CS1310	A	Fall	07	92
	CS3320	B	Spring	08	102
	CS3380	A	Fall	08	135

(b)

COURSE_PREREQUISITES

Course_name	Course_number	Prerequisites
Database	CS3380	CS3320
		MATH2410
Data Structures	CS3320	CS1310

Figure 1.5
Two views derived from the database in Figure 1.2. (a) The TRANSCRIPT view. (b) The COURSE_PREREQUISITES view.

1.3.4 Sharing of Data and Multiuser Transaction Processing

- A multiuser DBMS, as its name implies, must allow multiple users to access the database at the same time. This is essential if data for multiple applications is to be integrated and maintained in a single database.
- The DBMS must include concurrency control software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct.
    * For example, when several reservation agents try to assign a seat on an airline flight, the DBMS should ensure that each seat can be accessed by only one agent at a time for assignment to a passenger. These types of applications are generally called **online transaction processing (OLTP)** applications. A fundamental role of multiuser DBMS software is to ensure that concurrent transactions operate correctly and efficiently.
- 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.
- The preceding characteristics are important in distinguishing a DBMS from traditional file-processing software.

1.4 Actors on the Scene

- Users may be divided into
  - * Those who actually use and control the database content, and those who design, develop and maintain database applications (called "Actors on the Scene"), and
  - * Those who design and develop the DBMS software and related tools, and the computer systems operators (called "Workers Behind the Scene").
- Actors on the scene

1.4.1 Database Administrators

- In a database environment, the **primary resource is the database** itself, and the **secondary resource is the DBMS and related software**. Administering these resources is the responsibility of the **database administrator (DBA)**.
- The DBA is responsible for
  - * Authorizing access to the database,
  - * Coordinating and monitoring its use,
  - * Acquiring software and hardware resources as needed.
  - * Controlling its use i.e. security breaches and
  - * Monitoring efficiency of operations i.e system response time.

1.4.2 Database Designers

- Responsible for **identifying the data** to be stored in the database and for **choosing appropriate structures** to represent and store this data.
- They must communicate with the end-users and understand their needs.
- Database designers typically interact with each potential group of users and develop **views** of the database that meet the data and processing requirements of these groups.
- Each view is then analyzed and integrated with the views of other user groups. The final database design must be capable of supporting the requirements of all user groups.

1.4.3 End Users

- End users are the people whose jobs require access to the database for querying, updating, and generating reports.
- End-users can be categorized into:
  - * **Casual end users**: access database occasionally when needed.
    - Use a sophisticated database query language to specify their requests
    - They are middle- or high-level managers or other occasional browsers.
  - * **Naive or parametric end users:** they make up a sizable portion of database end users.
    - They constantly query and update the database, using standard types of queries and updates—called **canned transactions**—that have been carefully **programmed and tested**.
    - For example,
      - Bank tellers check account balances and post withdrawals and deposits.
      - Reservation agents for airlines, hotels, and car rental companies check availability for a given request and make reservations.

- Employees at receiving stations for shipping companies enter package identifications via bar codes and descriptive information through buttons to update a central database of received and in-transit packages.

* **Sophisticated end users:** include engineers, scientists, business analysts, and others who thoroughly familiarize themselves with the facilities of the DBMS in order to implement their own applications to meet their complex requirements.
* **Standalone users:** maintain personal databases by using ready-made program packages that provide easy-to-use menu-based or graphics-based interfaces. An example is the user of a tax package that stores a variety of personal financial data for tax purposes.

1.4.4 System Analysts and Application Programmers (Software Engineers)

- System analysts determine the requirements of end users, especially naive and parametric end users, and develop specifications for standard canned transactions that meet these requirements.
- Application programmers implement these specifications as programs; then they test, debug, document, and maintain these canned transactions.
- Such analysts and programmers—commonly referred to as software developers or software engineers—should be familiar with the full range of capabilities provided by the DBMS to accomplish their tasks.

1.5 Workers behind the Scene

- They are associated with the design, development, and operation of the DBMS software and system environment.
- These persons are typically not interested in the database content itself. We call them the workers behind the scene, and they include the following categories:
    * **DBMS system designers and implementers:** design and implement the DBMS modules and interfaces as a software package.
        - A DBMS is a very complex software system that consists of many components, or modules, including modules for
            - implementing the catalog,
            - query language processing,
            - interface processing,
            - accessing and buffering data,
            - controlling concurrency, and
            - handling data recovery and security.
        - The DBMS must interface with other system software such as the operating system and compilers for various programming languages.
    * **Tool developers**: design and implement tools—the software packages that facilitate database modelling and design, database system design, and improved performance.
        - They include packages for
            - database design,
            - performance monitoring,
            - natural language or graphical interfaces,
            - prototyping,

- Simulation, and
- Test data generation.

* **Operators and maintenance personnel (system administration personnel):** are responsible for the actual running and maintenance of the hardware and software environment for the database system.
* Although these categories of workers behind the scene are instrumental in making the database system available to end users, they do not use database for their own use.

1.6 Advantages of Using the DBMS Approach

- In addition to the following four main characteristics, a database also has some additional features that are described in this section
  * Self-describing nature of a database system
  * Insulation between programs and data, and data abstraction
  * Support of multiple views of the data
  * Sharing of data and multiuser transaction processing

1.6.1 Controlling Redundancy

- In traditional software development utilizing file processing, every user group maintains its own files for handling its data-processing applications.
- For example, consider the UNIVERSITY database example of Section 1.2; here, two groups of users might be the course registration personnel and the accounting office.
- This redundancy in storing the same data multiple times leads to several problems.
  * First, entering data on a new student—multiple times: once for each file where student data is recorded. This leads to **duplication of effort**.
  * Second, **storage space is wasted** when the same data is stored repeatedly, and this problem may be serious for large databases.
  * Third, files that represent the same **data may become inconsistent**. This may happen because an update is applied to some of the files but not to others. Even if an update—such as adding a new student—is applied to all the appropriate files, the data concerning the student may still be inconsistent because the updates are applied independently by each user group. For example, one user group may enter a student's birth date **erroneously as 'JAN-19-1988'**, whereas the other user groups may enter the **correct value of 'JAN-29-1988'.**
- In the database approach, the views of different user groups are integrated during database design. Ideally, we should have a database design that stores each logical data item—such as a student's name or birth date—in only one place in the database. This is known as data normalization, and it ensures consistency and saves storage space.
- It is sometimes necessary to use controlled redundancy for improving the performance of queries. For example, we may store StudentName and CourseNumber redundantly in a GRADE_REPORT file (Figure 1.5a) because whenever we retrieve a GRADE_REPORT record, we want to retrieve the student name and course number along with the grade, student number, and section identifier. By placing all the data together, we do not have to search multiple files to collect this data. In such cases, the DBMS should have the capability to control this redundancy so as to prohibit inconsistencies among the files. This may be done by automatically checking that the StudentName-

StudentNumber values in any GRADE_REPORT record in Figure 1.5a match one of the Name-StudentNumber values of a STUDENT record (Figure 1.2). Similarly, the SectionIdentifier-CourseNumber values in GRADE_REPORT can be checked against SECTION records. Such checks can be specified to the DBMS during database design and automatically enforced by the DBMS whenever the GRADE_REPORT file is updated. Figure 1.5b shows a GRADE3EPORT record that is inconsistent with the STUDENT file of Figure 1.2, which may be entered erroneously if the redundancy is not controlled.

(a)

GRADE_REPORT	StudentNumber	StudentName	SectionIdentifier	CourseNumber	Grade
	17	Smith	112	MATH2410	B
	17	Smith	119	CS1310	C
	8	Brown	85	MATH2410	A
	8	Brown	92	CS1310	A
	8	Brown	102	CS3320	B
	8	Brown	135	CS3380	A

(b)

GRADE_REPORT	StudentNumber	StudentName	SectionIdentifier	CourseNumber	Grade
	17	Brown	112	MATH2410	B

FIGURE 1.5 Redundant storage of StudentName and CourseNumber in GRADE_REPORT. (a) Consistent data. (b) Inconsistent record.

1.6.2 Restricting Unauthorized Access

- When multiple users share a large database, it is likely that most users will not be authorized to access all information in the database.
- For example, financial data is often considered confidential, and hence only authorized persons are allowed to access such data. In addition, some users may be permitted only to retrieve data, whereas others are allowed both to retrieve and to update. Hence, the type of access operation-retrieval or update-must also be controlled. Typically, users or user groups are given account numbers protected by passwords, which they can use to gain access to the database.
- A DBMS should provide a **security and authorization subsystem**, which the DBA uses to create accounts and to specify account restrictions. The DBMS should then enforce these restrictions automatically. Notice that we can apply similar controls to the DBMS software. For example, only the DBA's staff may be allowed to use certain **privileged software**, such as the software for creating new accounts. Similarly, parametric users may be allowed to access the database only through the canned transactions developed for their use.

1.6.3 Providing Persistent Storage for Program Objects

- Databases can be used to provide persistent storage for program objects and data structures. This is one of the main reasons for object-oriented database systems. Programming languages typically have complex data structures, such as record types in Pascal or class definitions in C++ or Java.
- Object-oriented database systems are compatible with programming languages such as C++ and Java, and the DBMS software automatically performs any necessary conversions. Hence, a complex object in C++ can be stored permanently in an object-oriented DBMS. Such an object is said to be

persistent, since it survives the termination of program execution and can later be directly retrieved by another C+ + program.

1.6.4 Providing Storage Structures for Efficient Query Processing

- Database systems must provide capabilities for efficiently executing queries and updates. Because the database is typically stored on disk, the DBMS must provide specialized data structures to speed up disk search for the desired records. Auxiliary files called **indexes** are used for this purpose. Indexes are typically based on tree data structures or hash data structures, suitably modified for disk search.
- The query processing and optimization module of the DBMS is responsible for choosing an efficient query execution plan for each query based on the existing storage structures. The choice of which indexes to create and maintain is part of physical database design and tuning, which is one of the responsibilities of the DBA staff.

1.6.5 Providing Backup and Recovery

- A DBMS must provide facilities for recovering from hardware or software failures. The backup and recovery subsystem of the DBMS is responsible for recovery.
- For example, if the computer system fails in the middle of a complex update transaction, the recovery subsystem is responsible for making sure that the database is restored to the state it was in before the transaction started executing.
- Alternatively, the recovery subsystem could ensure that the transaction is resumed from the point at which it was interrupted so that its full effect is recorded in the database.

1.6.6 Providing Multiple User Interfaces

- Because many types of users with varying levels of technical knowledge use a database, a DBMS should provide a variety of user interfaces.
- These include
    * query languages for casual users,
    * programming language interfaces for application programmers,
    * forms and command codes for parametric users, and
    * menu-driven interfaces and natural language interfaces for stand-alone users.
- Both forms-style interfaces and menu-driven interfaces are commonly known as graphical user interfaces (GUIs).
- Many specialized languages and environments exist for specifying GUls. Capabilities for providing Web GUl interfaces to a database- or Web-enabling a database-are also quite common.

1.6.7 Representing Complex Relationships among Data

- A database may include numerous varieties of data that are interrelated in many ways.
- Consider the example shown in Figure 1.2. The record for Brown in the STUDENT file is related to four records in the GRADE_REPORT file.

- Similarly, each section record is related to one course record as well as to a number of GRADE_REPORT records-one for each student who completed that section.
- A DBMS must have the capability to represent a variety of complex relationships among the data as well as to retrieve and update related data easily and efficiently.

1.6.8 Enforcing Integrity Constraints

- Most database applications have certain integrity constraints that must hold for the data. A DBMS should provide capabilities for defining and enforcing these constraints. The simplest type of integrity constraint involves specifying a data type for each data item.
- For example, in Figure 1.2, we may specify that the value of the
    * Class data item within each STUDENT record must be an integer between 1 and 5
    * Name must be a string of no more than 30 alphabetic characters.
- A more complex type of constraint that frequently occurs involves specifying that a record in one file must be related to records in other files.
- For example, in Figure 1.2, we can specify that "every section record must be related to a course record."
- Another type of constraint specifies uniqueness on data item values, such as "every course record must have a unique value for CourseNumber."
- These constraints are derived from the meaning or semantics of the data and of the miniworld it represents.
- It is the database designers' responsibility to identify integrity constraints during database design. Some constraints can be specified to the DBMS and automatically enforced.
- Other constraints may have to be checked by update programs or at the time of data entry. A data item may be entered **erroneously** and still satisfy the specified integrity constraints. For example, if a student receives a grade of ‘A’ but a grade of ‘C’ is entered in the database, the DBMS cannot discover this error automatically, because ‘C’ is a valid value for the Grade data type.

1.6.9 Permitting Inferencing and Actions Using Rules

- Some database systems provide capabilities for defining deduction rules for inferencing new information from the stored database facts. Such systems are called deductive database systems.
- For example, there may be complex rules in the miniworld application for determining when a **student is on probation**. These can be specified declaratively as rules, which when compiled and maintained by the DBMS can determine all students on probation.
- In a traditional DBMS, an explicit procedural program code would have to be written to support such applications. But if the miniworld rules change, it is generally more convenient to change the declared deduction rules than to recode procedural programs.
- In today’s database systems, we have:
    * **Triggers**: A trigger is a form of a rule activated by updates to the table, which results in performing some additional operations to some other tables, sending messages and so on.
    * **Stored procedures**: more involved procedures to enforce rules.
    * More powerful functionality is provided by **active database systems**, which provide active rules that can automatically initiate actions when certain events and conditions occur.

1.6.10 Additional Implications of Using the Database Approach

This section discusses some additional implications of using the database approach that can benefit most organizations.

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.
- Standards can be defined for names and formats of data elements, display formats, report structures, terminology, and so on.
- The DBA can enforce standards in a centralized database environment more easily than in an environment where each user group has control of its own files and software.

Reduced Application Development Time.

- A prime selling feature of the database approach is that developing a new application-such as the retrieval of certain data from the database for printing a new report-takes very little time.
- Designing and implementing a new database from scratch may take more time than writing a single specialized file application. However, once a database is up and running, substantially less time is generally required to create new applications using DBMS facilities.
- Development time using a DBMS is estimated to be one-sixth to one-fourth of that for a traditional file system.

FIexibility

- It may be necessary to change the structure of a database as requirements change. For example, a new user group may emerge that needs information not currently in the database. In response, it may be necessary to add a file to the database or to extend the data elements in an existing file.
- Modern DBMSs allow certain types of evolutionary changes to the structure of the database without affecting the stored data and the existing application programs.

Availability of Up-to-Date Information.

- A DBMS makes the database available to all users. As soon as one user's update is applied to the database, all other users can immediately see this update. This availability of up-to-date information is essential for many transaction-processing applications, such as reservation systems or banking databases, and it is made possible by the concurrency control and recovery subsystems of a DBMS.

Economies of Scale.

- The DBMS approach permits consolidation of data and applications, thus reducing the amount of wasteful overlap between activities of dataprocessing personnel in different projects or departments. This enables the whole organization to invest in more powerful processors, storage devices, or communication gear, rather than having each department purchase its own (weaker) equipment. This reduces overall costs of operation and management.

1.7 A BRIEF HISTORY OF DATABASE APPLICATIONS

1.7.1 Early Database Applications Using Hierarchical and Network Systems

- Many early database applications maintained records in large organzations, such as corporations, universities, hospitals, and banks.
- In many of these applications, there were large **numbers of records of similar structure**.
- For example, in a university application, similar information would be kept for each student, each course, each grade record, and so on. There were also many types of records and many interrelationships among them.
- One of the main problems with early database systems was the intermixing of conceptual relationships with the physical storage and placement of records on disk.
- For example, the grade records of a particular student could be physically stored next to the student record. Although this provided very efficient access for the original queries and transactions that the database was designed to handle, it did not provide **enough flexibility to access records efficiently when new queries and transactions were identified.**
- Another shortcoming of early systems was that they **provided only programming language interfaces.** This made it time-consuming and expensive to implement new queries and transactions, since new programs had to be written, tested, and debugged.
- Most of these database systems were implemented on large and expensive mainframe computers starting in the **mid-1960s and through the 1970s and 1980s**. The main types of early systems were based on three main paradigms: hierarchical systems, network model based systems, and inverted file systems.

1.7.2 Providing Application Flexibility with Relational Databases

- Relational databases were originally proposed to separate the **physical storage** of data from its conceptual representation and to provide a **mathematical foundation for databases**.
- The relational data model also introduced high-level query languages that provided an alternative to programming language interfaces; hence, it was a lot quicker to write new queries.
- Relational systems were initially targeted to the same applications as earlier systems, but were meant to provide flexibility to quickly develop new queries and to reorganize the database as requirements changed.
- Early experimental relational systems developed in the late 1970s and the commercial RDBMSs (relational database management systems) introduced in the early 1980s were quite slow, since they did not use physical storage pointers or record placement to access related data records.
- With the development of new storage and indexing techniques and better query processing and optimization, their performance improved. Eventually, relational databases became the dominant type of database systems for traditional database applications.
- Relational databases now exist on almost all types of computers, from small personal computers to large servers.

1.7.3 Object-Oriented Applications and the Need for More Complex Databases

- The emergence of object-oriented programming languages in the 1980s and the need to store and share complex-structured objects led to the development of object-oriented databases. Initially, they were considered a competitor to relational databases, since they **provided** more **general data structures**. They also incorporated many of the useful object oriented paradigms, such as **abstract data types, encapsulation of operations, inheritance**, and **object identity**.
- However, the complexity of the model and the lack of an early standard contributed to their limited use.
- They are now mainly used in specialized applications, such as engineering design, multimedia publishing, and manufacturing systems.

1.7.4 Interchanging Data on the Web for E-Commerce

- The World Wide Web provided a large network of interconnected computers. Users can create documents using a Web publishing language, such as HTML (Hypertext Mark-up Language), and store these documents on Web servers where other users (clients) can access them.
- Documents can be linked together through hyperlinks, which are pointers to other documents.
- In the 1990s, electronic commerce (e-commerce) emerged as a major application on the Web. It quickly became apparent that parts of the information on e-commerce Web pages were often dynamically extracted data from DBMSs. A variety of techniques were developed to allow the interchange of data on the Web.
- Currently, XML (extended Mark-up Language) is considered to be the primary standard for interchanging data among various types of databases and Web pages. XML combines concepts from the models used in document systems with database modelling concepts.

1.7.5 Extending Database Capabilities for New Applications

- The success of database systems in traditional applications encouraged developers of other types of applications to attempt to use them. Such applications traditionally used their own specialized file and data structures. The following are examples of these applications:
    * **Scientific applications** that store large amounts of data resulting from scientific experiments in areas such as **high-energy physics** or the **mapping of the human genome.**
    * **Storage and retrieval of images**, from scanned news or personal photographs to satellite photograph images and images from medical procedures such as X-rays or MRI (magnetic resonance imaging).
    * **Storage and retrieval of videos**, such as movies, or video clips from news or personal digital cameras.
    * **Data mining applications** that analyze large amounts of data searching for the occurrences of specific patterns or relationships.
    * Spatial applications that store spatial locations of data such as **weather information** or maps used in **geographical information systems**.
    * Time series applications that store information such as economic data at regular points in time, for example, **daily sales** or monthly **gross national product figures**.

- It was quickly apparent that basic relational systems were not very suitable for many of these applications, usually for one or more of the following reasons:
    * More **complex data structures** were needed for modelling the application than the simple relational representation.
    * **New data types** were needed in addition to the basic numeric and character string types.
    * New **operations and query language constructs** were necessary to manipulate the new data types.
    * New storage and indexing structures were needed.
- This led DBMS developers to add functionality to their systems. Some functionality was general purpose, such as incorporating concepts from object-oriented databases into relational systems.
- Other functionality was special purpose, in the form of optional modules that could be used for specific applications. For example, users could buy a time series module to use with their relational DBMS for their time series application.
- Many large organizations use a variety of software application packages that work closely with database back-ends.
- **Enterprise Resource Planning (ERP)** is used to consolidate a variety of functional areas within an organization, including production, sales, distribution, marketing, finance, human resources, and so on.
- Customer Relationship Management (CRM) software spans order processing as well as marketing and customer support functions. These applications are Web-enabled in that internal and external users are given a variety of Web portal interfaces to interact with the back-end databases.

1.7.6 Databases versus Information Retrieval

- **Traditionally**, database technology applies to **structured and formatted data** that arises in routine applications in government, business, and industry.
- Database technology is heavily used in manufacturing, retail, banking, insurance, finance, and health care industries, where structured data is collected through forms, such as invoices or patient registration documents.
- An area related to database technology is **Information Retrieval (IR)**, which deals with books, manuscripts, and various forms of library-based articles. Data is indexed, cataloged, and annotated using keywords.
- IR is concerned with searching for material based on these keywords, and with the many problems dealing with document processing and free-form text processing.
- There has been a considerable amount of work done on
    * Searching for text based on keywords,
    * Finding documents and ranking them based on relevance,
    * A utomatic text categorization,
    * Classification of text documents by topics, and so on.
- With the advent of the Web HTML pages running into the billions, there is a need to apply many of the IR techniques to processing data on the Web.
- Data on Web pages typically contains **images, text, and objects** that are active and **change dynamically**.

- Retrieval of information on the Web is a new problem that requires techniques from databases and IR to be applied in a variety of new combinations.

1.8 WHEN NOT TO USE A DBMS

- In spite of the advantages of using a DBMS, there are a few situations in which such a system 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.
- Additional problems may arise if the database designers and DBA do not properly design the database or if the database systems applications are not implemented properly.
- Hence, it may be more desirable to use regular files under the following circumstances:
    * The database and applications are **simple**, well defined, and **not expected to change**.
    * There are stringent **real-time requirements** for some programs that may not be met because of DBMS overhead. (Ex: Embedded systems)
    * **Multiple-user access** to data is **not required**.
    * Embedded systems with limited storage capacity, where a general-purpose DBMS would not fit.
- Certain industries and applications have elected not to use general-purpose DBMSs. For example,
    * **Computer-Aided Design (CAD)** tools used by mechanical and civil engineers have proprietary file and data management software that is geared for the internal manipulations of drawings and 3D objects.
    * **Communication and switching systems** designed by companies like AT&T were early manifestations of database software that was made to run very fast with hierarchically organized data for quick access and routing of calls.
    * **Geographic information systems (GIS)** implementations often implement their own data organization schemes for efficiently implementing functions related to processing maps, physical contours, lines, polygons.
- General-purpose DBMSs are inadequate for their purpose.

Database System Concepts and Architecture

- 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 client/server system architecture.
- This evolution mirrors the trends in computing, where **large centralized mainframe computers** are being replaced by **hundreds of distributed workstations and personal computers** connected **via communications networks to** various types of server machines—**Web servers, database servers, file servers, application servers**, and so on.
- In a basic client/server DBMS architecture, the system functionality is distributed between two types of modules:
    * A client module is designed to run on a user workstation or personal computer. Typically, application programs and user interfaces that access the database run in the client module. Hence, the client module handles user interaction and provides the user-friendly interfaces such as forms- or menu-based GUIs (graphical user interfaces).
    * The other kind of module, called a server module, typically handles data storage, access, search, and other functions.

2.1 Data Models, Schemas, and Instances

- One fundamental characteristic of the database approach is that it provides some level of **data abstraction**.
- Data abstraction generally refers to the suppression of details of data organization and storage, and the highlighting of the essential features for an improved understanding of data.
- One of the main characteristics of the database approach is to support data abstraction so that **different users can perceive data at their preferred level of detail**.
- **A data model—a collection of concepts that can be used to describe the structure of a database—provides the necessary means to achieve this abstraction.**
- **By structure of a database we mean the data types, relationships, and constraints that apply to the data.**
- Most data models also include a set of basic operations for specifying retrievals and updates on the database.
- In addition to the **basic operations** provided by the data model, it is becoming more common to include concepts in the data model to specify the **dynamic aspect or behaviour** of a database application. This allows the database designer to specify a set of valid user-defined operations that are allowed on the database objects.
- An example of a **user-defined operation could be COMPUTE_GPA, which can be applied to a STUDENT object.**
- On the other hand, generic operations to insert, delete, modify, or retrieve any kind of object are often included in the basic data model operations.
- Concepts to specify behaviour are fundamental to object-oriented data models but are also being incorporated in more traditional data models.

- For example, object-relational models extend the basic relational model to include such concepts, among others. In the basic relational data model, there is a provision to attach behaviour to the relations in the form of persistent stored modules, popularly known as stored procedures.

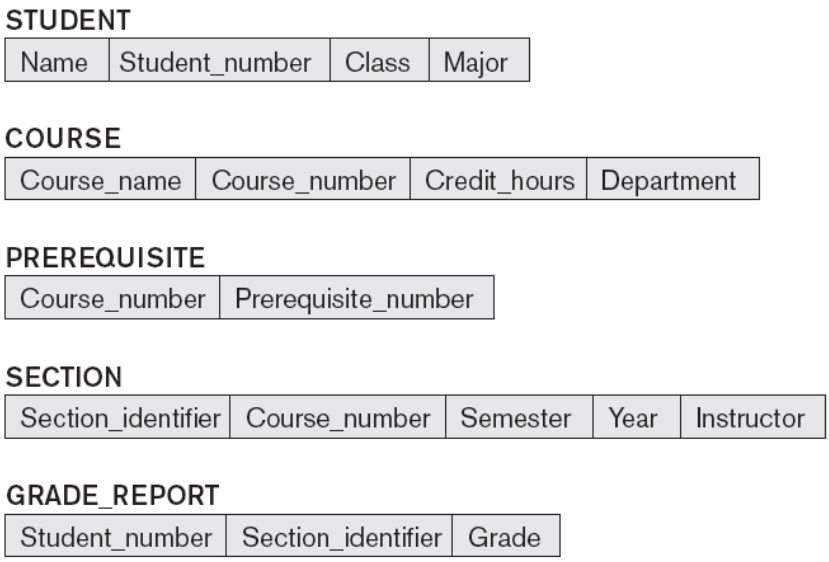
2.1.1 Categories of Data Models

- **High-level or conceptual data models** provide concepts that are close to the way many users perceive data.
- **Low-level or physical data models** provide concepts that describe the details of how data is stored on the computer storage media, typically magnetic disks. These models are generally meant for computer specialists, not for end users.
- **Representational (or implementation) data models** provide concepts that may be easily understood by end users but that are not too far removed from the way data is organized in computer storage.
- Representational data models hide many details of data storage on disk but can be implemented on a computer system directly.
- **Conceptual data models use concepts such as entities, attributes, and relationships.**
    * An **entity** represents a real-world object or concept, such as an employee or a project from the miniworld that is described in the database.
    * An **attribute** represents some property of interest that further describes an entity, such as the employee's name or salary.
    * A **relationship** among two or more entities represents an association among the entities, for example, a works-on relationship between an employee and a project.
- **Entity-Relationship model—a popular high-level conceptual data model**.
- Additional abstractions used for advanced modelling are generalization, specialization, and categories (union types).
- Representational or implementation data models are the models used most frequently in traditional commercial DBMSs. These include the widely used relational data model, as well as the so-called legacy data models—the network and hierarchical models—that have been widely used in the past.
- **Representational data models** represent data by using record structures and hence are sometimes **called record-based data models**.
- We can regard the **object data model** as an example of a new family of **higher-level implementation data models** that are closer to conceptual data models. A standard for object databases called the ODMG object model has been proposed by the Object Data Management Group (ODMG).
- 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.
    * An **access path** is a structure that makes the search **for particular database records** efficient.
    * An **index** is an **example of an access path** that allows direct access to data using an index term or a keyword. It is similar to the index of a book, except that it may be organized in a linear, hierarchical (tree-structured), or some other fashion.

2.1.2 Schemas, Instances, and Database State

- In any data model, it is important to distinguish between the description of the database and the database itself.
- **The description of a database is called the database schema**, which is specified during database design and is not expected to change frequently.
- Most data models have certain conventions for displaying schemas as diagrams. A displayed schema is called a schema diagram.
- Figure 2.1 shows a schema diagram for the database shown in Figure 1.2. The diagram displays the structure of each record type but not the actual instances of records.

Figure 2.1
Schema diagram for the database in Figure 1.2.



- We call each object in the schema—such as STUDENT or COURSE—a schema construct.
- A schema diagram displays only some aspects of a schema, such as the names of record types and data items, and some types of constraints.
- Other aspects are not specified in the schema diagram; for example, Figure 2.1 shows neither the data type of neither each data item, nor the relationships among the various files.
- Many types of constraints are not represented in schema diagrams. A constraint such as students majoring in computer science must take CS1310 before the end of their sophomore year is quite difficult to represent diagrammatically.
- The actual data in a database may change quite frequently. For example, the database shown in Figure 1.2 changes every time we add a new student or enters a new grade. **The data in the database at a particular moment in time is called a database state or snapshot.** It is also called the current set of **occurrences or instances** in the database.
- In a given database state, each schema construct has its own current set of instances; for example, the STUDENT construct will contain the set of individual student entities (records) as its instances.
- Many database states can be constructed to correspond to a particular database schema. Every time we insert or delete a record or change the value of a data item in a record, we change one state of the database into another state.
- The distinction between database schema and database state is very important.
- When we **define a new database**, we specify its **database schema** only to the DBMS. At this point, the corresponding database state is the **empty state** with no data.

- We get the *initial state* of the database when the database is first **populated or loaded** with the initial data. From then on, every time an update operation is applied to the database, we get another database state.
- At any point in time, the database has a current state.
- The DBMS is partly responsible for ensuring that every state of the database is a valid state—that is, a state that satisfies the structure and constraints specified in the schema. Hence, specifying a correct schema to the DBMS is extremely important and the schema must be designed with utmost care.
- The DBMS stores the **descriptions of the schema constructs and constraints—also called the meta-data—in the DBMS catalog** so that DBMS software can refer to the schema whenever it needs to.
- **The schema is sometimes called the intension, and a database state is called an extension of the schema.**
- Although, as mentioned earlier, the schema is not supposed to change frequently, it is not uncommon that changes occasionally need to be applied to the schema as the application requirements change. For example, we may decide that another data item needs to be stored for each record in a file, such as adding the Date_of_birth to the STUDENT schema in Figure 2.1. This is known as **schema evolution**.

2.2 Three-Schema Architecture and Data Independence

- Three of the four important characteristics of the database approach, listed in Section 1.3, are (1) use of a catalog to store the database description (schema) so as to make it self-describing, (2) insulation of programs and data (program-data and program-operation independence),
  (3) support of multiple user views,
  (4) Sharing of data and multiuser transaction processing.
- In this section, we specify architecture for database systems, called the three-schema architecture (also known as the **ANSI/SPARC architecture**) that was proposed to help achieve and visualize these characteristics.
- Then we discuss the concept of data independence further.

2.2.1 The Three-Schema Architecture

- The goal of the three-schema architecture, illustrated in Figure 2.2, is to separate the user applications from the physical database.

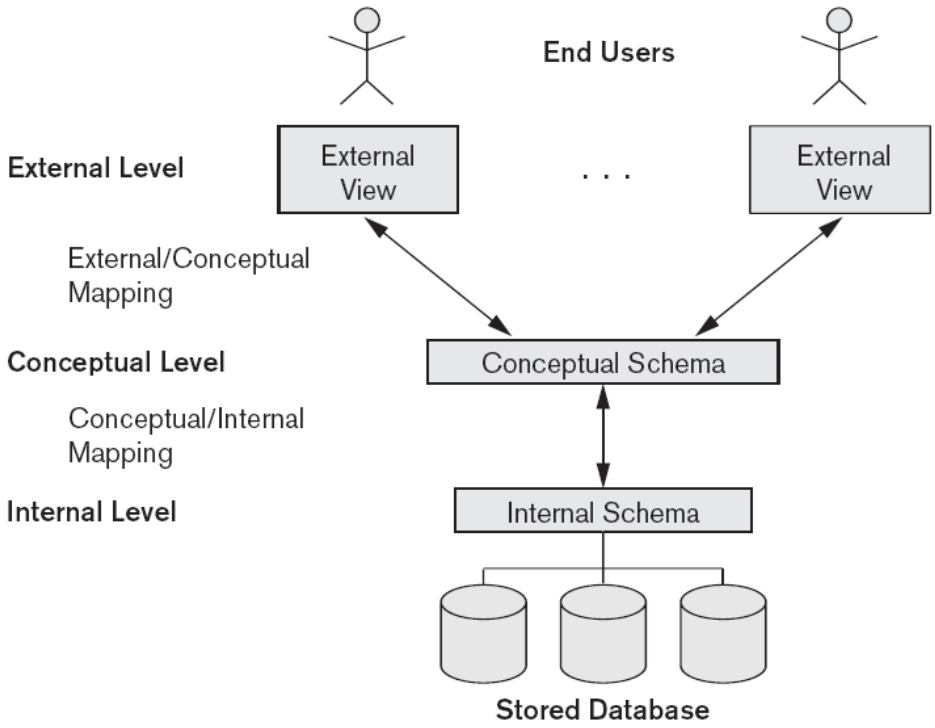


Figure 2.2
The three-schema architecture.

- In this architecture, schemas can be defined at the following three levels:
  - 1. The **internal level has an internal schema**, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.
  - 2. The **conceptual level** has a **conceptual schema**, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. Usually, a representational data model is used to describe the conceptual schema when a database system is implemented. This implementation conceptual schema is often based on a conceptual schema design in a high-level data model.
  - 3. The **external or view level** includes a number of external schemas or user views. 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. As in the previous level, each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high-level data model.
- In a DBMS based on the three-schema architecture, each user group refers to its own external schema. Hence, the DBMS must transform a request specified on an external schema into a request against the conceptual schema, and then into a request on the internal schema for processing over the stored database. If the request is database retrieval, the data extracted from the stored database must be reformatted to match the user's external view. **The processes of transforming requests and results between levels are called mappings.**
- These mappings may be time-consuming, so some DBMSs—especially those that are meant to support small databases—do not support external views.

2.2.2 Data Independence

- The three-schema architecture can be used to further explain the concept of **data independence**, which can be **defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level.**
- We can define two types of data independence:
  1. **Logical data independence** is the **capacity to change the conceptual schema without having to change external schemas or application programs.** We may change the conceptual schema to expand the database (by adding a record type or data item), to change constraints, or to reduce the database (by removing a record type or data item).
  2. **Physical data independence is the capacity to change the internal schema without having to change the conceptual schema.** Hence, the external schemas need not be changed as well. Changes to the internal schema may be needed because some physical files were reorganized—for example, by creating additional access structures—to improve the performance of retrieval or update.
- Generally, **physical data independence exists** in most databases and file environments where physical details such as the exact location of data on disk, and hardware details of storage encoding, placement, compression, splitting, merging of records, and so on are hidden from the user. Applications remain unaware of these details.
- On the other hand, **logical data independence is harder to achieve** because it allows structural and constraint changes without affecting application programs—a much stricter requirement.
- The three-schema architecture can make it easier to achieve true data independence, both physical and logical.

2.3 Database Languages and Interfaces

2.3.1 DBMS Languages

- Once the design of a database is completed and a DBMS is chosen to implement the database, the first step is to **specify conceptual and internal schemas** for the database **and** any **mappings** between the two.
- In many DBMSs where **no strict separation** of levels is maintained, one language, called the **data definition language (DDL)**, **is used by the DBA and by database designers to define both schemas (conceptual and internal schemas).**
- 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.
- In DBMSs where **a clear separation** is maintained between the conceptual and internal levels, the **DDL** is used to **specify the conceptual schema only**.
- Another language, the **storage definition language (SDL)**, **is used to specify the internal schema. The mappings between the two schemas may be specified in either one of these languages.**
- 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. These permit the DBA staff to control indexing choices and mapping of data to storage.

- For true three-schema architecture, we would need a third language, the **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.
- Once the database schemas are compiled and the database is populated with data, users must have some means to manipulate the database. Typical manipulations include retrieval, insertion, deletion, and modification of the data. The DBMS provides a set of operations or a language called the **data manipulation language (DML)** for these purposes.
- In current DBMSs, the preceding types of languages are usually not considered distinct languages; rather, a comprehensive integrated language is used that includes constructs for conceptual schema definition, view definition, and data manipulation.
- A typical example of a comprehensive database language is the SQL relational database language, which represents a combination of DDL, VDL, and DML, as well as statements for constraint specification, schema evolution, and other features. The SDL was a component in early versions of SQL but has been removed from the language to keep it at the conceptual and external levels only.
- There are two main types of DMLs.
    * A **high-level or nonprocedural DML** can be used on its own to specify complex database operations concisely. Many DBMSs allow high-level DML statements either to be entered interactively from a display monitor or terminal or to be embedded in a general-purpose programming language. In the latter case, DML statements must be identified within the program so that they can be extracted by a pre-compiler and processed by the DBMS.
    * 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. Therefore, it needs to use programming language constructs, such as looping, to retrieve and process each record from a set of records. Low-level DMLs are also called **record-at-a-time DMLs** because of this property.
- A DML is designed for the **hierarchical model**, which is a **low-level DML** that uses commands such as GET UNIQUE, GET NEXT, or GET NEXT WITHIN PARENT to navigate from record to record within a hierarchy of records in the database.
- **High-level DMLs**, such as **SQL**, can specify and retrieve many records in a single DML statement; therefore, they are called **set-at-a-time or set-oriented DMLs**. A query in a high-level DML often specifies which data to retrieve rather than how to retrieve it; therefore, such languages are also called declarative.
- 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 general, both retrieval and update commands of a high-level DML may be used interactively and are hence considered part of the query language.
- Casual end users typically use a high-level query language to specify their requests, whereas programmers use the DML in its embedded form.

- For naive and parametric users, there usually are user-friendly interfaces for interacting with the database; these can also be used by casual users or others who do not want to learn the details of a high-level query language.

2.3.2 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. Menus do away with the need to memorize the specific commands and syntax of a query language; rather, the query is composed step-by step by picking options from a menu that is displayed by the system. Pull-down menus are a very popular technique in **Web-based user interfaces**. They are also often used in **browsing interfaces**, which allow a user to look through the contents of a database in an exploratory and unstructured manner.

- **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. Many DBMSs have **forms specification languages**, which are special languages that help programmers, specify such forms. **SQL*Forms** is a form-based language that specifies queries using a form designed in conjunction with the relational database schema. **Oracle Forms** is a component of the Oracle product suite that provides an extensive set of features to design and build applications using forms. Some systems have utilities that define a form by letting the end user interactively construct a sample form on the screen.

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

- **Natural Language Interfaces:** These interfaces accept requests written in English or some other language and attempt to understand them. A natural language interface usually has its own schema, which is similar to the database conceptual schema, as well as a dictionary of important words. Today, we see search engines that accept 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:** 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. The speech input is detected using a library of predefined words and used to set up the parameters that are supplied to the queries. For output, a similar conversion from text or numbers into speech takes place.

- **Interfaces for Parametric User: 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. Systems analysts and programmers design and implement a special interface for each known class of naive users. Usually a small set of abbreviated commands is included, with the goal of minimizing the number of keystrokes required for each request. For example, function keys in a terminal can be programmed to initiate various commands. This allows the parametric user to proceed with a minimal number of keystrokes.

- **Interfaces for the DBA:** Most database systems contain **privileged commands** that can be used only by the DBA staff. These include commands for creating accounts, setting system parameters, granting account authorization, changing a schema, and reorganizing the storage structures of a database.

2.4 The Database System Environment

A DBMS is a complex software system. In this section, we discuss the types of software components that constitute a DBMS and the types of computer system software with which the DBMS interacts.

2.4.1 DBMS Component Modules

- Figure 2.3 illustrates, in a simplified form, the typical DBMS components. The figure is divided into two parts. The top part of the figure refers to the various users of the database environment and their interfaces. The lower part shows the internals of the DBMS responsible for storage of data and processing of transactions.
- The database and the DBMS catalog are usually stored on disk. Access to the disk is controlled primarily by the **operating system (OS)**, which schedules disk read/write.
- Many DBMSs have their **own buffer management module** to schedule disk read/write, because this has a considerable effect on performance. Reducing disk read/write improves performance considerably.
- 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.

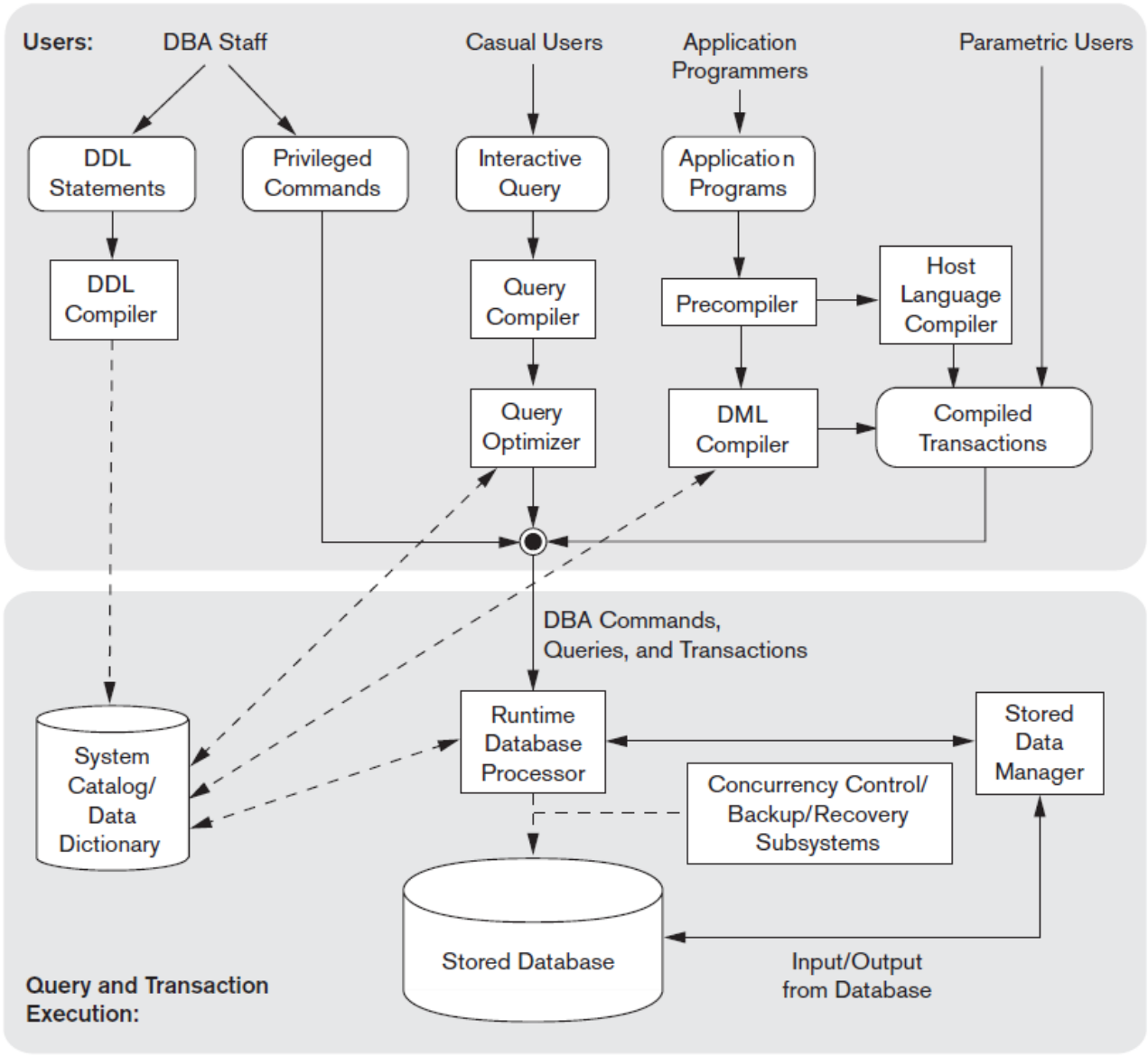


Figure 2.3
Component modules of a DBMS and their interactions.

- Let us consider the top part of Figure 2.3 first. It shows interfaces for the different users as follows:
  - * DBA staff:
    - work with **interactive interfaces** to formulate queries
    - 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. The catalog includes information such as the names and sizes of files, names and data types of data items, storage details of each file, mapping information among schemas, and constraints.
  - * Casual users:
    - Who work with **interactive interfaces** to formulate queries.
    - Menu-based or form-based interaction is used to generate the interactive query.

- These queries are parsed and validated for correctness of the query syntax, the names of files and data elements, and so on by a **query compiler** that **compiles them into an internal form**.
- This internal query is subjected to **query optimization** is concerned with the **rearrangement** and possible **reordering** of operations, **elimination** of **redundancies**, and use of correct algorithms and indexes during execution. It consults the system catalog for statistical and other physical information about the stored data and generates executable code that performs the necessary operations for the query and makes calls on the runtime processor.

* Application programmers
  - who create programs using some host programming languages such as Java, C, or C++
  - These are submitted to a **precompiler**. The precompiler **extracts DML commands** from an application program written in a host programming language. These commands are **sent to the DML compiler** for compilation into object code for database access. The **rest of the program is sent to the host language compiler**. The object codes for the DML commands and the rest of the program are **linked**, forming a canned transaction whose **executable code includes calls to the runtime database processor**.
* Parametric users
  - Who does data entry work by supplying parameters to predefined transactions.
  - An example is a bank withdrawal transaction where the account number and the amount may be supplied as parameters.

- In the lower part of Figure 2.3, the 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.
  * It also works with the stored data manager, which in turn uses basic operating system services for carrying out low-level input/output (read/write) operations between the disk and main memory.
  * Handles management of buffers in the main memory.
  * Concurrency control and backup and recovery systems integrated into the working of the runtime database processor for purposes of transaction management.
- It is now common to have the **client program** that accesses the DBMS running on a separate computer from the computer on which the database resides. The former is called the **client computer** running DBMS client software and the latter is called the **database server**. In some cases, the client accesses a middle computer, called the **application server**, which in turn accesses the database server.
- Figure 2.3 is not meant to describe a specific DBMS; rather, it illustrates typical DBMS modules. The DBMS interacts with the operating system when disk accesses—to the database or to the catalog—are needed.
- If the computer system is shared by many users, the OS will schedule DBMS disk access requests and DBMS processing along with other processes. On the other hand, if the computer system is mainly dedicated to running the database server, the DBMS will control main memory buffering of

disk pages. The DBMS also interfaces with compilers for general purpose host programming languages, and with application servers and client programs running on separate machines through the system network interface.

2.4.2 Database System Utilities

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

  ■ **Loading**. A loading utility is used to load existing data files—such as text files or sequential files—into the database. Usually, the current (source) format of the data file and the desired (target) database file structure is specified to the utility, which then automatically reformats the data and stores it in the database. With the proliferation of DBMSs, transferring data from one DBMS to another is becoming common in many organizations. Some vendors are offering products that generate the appropriate loading programs, given the existing source and target database storage descriptions (internal schemas). Such tools are also called **conversion tools**. Ex: hierarchical DBMS called **IMS (IBM)** and for many network DBMSs including **IDMS (Computer Associates)**, **SUPRA (Cincom)**, and **IMAGE (HP)**, the vendors or **third-party** companies are making a variety of conversion tools available (e.g., **Cincom's SUPRA Server** SQL) to transform data into the relational model.

  ■ **Backup**: A backup utility creates a backup copy of the database, usually by dumping the entire database onto tape or other mass storage medium. The backup copy can be used to restore the database in case of catastrophic disk failure. Incremental backups are also often used, where only changes since the previous backup are recorded. Incremental backup is more complex, but saves storage space.

  ■ **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.

2.4.3 Tools, Application Environments, and Communications Facilities

- Other tools are often available to database designers, users, and the DBMS. **CASE tools** are used in the **design phase** of database systems.
- Another tool that can be quite useful in large organizations is an **expanded data dictionary (or data repository)** system. In addition to storing catalog information about schemas and constraints, the data dictionary stores other information, such as design decisions, usage standards, application program descriptions, and user information. Such a system is also called an **information repository**.

This information can be accessed directly by users or the DBA when needed. A data dictionary utility is similar to the DBMS catalog, but it includes a wider variety of information and is accessed mainly by users rather than by the DBMS software.

- **Application development environments**, such as **PowerBuilder (Sybase)** or **JBuilder (Borland),** have been quite popular. These systems provide an environment for developing database applications and include facilities that help in many facets of database systems, including database design, GUI development, querying and updating, and application program development.
- The DBMS also needs to interface with **communications software**, whose function is to allow users at locations remote from the database system site to access the database through computer terminals, workstations, or personal computers. These are connected to the database site through data communications hardware such as Internet routers, phone lines, long-haul networks, local networks, or satellite communication devices. The integrated DBMS and data communications system is called a DB/DC system.

2.5 Centralized and Client/Server Architectures for DBMSs

2.5.1 Centralized DBMSs Architecture

- Earlier 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. The reason was that most users accessed such systems via computer terminals that did not have processing power and only provided display capabilities. Therefore, all processing was performed remotely on the computer system, 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.
- As prices of hardware declined, most users replaced their terminals with PCs and workstations. At first, database systems used these computers similarly to how they had used display terminals, so that the DBMS itself was still a centralized DBMS in which all the DBMS functionality, application program execution, and user interface processing were carried out on one machine.
- Figure 2.4 illustrates the physical components in a centralized architecture. Gradually, DBMS systems started to exploit the available processing power at the user side, which led to client/server DBMS architectures.

2.5.2 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, data base 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.
- Another machine can be designated as a printer server by being connected to various printers; all print requests by the clients are forwarded to this machine.
- Web servers or e-mail servers also fall into the specialized server category.
- The resources provided by specialized servers can be accessed by many client machines.

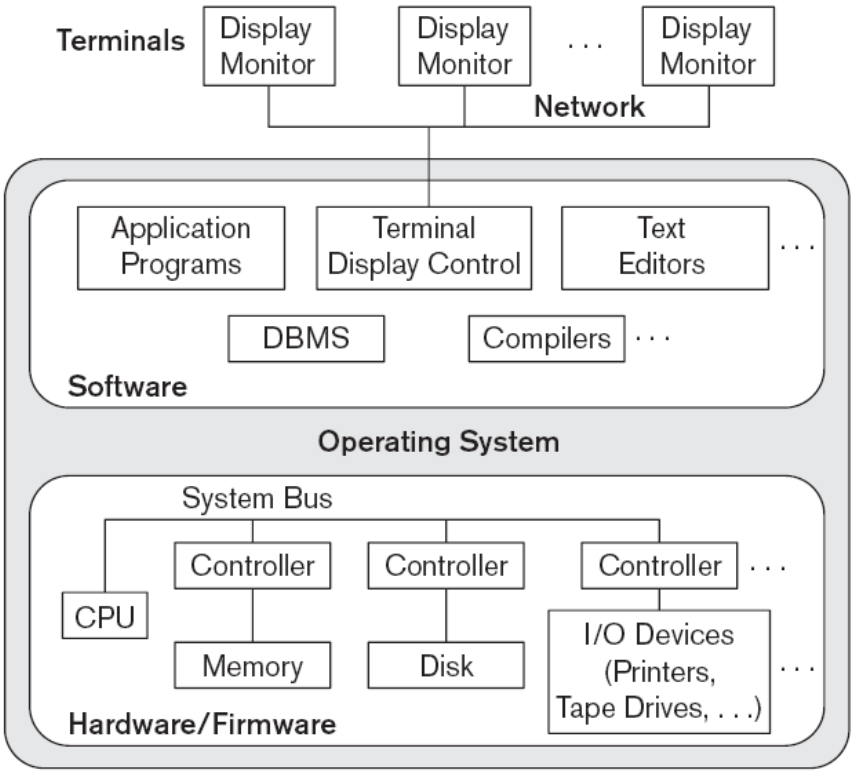


Figure 2.4
A physical centralized architecture.

- The client machines provide the user with the appropriate interfaces to utilize these servers, as well as with local processing power to run local applications. This concept can be carried over to other software packages, with specialized programs—such as a CAD (computer-aided design) package—being stored on specific server machines and being made accessible to multiple clients. Figure 2.5 illustrates client/server architecture at the logical level; Figure 2.6 is a simplified diagram that shows the physical architecture. Some machines would be client sites only (for example, diskless workstations or workstations/PCs with disks that have only client software installed).

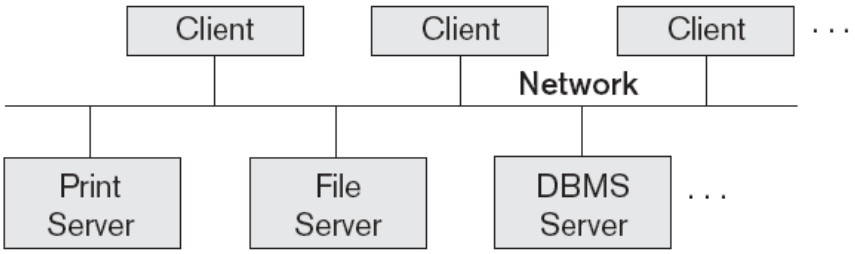


Figure 2.5
Logical two-tier client/server architecture.

- Other machines would be dedicated servers, and others would have both client and server functionality.
- The concept of client/server architecture assumes an underlying framework that consists of many PCs and workstations as well as a smaller number of mainframe machines, connected via LANs and other types of computer networks.
- A client in this framework is typically a user machine that provides user interface capabilities and local processing. When a client requires access to additional functionality— such as database access—that does not exist at that machine, it connects to a server that provides the needed functionality.

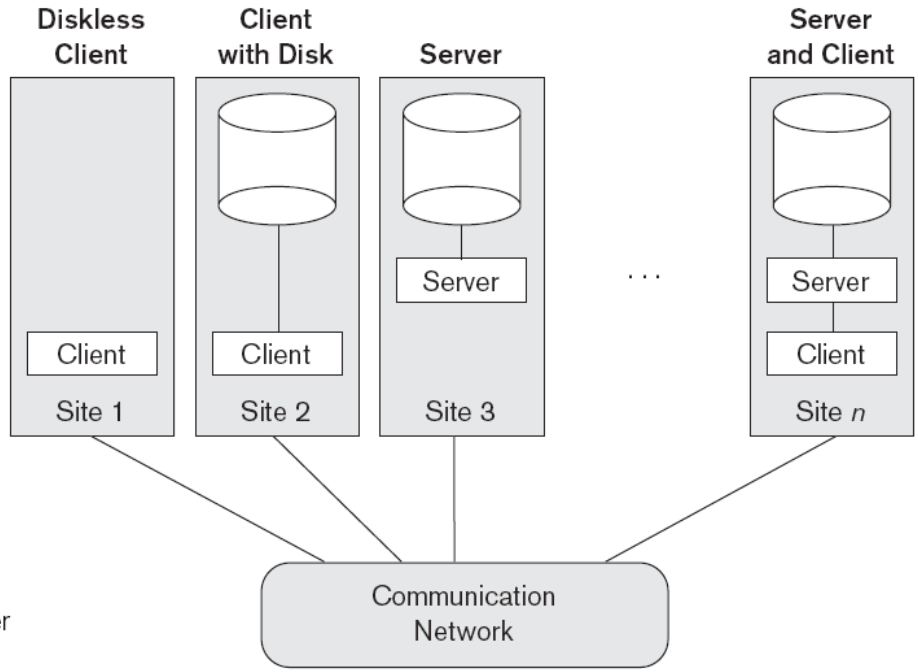


Figure 2.6
Physical two-tier client/server architecture.

- A server is a system containing both hardware and software that can provide services to the client machines, such as file access, printing, archiving, or database access.
- In general, some machines install only client software, others only server software, and still others may include both client and server software, as illustrated in Figure 2.6. However, it is more common that client and server software usually run on separate machines. Two main types of basic DBMS architectures were created on this underlying client/server framework: two-tier and three-tier.

2.5.3 Two-Tier Client/Server Architectures for DBMSs

- In relational database management systems (RDBMSs), many of which started as centralized systems, the system components that were first moved to the client side were the user interface and application programs.
- Because SQL provided a standard language for RDBMSs, this created a logical dividing point between client and server. Hence, the query and transaction functionality related to SQL processing remained on the server side.
- In such architecture, the server is often called a **query server** or **transaction server** because it provides these two functionalities. In an RDBMS, the server is also often called an **SQL server**.
- The user interface programs and application programs can run on the client side.
- When DBMS access is required, the program establishes a connection to the DBMS (which is on the server side); once the connection is created, the client program can communicate with the DBMS. A standard called **Open Database Connectivity (ODBC)** provides an **application programming interface (API)**, 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.
- A related standard for the Java programming language, called JDBC, has also been defined. This allows Java client programs to access one or more DBMSs through a standard interface.
- The different approach to two-tier client/server architecture was taken by some object-oriented DBMSs, where the software modules of the DBMS were divided between client and server in a more integrated way. For example, the **server level** may include the part of the DBMS software responsible for handling data storage on disk pages, local concurrency control and recovery, buffering and caching of disk pages, and other such functions. Meanwhile, the **client level** may handle the user interface; data dictionary functions; DBMS interactions with programming language compilers; global query optimization, concurrency control, and recovery across multiple servers; structuring of complex objects from the data in the buffers; and other such functions. In this approach, the client/server interaction is more tightly coupled and is done internally by the DBMS modules.
- In such client/server architecture, the server has been called a **data server** because it provides data in disk pages to the client. This data can then be structured into objects for the client programs by the client-side DBMS software.
- The architectures described here are called two-tier architectures because the software components are distributed over two systems: client and server.
- Advantages of this architecture are: simplicity and seamless compatibility with existing systems.
- The emergence of the Web changed the roles of clients and servers, leading to the three-tier architecture.

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

- Many Web applications use an architecture called the three-tier architecture, which adds an intermediate layer between the client and the database server, as illustrated in Figure 2.7(a).

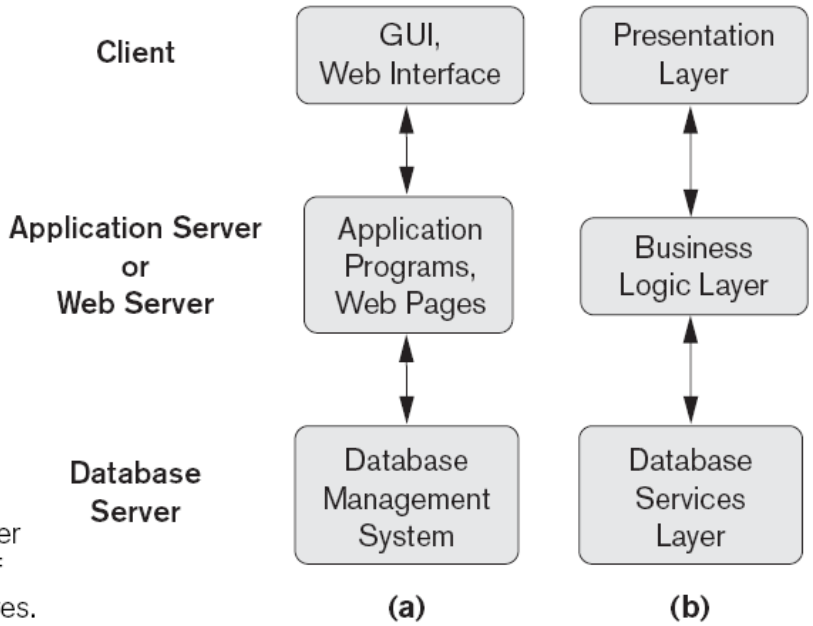


Figure 2.7
Logical three-tier client/server architecture, with a couple of commonly used nomenclatures.

- This intermediate layer or **middle tier** is called the **application server or the Web server,** depending on the application. This server plays an intermediary role by running application programs and storing business rules (procedures or constraints) that are used to access data from the database server.
- It can also improve database security by checking a client's credentials before forwarding a request to the database server.
- Clients contain GUI interfaces and some additional application-specific business rules.
- The intermediate server accepts requests from the client, processes the request and sends database queries and commands to the database server, and then acts as a conduit for passing (partially) processed data from the database server to the clients, where it may be processed further and filtered to be presented to users in GUI format. Thus, the user interface, application rules, and data access act as the three tiers.
- Figure 2.7(b) shows another architecture used by database and other application package vendors.
- The presentation layer displays information to the user and allows data entry.
- The business logic layer handles intermediate rules and constraints before data is passed up to the user or down to the DBMS.
- The bottom layer includes all data management services. The middle layer can also act as a Web server, which retrieves query results from the database server and formats them into dynamic Web pages that are viewed by the Web browser at the client side.
- Other architectures have also been proposed. It is possible to divide the layers between the user and the stored data further into finer components, thereby giving rise to n-tier architectures; where n may be four or five tiers. Typically, the business logic layer is divided into multiple layers.
- Advances in encryption and decryption technology make it safer to transfer sensitive data from server to client in encrypted form, where it will be decrypted. This technology gives higher levels of data security, but the network security issues remain a major concern.
- Various technologies for data compression also help to transfer large amounts of data from servers to clients over wired and wireless networks.

2.6 Classification of Database Management Systems

- **First criterion is based on data model** on which the DBMS is based.
    * The main data model used in many current commercial DBMSs is the relational **data model**. The basic relational data model represents a database as a collection of tables, where each table can be stored as a separate file.
    * The **object data model** has been implemented in some commercial systems but has not had widespread use. The object data model defines a database in terms of objects, their properties, and their operations. Objects with the same structure and behavior belong to a class, and classes are organized into hierarchies (or acyclic graphs). The operations of each class are specified in terms of predefined procedures called methods.
    * Many legacy applications still run on database systems based on the **hierarchical and network data models.** Examples of **hierarchical DBMSs include IMS (IBM) and some other systems like System 2K (SAS Inc.) and TDMS**. IMS is still used at governmental and industrial installations, including hospitals and banks, although many of its users have converted to relational systems. The **network data model was used by many vendors and**

the resulting products like IDMS (Cullinet—now Computer Associates), DMS 1100 (Univac—now Unisys), IMAGE (Hewlett-Packard), VAXDBMS (Digital—then Compaq and now HP), and SUPRA (Cincom) still have a following and their user groups have their own active organizations.

- **We can categorize DBMSs based on the data model: relational, object, object-relational, hierarchical, network, and other.**
- More recently, some experimental DBMSs are **based on the XML (eXtended Markup Language) model, which is a tree-structured (hierarchical) data model.**
- These have been called native XML DBMSs. Several commercial relational DBMSs have added XML interfaces and storage to their products.

- The **second criterion** used to classify DBMSs is the **number of users** supported by the system. **Single-user** systems support only one user at a time and are mostly used with PCs. **Multiuser** systems, which include the majority of DBMSs, support concurrent multiple users.

- The **third criterion** is the **number of sites** over which the database is distributed. A DBMS is centralized if the data is stored at a single computer site. A **centralized DBMS** can support multiple users, but the DBMS and the database reside totally at a single computer site. A **distributed DBMS (DDBMS)** can have the actual database and DBMS software distributed over many sites, connected by a computer network.
- **Homogeneous DDBMSs** use the same DBMS software at all the sites, whereas **heterogeneous DDBMSs** can use different DBMS software at each site.

- The **fourth criterion is cost**. It is difficult to propose a classification of DBMSs based on cost. Today we have **open source (free) DBMS products like MySQL and PostgreSQL** that are supported by third-party vendors with additional services.
- The **main RDBMS products** are available as free examination 30-day copy versions as well as personal versions, which may cost under **$100** and allow a fair amount of functionality. The giant systems are being sold in modular form with components to handle distribution, replication, parallel processing, mobile capability, and so on, and with a large number of parameters that must be defined for the configuration.
- They are sold in the **form of licenses**—site licenses allow **unlimited use** of the database system with any number of copies running at the customer site.
- Another type of license limits the number of concurrent users or the number of user seats at a location. Standalone **single user versions of some systems like Microsoft Access** are sold per copy or included in the overall configuration of a desktop or laptop. In addition, data warehousing and mining features, as well as support for additional data types, are made available at extra cost.
- It is possible to pay millions of dollars for the installation and maintenance of large database systems annually.
- We can also classify a DBMS on the basis of the types of access path options for storing files. One well-known family of DBMSs is based on inverted file structures.
- Finally, a DBMS can be general purpose or **special purpose**. When performance is a primary consideration, a special-purpose DBMS can be designed and built for a specific application; such a system cannot be used for other applications without major changes. Many airline reservations and

telephone directory systems developed in the past are special-purpose DBMSs. These fall into the category of **online transaction processing (OLTP) systems**, which must support a large number of concurrent transactions without imposing excessive delays.

- Two older, historically important data models, now known as legacy data models, are the network and hierarchical models. The network model represents data as record types and also represents a limited type of **1: N relationship, called a set type**.
- A 1: N, or one-to-many, relationship relates one instance of a record to many record instances using some pointer linking mechanism in these models. Figure 2.8 shows a network schema diagram for the database of Figure 2.1, where record types are shown as rectangles and set types are shown as labeled directed arrows.

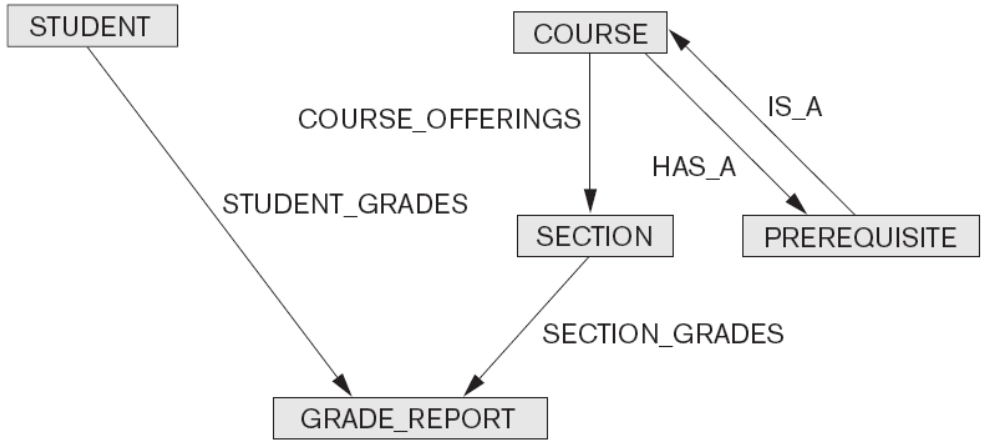


Figure 2.8
The schema of Figure 2.1 in network model notation.

Data Modelling Using the Entity-Relationship (ER) Model

- **Entity-Relationship (ER) model** is a popular **high-level conceptual data model.**
- It is used for the conceptual design of database applications, and many database design tools employ its concepts.

3.1 Using High-Level Conceptual Data Models for Database Design

- Figure 3.1 shows a simplified overview of the database design process.

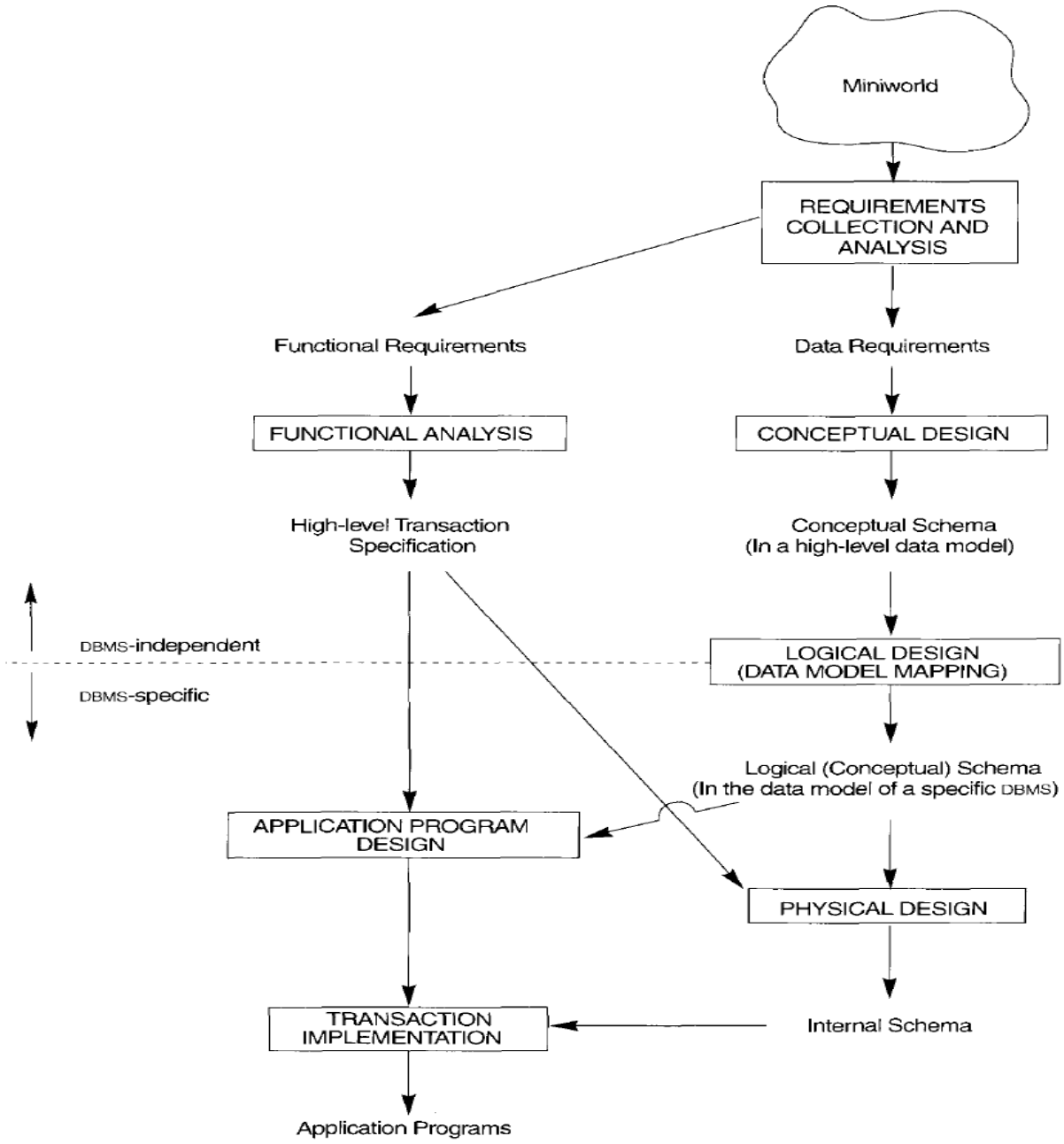


FIGURE 3.1 A simplified diagram to illustrate the main phases of database design.

- The first step shown is **requirements collection and analysis**. During this step, the database designers interview prospective database users to understand and document their **data requirements**. The result of this step is a concisely written set of users' requirements.

- These requirements should be specified in as detailed and complete a form as possible. In parallel with specifying the data requirements, it is useful to specify the known **functional requirements** of the application. These consist of the user defined operations (or transactions) that will be applied to the database, including both retrievals and updates.
- In software design, it is common to use data flow diagrams, sequence diagrams, scenarios, and other techniques to specify functional requirements.
- Once the requirements have been collected and analysed, 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**; these are expressed using the concepts provided by the high-level data model. Because these concepts do not include implementation details, they are usually **easier to understand** and can be used to communicate with nontechnical users.
- The high-level conceptual schema can also be **used as a reference to ensure that all users' data requirements are met and that the requirements do not conflict**. This approach enables database designers to concentrate on specifying the properties of the data, without being concerned with storage and implementation details. This makes it is easier to create a good conceptual database design.
- 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. This also serves to confirm that the conceptual schema meets all the identified functional requirements. Modifications to the conceptual schema can be introduced if some functional requirements cannot be specified using the initial schema.
- The next step in database design is the actual implementation of the database, using a commercial DBMS. Most current commercial DBMSs use an implementation data model—such as the relational or the object-relational database model—so **the conceptual schema is transformed from the high-level data model into the implementation data model. This step is called logical design or data model mapping**; its result is a database schema in the implementation data model of the DBMS.
- 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.

3.2 An Example Database Application

- In this section, we describe a sample database application, called COMPANY, which serves to illustrate the basic ER model concepts and their use in schema design.
- We list the data requirements for the database here, and then create its conceptual schema step-by-step as we introduce the modeling concepts of the ER model. The COMPANY database keeps track of a company's employees, departments, and projects.
- Suppose that after the requirements collection and analysis phase, the database designers provide the following description of the miniworld—the part of the company that will be represented in the database.

■ The company is organized into departments. Each department has a unique name, a unique number, and a particular employee who manages the department. We keep track of the start date when that employee began managing the department. A department may have several locations.

■ A department controls a number of projects, each of which has a unique name, a unique number, and a single location.

■ We store each employee's name, Social Security number, address, salary, 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. We keep track of the current number of hours per week that an employee works on each project. We also keep track of the direct supervisor of each employee (who is another employee).

■ We want to keep track of the dependents of each employee for insurance purposes. We keep each dependent's first name, sex, birth date, and relationship to the employee.

- Figure 3.2 shows how the schema for this database application can be displayed by means of the graphical notation known as ER diagrams. This figure will be explained gradually as the ER model concepts are presented.

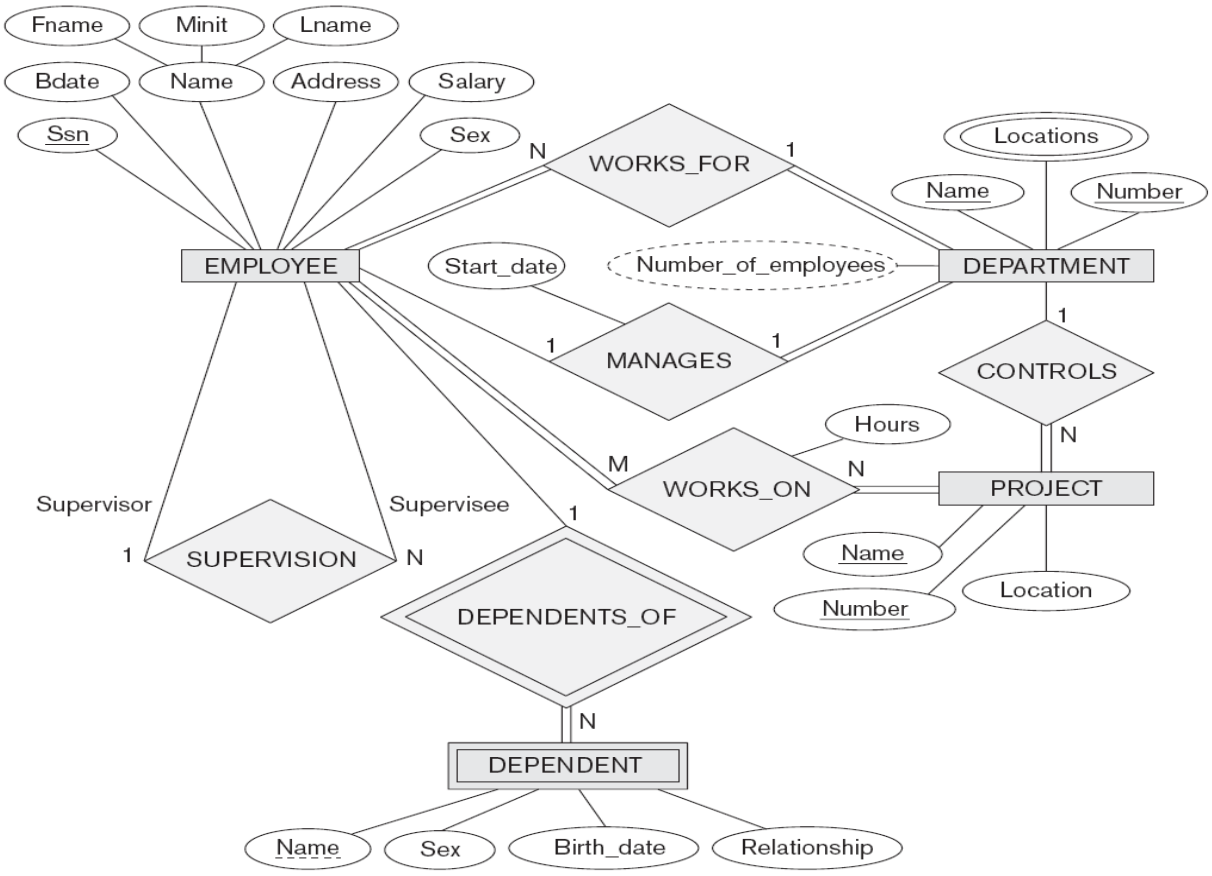


Figure 3.2 An ER schema diagram for the COMPANY database

Note: Social Security number, or SSN, is a unique nine-digit identifier assigned to each individual in the United States to keep track of his or her employment, benefits, and taxes. Other countries may have similar identification schemes, such as personal identification card numbers

3.3 Entity Types, Entity Sets, Attributes, and Keys

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

3.3.1 Entities and Attributes

- **Entities and Their Attributes**: The basic object that the ER model represents is an entity, which 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).
- Each **entity has attributes**—the particular properties that describe it. For example, an EMPLOYEE entity may be described by the employee's name, age, address, salary, and job.
- A particular entity will have a. value for each of its attributes. The attribute values that describe each entity become a major part of the data stored in the database.
- Figure 3.3 shows two entities and the values of their attributes. The EMPLOYEE entity e1 has four attributes: Name, Address, Age, and Home_phone; their values are 'John Smith,' '2311 Kirby, Houston, Texas 77001', '55', and '713-749-2630', respectively.
- The COMPANY entity c1 has three attributes: Name, Headquarters, and President; their values are 'Sunco Oil', 'Houston', and 'John Smith', respectively.
- **Several types of attributes** occur in the ER model: **simple versus composite**, **singlevalued versus multivalued, and stored versus derived**.

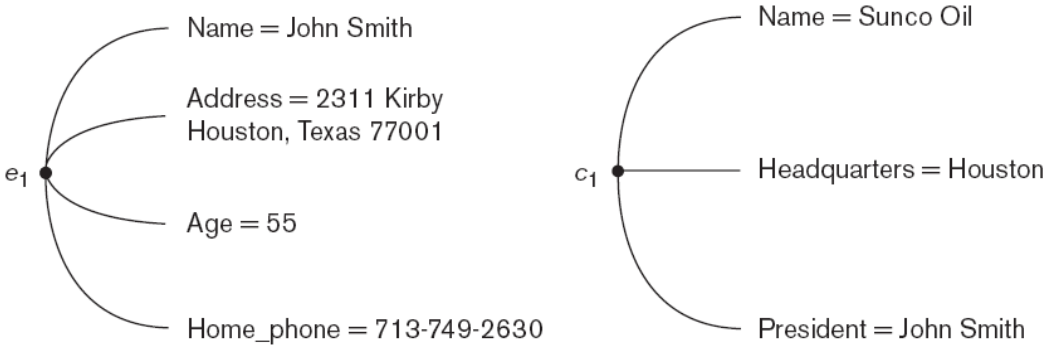


Figure 3.3 Two Entities, EMPLOYEEE e_1 and COMPANY c_1 and their attributes

Composite versus Simple (Atomic) Attributes:

- **Composite attributes** can be divided into smaller subparts, which represent more basic attributes with independent meanings. For example, the Address attribute of the EMPLOYEE entity shown in Figure 3.3 can be subdivided into Street_address, City, State, and Zip, with the values '2311 Kirby', 'Houston', 'Texas', and '77001.'
- Note: Zip Code is the name used in the United States for a five-digit postal code, such as 76019, which can be extended to nine digits, such as 76019-0015. We use the five-digit Zip in our examples.
- Attributes that are not divisible are called **simple or atomic attributes**.
- Composite attributes can form a hierarchy; for example, Street_address can be further subdivided into three simple component attributes: Number, Street, and Apartment_number, as shown in Figure 3.4.

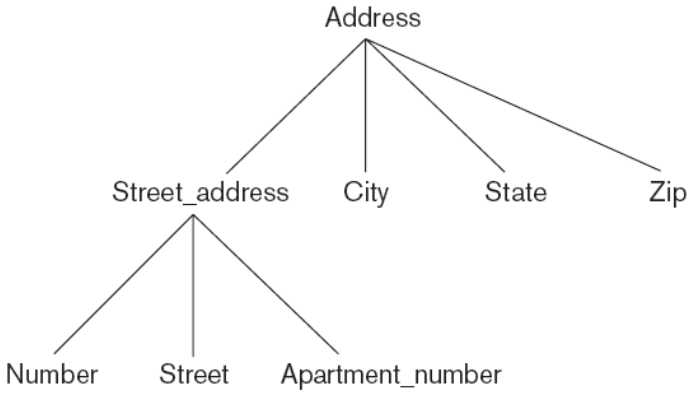


Figure 3.4 A hierarchy of composite attributes

- The value of a composite attribute is the concatenation of the values of its component simple attributes.
- Composite attributes are useful to model situations in which a user sometimes refers to the composite attribute as a unit but at other times refers specifically to its components. If the composite attribute is referenced only as a whole, there is no need to subdivide it into component attributes. For example, if there is no need to refer to the individual components of an address (Zip Code, street, and so on), then the whole address can be designated as a simple attribute.

Single-Valued versus Multivalued Attributes:

- Most attributes have a single value for a particular entity; such attributes are called **single-valued**. For example, Age is a single-valued attribute of a person.
- In some cases, an attribute can have a set of values for the same entity—for instance, a Colors attribute for a car, or a College_degrees attribute for a person. Cars with one color have a single value, whereas two-tone cars have two color values. Similarly, one person may not have a college degree, another person may have one, and a third person may have two or more degrees; therefore, different people can have different numbers of values for the College_degrees attribute. Such attributes are called **multivalued**.
- A multivalued attribute may have lower and upper bounds to constrain the number of values allowed for each individual entity. For example, the Colors attribute of a car may be restricted to have between one and three values, if we assume that a car can have three colors at most.

Stored versus Derived Attribute:

- 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 (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**.
- 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.

NULL Values:

- In some cases, a particular entity may **not have an applicable value for an attribute**. For example, the Apartment_number attribute of an address applies only to addresses that are in apartment buildings and not to other types of residences, such as single-family homes.
- Similarly, a College_degrees attribute applies only to people with college degrees. For such situations, a special value called NULL is created.
- An address of a single-family home would have NULL for its Apartment_number attribute, and a person with no college degree would have NULL for College_degrees. The meaning of this type of NULL is **not applicable**.
- NULL can also be used if we **do not know the value of an attribute** for a particular entity—for example, if we do not know the home phone number of 'John Smith' in Figure 3.3., The meaning of this type of NULL is **unknown**.
- The unknown category of NULL can be further classified into two cases. The first case arises 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. The second case arises when it is **not known whether the attribute value exists**—for example, if the Home_phone attribute of a person is NULL.

Complex Attributes:

- In general, composite and multivalued attributes can be nested arbitrarily. We can represent arbitrary nesting by grouping components of a **composite attribute between parentheses ()** and separating the components with commas, and by displaying **multivalued attributes between braces { }**. Such attributes are called **complex attributes**. For example, if a person can have more than one residence and each residence can have a single address and multiple phones, an attribute Address_phone for a person can be specified as shown in Figure 3.5. 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))
}

Figure 3.5 A complex attribute: Address_phone

3.3.2 Entity Types, Entity Sets, Keys, and Value Sets

Entity Types and Entity Sets.

- A database usually contains groups of entities that are similar. For example, a company employing hundreds of employees may want to store similar information concerning each of the employees.
- These employee entities share the same attributes, but each entity has its own value(s) for each attribute.
- 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. Figure 3.6 shows two entity types: EMPLOYEE and COMPANY and a list of some of the attributes for each. A few individual entities of each type are also illustrated, along with the values of their 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. For example, EMPLOYEE refers to both a type of entity as well as the current set of all employee entities in the database.

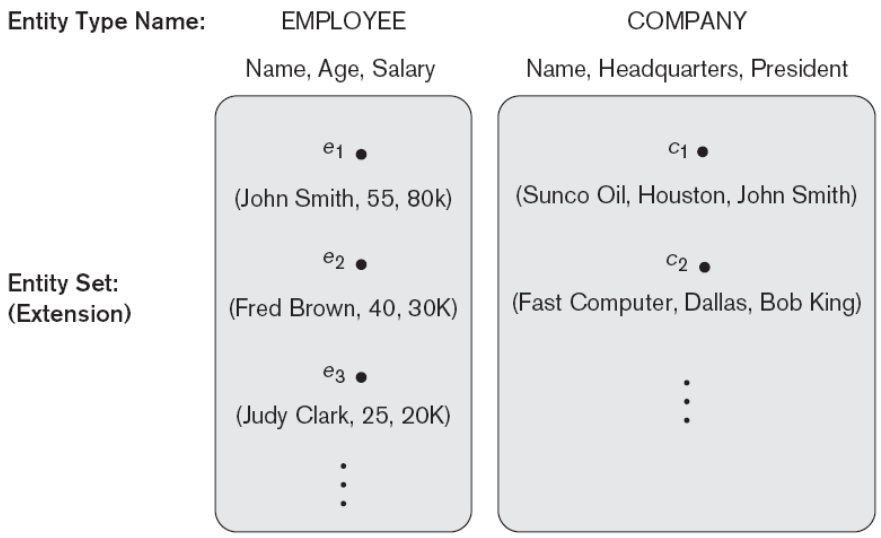


Figure 3.6 Two entity types, EMPLOYEE and COMPANY, and some member entities of each.

- An **entity type** is represented in ER diagrams (see Figure 3.2) **as a rectangular box** 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**. Figure 3.7(a) shows a CAR entity type in this notation.

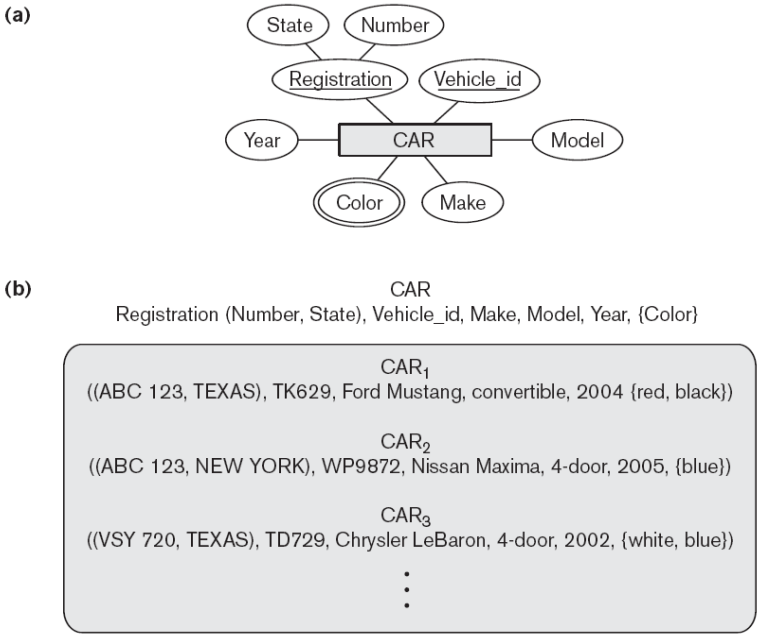


Figure 3.7 The CAR entity type with two key attributes, Registration and Vehicle_id.
(a) ER diagram notation (b) Entity set with three entities.

- **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**.

Key Attributes of an Entity Type:

- An important constraint on the entities of an entity type is the key or uniqueness constraint on attributes. An **entity type usually has one or more attributes whose values are distinct for each individual entity in the entity set. Such an attribute is called a key attribute** and its values can be used to identify each entity uniquely. For example, the Name attribute is a key of the COMPANY entity type in Figure 3.6 because no two companies are allowed to have the same name. For the PERSON entity type, a typical key attribute is SSN (Social Security number).
- Sometimes several attributes together form a key, meaning that the combination of the attribute values must be distinct for each entity. If a set of attributes possesses this property, the proper way to represent this in the ER model that we describe here is to define a composite attribute and designate it as a key attribute of the entity type. Notice that such a composite key must be minimal; that is, all component attributes must be included in the composite attribute to have the uniqueness property.
- Superfluous attributes must not be included in a key. In ER diagrammatic notation, each key attribute has its name underlined inside the oval, as illustrated in Figure 3.7(a).
- Specifying that an attribute is a key of an entity type means that the preceding uniqueness property must hold for every entity set of the entity type. Hence, it is a constraint that prohibits any two entities from having the same value for the key attribute at the same time. It is not the property of a particular entity set; rather, it is a constraint on any entity set of the entity type at any point in time. This key constraint (and other constraints we discuss later) is derived from the constraints of the miniworld that the database represents.
- Some entity types have more than one key attribute. For example, each of the Vehicle_id and Registration attributes of the entity type CAR (Figure 3.7) is a key in its own right. The Registration attribute is an example of a composite key formed from two simple component attributes, State and Number, neither of which is a key on its own.
- An entity type may also have no key, in which case it is called a weak entity type (see Section 3.5).
- In our diagrammatic notation, if two attributes are underlined separately, then each is a key on its own. Unlike the relational model, there is no concept of primary key in the ER model that we present here; the primary key will be chosen during mapping to a relational schema.

Value Sets (Domains) of Attributes:

- Each simple attribute of an entity type is associated with a value set (or domain of values), which specifies the **set of values that may be assigned to that attribute for each individual entity**. In Figure 3.6, if the range of ages allowed for employees is between 16 and 70, we can specify the value set of the Age attribute of EMPLOYEE to be the set of integer numbers between 16 and 70. Similarly, we can specify the value set for the Name attribute to be the set of strings of alphabetic characters separated by blank characters, and so on.
- **Value sets are not displayed in ER diagrams**, and are typically specified using the basic data types available in most programming languages, such as integer, string, Boolean, float, enumerated type, subrange, and so on.
- Mathematically, an attribute A of entity set E whose value set is V can be defined as a function from E to the power set P(V) of V: (Note: The power set P (V) of a set V is the set of all subsets of V.)

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

- We refer to the **value of attribute A for entity e as A(e)**. The previous definition covers both single-valued and multivalued attributes, as well as NULLs.

- A NULL value is represented by the empty set. For single-valued attributes, A(e) is restricted to being a singleton set for each entity e in E, whereas there is no restriction on multivalued attributes. (Note: A singleton set is a set with only one element (value)).
- For a composite attribute A, the value set V is the power set of the Cartesian product of P(V1), P(V2), ..., P(Vn), where V1, V2, ..., Vn are the value sets of the simple component attributes that form A:

$$V = P(P(V1) \times P(V2) \times ... \times P(Vn))$$

- The value set provides all possible values. Usually only a small number of these values exist in the database at a particular time. Those values represent the data from the current state of the miniworld. They correspond to the data as it actually exists in the miniworld.

3.3.3 Initial Conceptual Design of the COMPANY Database

- According to the requirements listed in Section 3.2, we can identify four entity types—one corresponding to each of the four items in the specification (see Figure 3.8):

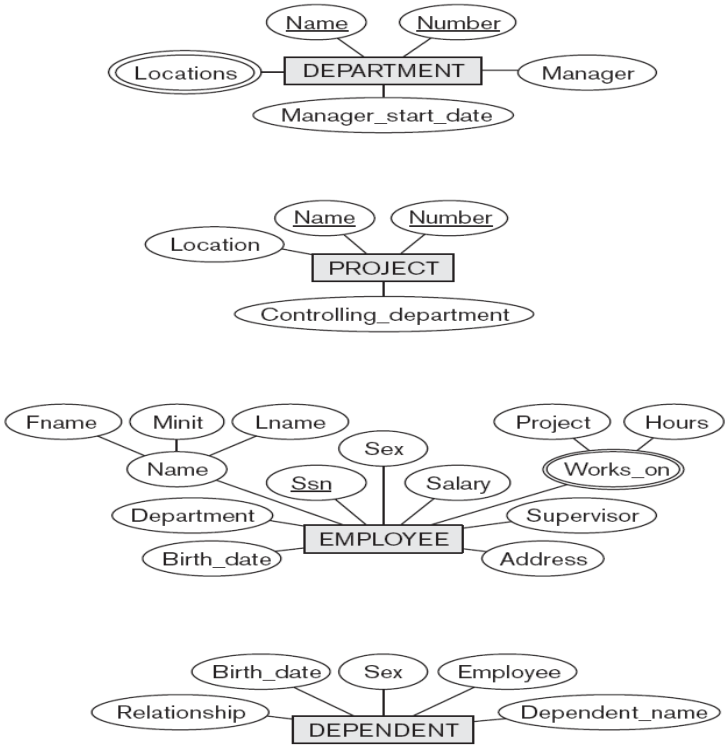


Figure 3.8 Preliminary design of entity types for the COMPANY database. Some of the shown attributes will be refined into relationships.

- 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.
- An entity type PROJECT with attributes Name, Number, Location, and Controlling_department. Both Name and Number are (separate) key attributes.
- 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.
- An entity type DEPENDENT with attributes Employee, Dependent_name, Sex, Birth_date, and Relationship (to the employee).

- The number of hours per week an employee works on each project is a characteristic listed as part of the third requirement in Section 3.2, and it can be represented by a multivalued composite attribute of EMPLOYEE called Works_on with the simple components (Project, Hours).
- Alternatively, it can be represented as a multivalued composite attribute of PROJECT called Workers with the simple components (Employee, Hours). We choose the first alternative in Figure 3.8, which shows each of the entity types just described. The Name attribute of EMPLOYEE is shown as a composite attribute, presumably after consultation with the users.

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

- In Figure 3.8 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;
  - * the attribute Supervisor of EMPLOYEE refers to another employee (the one who supervises this employee);
  - * the attribute Department of EMPLOYEE refers to the department for which the employees works; and so on.
- In the ER model, these references should not be represented as attributes but as relationships, which are discussed in this section.
- The COMPANY database schema will be refined in Section 3.6 to represent relationships explicitly. In the initial design of entity types, relationships are typically captured in the form of attributes. As the design is refined, these attributes get converted into relationships between entity types.

3.4.1 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.
- As for the case of entity types and entity sets, a relationship type and its corresponding relationship set are customarily referred to by the same name, R.
- Mathematically, the relationship set R is a set of relationship instances r_i, where each r_i associates *n* individual entities (e_1, e_2, ..., e_n), and each entity e_j in r_i is a member of entity type E_j, $1 \leq j \leq n$. Hence, a **relationship set is a mathematical relation on E_1, E_2, ..., E_n**;
- **Alternatively, it can be defined as a subset of the Cartesian product of the entity sets $E_1 \times E_2 \times ... \times E_n$**. Each of the entity types E_1, E_2, ..., E_n is said to participate in the relationship type R; similarly, each of the individual entities e_1, e_2, ..., e_n is said to participate in the relationship instance $r_i = (e_1, e_2, ..., e_n)$.
- 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. Each such relationship instance r_i represents the fact that the entities participating in r_i are related in some way in the corresponding miniworld situation.
- 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 in the corresponding entity set. Each relationship instance in the relationship set WORKS_FOR associates one EMPLOYEE entity and one DEPARTMENT entity. Figure 3.9 illustrates this example, where each relationship instance r_i is shown connected to the EMPLOYEE and DEPARTMENT entities that participate in r_i. In the miniworld represented by Figure 3.9, employee's e_1, e_3, and e_6 work for department d1; employee's e2 and e4 work for department d2; and employee's e_5 and e_7 work for department d_3.

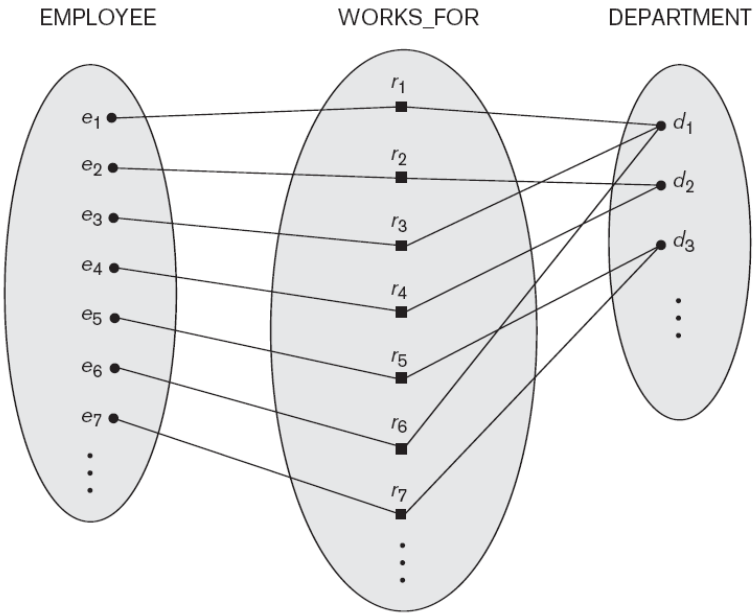


Figure 3.9 Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

- 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 (see Figure 3.2).

3.4.2 Relationship Degree, Role Names, and Recursive Relationships

Degree of a Relationship Type:

- **The degree of a relationship type is the number of participating entity types.** Hence, the WORKS_FOR relationship is of degree two.
- A relationship type of degree two is called binary, and one of degree three is called ternary. An example of a ternary relationship is SUPPLY, shown in Figure 3.10, where each relationship instance r_i associates three entities—a supplier s, a part p, and a project j—whenever s supplies part p to project j.
- Relationships can generally be of any degree, but the ones most common are binary relationships.
- Higher degree relationships are generally more complex than binary relationships.

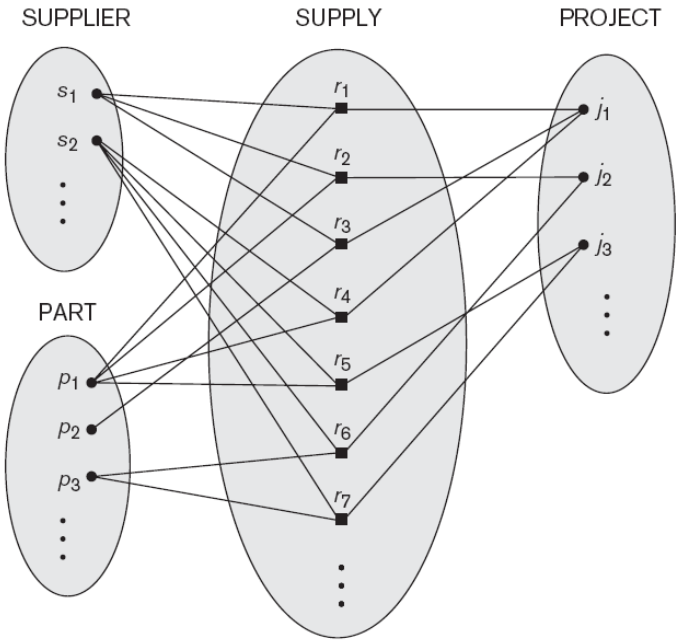


Figure 3.10 Some relationship instances in the SUPPLY ternary relationship set.

Relationships as Attributes:

- Consider the WORKS_FOR relationship type in Figure 3.9. 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. This is what we did in Figure 3.8 when we specified the initial design of the entity type EMPLOYEE for the COMPANY database.
- When we think of a binary relationship as an attribute, we always have two options. In this example, the **alternative** is to think of a **multivalued attribute Employee of the entity type DEPARTMENT** whose values for each DEPARTMENT entity is the set of EMPLOYEE entities who work for that department.
- The value set of this Employee attribute is the power set of the EMPLOYEE entity set. Either of these two attributes—Department of EMPLOYEE or Employee of DEPARTMENT—can represent the WORKS_FOR relationship type. If both are represented, they are constrained to be inverses of each other.

Role Names and Recursive Relationships:

- Each entity type that participates in a relationship type plays a particular role in the relationship. The role name signifies the role that a participating entity from the entity type plays in each relationship instance, and helps to explain what the relationship means.
- For example, in the WORKS_FOR relationship type, EMPLOYEE plays the role of employee or worker and DEPARTMENT plays the role of department or employer.
- Role names are not technically necessary in relationship types where all the participating entity types are distinct, since each participating entity type name can be used as the role name. However, 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**.

- Figure 3.11 shows an 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). Each relationship instance r_i in SUPERVISION associates two employee entities e_j and e_k, one of which plays the role of supervisor and the other the role of supervisee. In Figure 3.11, the lines marked '1' represent the supervisor role, and those marked '2' represent the supervisee role; hence, e_1 supervises e_2 and e_3, e_4 supervises e_6 and e_7, and e_5 supervises e_1 and e_4. In this example, each relationship instance must be connected with two lines, one marked with '1' (supervisor) and the other with '2' (supervisee).

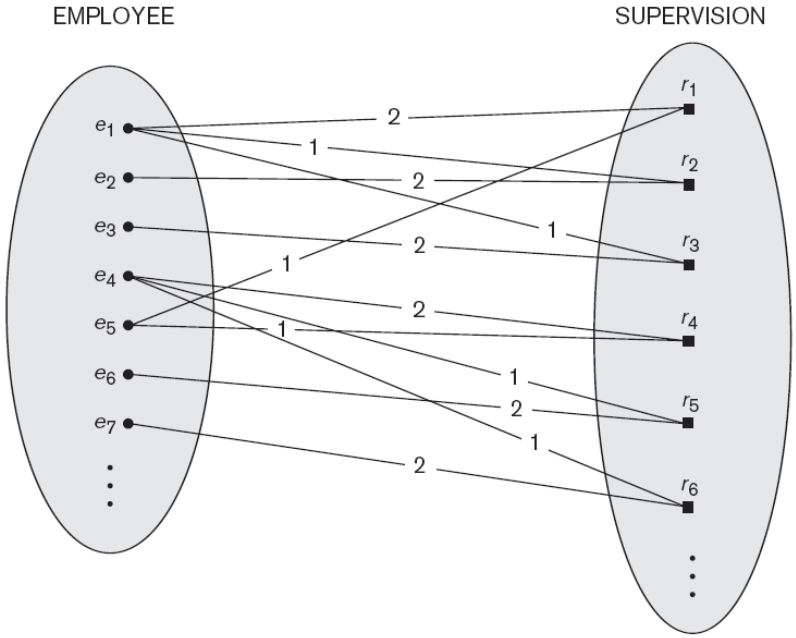


Figure 3.11 A recursive relationship SUPERVISION between EMPLOYEE in the supervisor role (1) and EMPLOYEE in the subordinate role (2).

3.4.3 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 mini-world situation that the relationships represent.
- For example, in Figure 3.9, 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. We can distinguish **two main types of binary relationship constraints: cardinality ratio and participation**.

Cardinality Ratios for Binary Relationships:

- The cardinality ratio for a binary relationship specifies the maximum number of relationship instances that an entity can participate in. For example, in the WORKS_FOR binary relationship type, DEPARTMENT: EMPLOYEE is of cardinality ratio 1: N, meaning that each department can be related to (that is, employs) any number of employees, but an employee can be related to (work for) only one department. This means that for this particular relationship WORKS_FOR, a particular department entity can be related to any number of employees (N indicates there is no maximum number). On the other hand, an employee can be related to a maximum of one department.
- The possible cardinality ratios for binary relationship types are 1:1, 1:N, N:1, and M:N.
- An example of a 1:1 binary relationship is MANAGES (Figure 3.12), which relates a department entity to the employee who manages that department. This represents the miniworld constraints

that—at any point in time—an employee can manage one department only and a department can have one manager only.

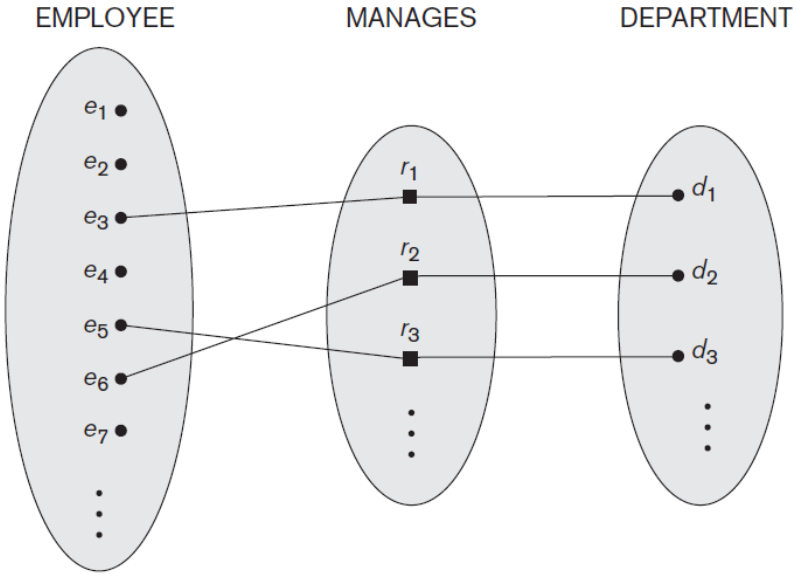


Figure 3.12 A 1:1 relationship MANAGES

- The relationship type WORKS_ON (Figure 3.13) is of cardinality ratio M:N, because the mini-world rule is that an employee can work on several projects and a project can have several employees.
- Cardinality ratios for binary relationships are represented on ER diagrams by displaying 1, M, and N on the diamonds as shown in Figure 3.2.
- An alternative notation (see Section 3.7.4) allows the designer to specify a specific maximum number on participation, such as 4 or 5.

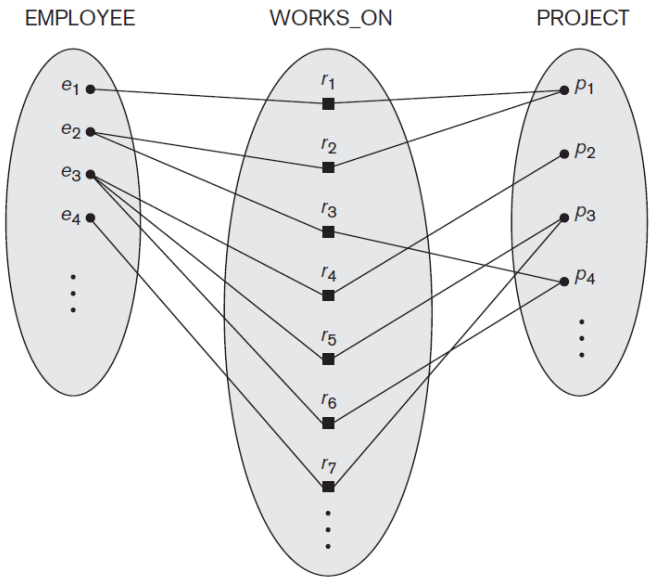


Figure 3.13 An M:N relationship, WORKS_ON

Participation Constraints and Existence Dependencies:

- The participation constraint specifies whether the existence of an entity depends on its being related to another entity via the relationship type. This constraint specifies the minimum number of relationship instances that each entity can participate in, and is sometimes called the minimum cardinality constraint.
- There are **two types of participation constraints—total and partial**—that we illustrate by example. If a company policy states that every employee must work for a department, then an employee entity can exist only if it participates in at least one WORKS_FOR relationship instance (Figure 3.9). Thus, the participation of EMPLOYEE in WORKS_FOR is called total participation, meaning that every entity in the total set of employee entities must be related to a department entity via WORKS_FOR. **Total participation is also called existence dependency**.
- In Figure 3.12 we do not expect every employee to manage a department, so the participation of EMPLOYEE in the MANAGES relationship type is partial, meaning that some or part of the set of employee entities are related to some department entity via MANAGES, but not necessarily all.
- The **cardinality ratio and participation constraint, taken together, is the structural constraints of a relationship type**.
- **In ER diagrams, total participation (or existence dependency) is displayed as a double line connecting the participating entity type to the relationship, whereas partial participation is represented by a single line** (see Figure 3.2).
- In this notation, we can either specify **no minimum (partial participation) or a minimum of one (total participation)**.
- The alternative notation (see Section 3.7.4) allows the designer to specify a specific minimum number on participation in the relationship, such as 4 or 5.

Attributes of Relationship Types

- Relationship types can also have attributes, similar to those of entity types. For example, to record the number of hours per week that an employee works on a particular project, we can include an attribute Hours for the WORKS_ON relationship type in Figure 3.13.
- Another example is to include the date on which a manager started managing a department via an attribute Start_date for the MANAGES relationship type in Figure 3.12.
- Notice that attributes of 1:1 or 1: N relationship types can be migrated to one of the participating entity types. For example, the Start_date attribute for the MANAGES relationship can be an attribute of either EMPLOYEE or DEPARTMENT, although conceptually it belongs to MANAGES. This is because MANAGES is a 1:1 relationship, so every department or employee entity participates in at most one relationship instance. Hence, the value of the Start_date attribute can be determined separately, either by the participating department entity or by the participating employee (manager) entity.
- For a 1: N relationship type, a relationship attribute can be migrated only to the entity type on the N-side of the relationship. For example, in Figure 3.9, if the WORKS_FOR relationship also has an attribute Start_date that indicates when an employee started working for a department, this attribute can be included as an attribute of EMPLOYEE. This is because each employee works for only one department, and hence participates in at most one relationship instance in WORKS_FOR.
- In both 1:1 and 1:N relationship types, the decision where to place a relationship attribute—as a relationship type attribute or as an attribute of a participating entity type—is determined subjectively by the schema designer.

- For M: N relationship types, some attributes may be determined by the combination of participating entities in a relationship instance, not by any single entity. Such attributes must be specified as relationship attributes. An example is the Hours attribute of the M: N relationship WORKS_ON (Figure 3.13); the number of hours per week an employee currently works on a project is determined by an employee-project combination and not separately by either entity.

3.5 Weak Entity Types

- **Entity types that do not have key attributes of their own are called weak entity types** (child entity type or the subordinate entity type).
- In contrast, regular entity types that do **have a key attribute**—which include all the examples discussed so far—are called **strong entity types**.
- Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with one of their attribute values. We call this **other entity type the identifying or owner entity type**, (parent entity type or the dominant entity) and we call the relationship type that relates a weak entity type to its owner the identifying relationship of the weak entity type.
- A weak entity type **always has a total participation constraint** (existence dependency) with respect to its identifying relationship because a weak entity cannot be identified without an owner entity. However, **not every existence dependency results in a weak entity type.** For example, a **DRIVER_LICENSE entity** cannot exist unless it is related to a PERSON entity, even though it has its own key (License_number) and hence is not a weak entity.
- Consider the entity type DEPENDENT, related to EMPLOYEE, which is used to keep track of the dependents of each employee via a 1: N relationship (Figure 3.2). In our example, the attributes of DEPENDENT are Name (the first name of the dependent), Birth_date, Sex, and Relationship (to the employee). Two dependents of two distinct employees may, by chance, have the same values for Name, Birth_date, Sex, and Relationship, but they are still distinct entities. They are identified as distinct entities only after determining the particular employee entity to which each dependent is related. Each employee entity is said to own the dependent entities that are related to it.
- A **weak entity type normally has a partial key (**discriminator), which is the attribute that can uniquely identify weak entities that are related to the same owner entity. In our example, if we assume that no two dependents of the same employee ever have the same first name, the attribute Name of DEPENDENT is the partial key. In the worst case, **a composite attribute of all the weak entity's attributes will be the partial key**.
- In ER diagrams, both a **weak entity type and its identifying relationship are distinguished by surrounding their boxes and diamonds with double lines** (see Figure 3.2). The **partial key** attribute is underlined with a **dashed or dotted line**.
- Weak entity types can sometimes be represented as complex (composite, multivalued) attributes. In the preceding example, we could specify a multivalued attribute Dependents for EMPLOYEE, which is a composite attribute with component attributes Name, Birth_date, Sex, and Relationship. The choice of which representation to use is made by the database designer.
- One criterion that may be used is to choose the weak entity type representation if there are many attributes. If the weak entity participates independently in relationship types other than its identifying relationship type, then it should not be modeled as a complex attribute.

- In general, any number of levels of weak entity types can be defined; an owner entity type may itself be a weak entity type. In addition, a weak entity type may have more than one identifying entity type and an identifying relationship type of degree higher than two, as we illustrate in Section 3.9.

3.6 Refining the ER Design for the COMPANY Database

- We can now refine the database design in Figure 3.8 by changing the attributes that represent relationships into relationship types. The cardinality ratio and participation constraint of each relationship type are determined from the requirements listed in Section 3.2.
- If some cardinality ratio or dependency cannot be determined from the requirements, the users must be questioned further to determine these structural constraints.
- In our example, we specify the following relationship types:
    * MANAGES, a 1:1 relationship type between EMPLOYEE and DEPARTMENT. EMPLOYEE participation is partial. DEPARTMENT participation is not clear from the requirements. We question the users, who say that a department must have a manager at all times, which implies total participation. The attribute Start_date is assigned to this relationship type.
    * WORKS_FOR, a 1: N relationship type between DEPARTMENT and EMPLOYEE. Both participations are total.
    * CONTROLS, a 1: N relationship type between DEPARTMENT and PROJECT. The participation of PROJECT is total, whereas that of DEPARTMENT is determined to be partial, after consultation with the users indicates that some departments may control no projects.
    * SUPERVISION, a 1: N relationship type between EMPLOYEE (in the supervisor role) and EMPLOYEE (in the supervisee role). Both participations are determined to be partial, after the users indicate that not every employee is a supervisor and not every employee has a supervisor.
    * WORKS_ON, determined to be an M: N relationship type with attribute Hours, after the users indicate that a project can have several employees working on it. Both participations are determined to be total.
    * DEPENDENTS_OF, a 1: N relationship type between EMPLOYEE and DEPENDENT, which is also the identifying relationship for the weak entity type DEPENDENT. The participation of EMPLOYEE is partial, whereas that of DEPENDENT is total.
- After specifying the above six relationship types, we remove from the entity types in Figure 3.8 all attributes that have been refined into relationships. These include
    * Manager and Manager_start_date from DEPARTMENT;
    * Controlling_department from PROJECT;
    * Department, Supervisor, and Works_on from EMPLOYEE; and
    * Employee from DEPENDENT.
- It is important to have the least possible redundancy when we design the conceptual schema of a database.

3.7 ER Diagrams, Naming Conventions, and Design Issues

3.7.1 Summary of Notation for ER Diagrams

- Figures 3.9 through 3.13 illustrate examples of the participation of entity types in relationship types by displaying their sets or extensions—the individual entity instances in an entity set and the individual relationship instances in a relationship set.
- Figure 3.2 displays the COMPANY ER database schema as an ER diagram. We now review the full ER diagram notation.
    * **Entity types** such as EMPLOYEE, DEPARTMENT, and PROJECT are shown in **rectangular boxes**.
    * **Relationship types** such as WORKS_FOR, MANAGES, CONTROLS, and WORKS_ON are shown in **diamond-shaped boxes** attached to the participating entity types with straight lines.
    * **Attributes** are shown in **ovals**, and each attribute is attached by a straight line to its entity type or relationship type.
    * Component attributes of a composite attribute are attached to the oval representing the composite attribute, as illustrated by the Name attribute of EMPLOYEE.
    * **Multivalued attributes** are shown in **double ovals**, as illustrated by the Locations attribute of DEPARTMENT.
    * **Key attributes** have their **names underlined**.
    * **Derived attributes** are shown in **dotted ovals**, as illustrated by the Number_of_employees attribute of DEPARTMENT.
    * **Weak entity types** are distinguished by being placed in **double rectangles** and by having their **identifying relationship** placed in **double diamonds**, as illustrated by the DEPENDENT entity type and the DEPENDENTS_OF identifying relationship type. The **partial key of the weak entity type** is underlined with a **dotted line**.
- In Figure 3.2 the cardinality ratio of each binary relationship type is specified by attaching a 1, M, or N on each participating edge. The cardinality ratio of
    * DEPARTMENT : EMPLOYEE in MANAGES is 1:1, whereas it is
    * 1:N for DEPARTMENT: EMPLOYEE in WORKS_FOR, and
    * M: N for WORKS_ON.
- The participation constraint is specified by a single **line for partial participation and by double lines for total participation (existence dependency).**
- In Figure 3.2 we show the role names for the SUPERVISION relationship type because the same EMPLOYEE entity type plays two distinct roles in that relationship.
- Notice that the cardinality ratio is 1: N from supervisor to supervisee because each employee in the role of supervisee has at most one direct supervisor, whereas an employee in the role of supervisor can supervise zero or more employees.
- Figure 3.14 summarizes the conventions for ER diagrams.

3.7.2 Proper Naming of Schema Constructs

- One should choose names for entity types, attributes, relationship types, and (particularly) roles based on the meanings attached to the different constructs in the schema.
- We choose to use **singular names for entity types**, rather than plural ones, because the entity type name applies to each individual entity belonging to that entity type.
- In the ER diagrams, we will use the convention that

  - * **entity type and relationship type names are uppercase letters**,
  - * **attribute names have their initial letter capitalized**, and
  - * **role names are lowercase letters**.
- We have used this convention in Figure 3.2.
- As a general practice, given a narrative description of the database requirements, the **nouns** appearing in the narrative tend to **give rise to entity type names,** and the **verbs tend to indicate names of relationship types**. **Attribute names generally arise from additional nouns that describe the nouns** corresponding to entity types.
- Another naming consideration involves choosing binary relationship names to make the ER diagram of the **schema readable from left to right and from top to bottom**.
- We have generally followed this guideline in Figure 3.2. To explain this naming convention further, we have one exception to the convention in Figure 3.2—the **DEPENDENTS_OF relationship type, which reads from bottom to top**. When we describe this relationship, we can say that the DEPENDENT entities (bottom entity type) are DEPENDENTS_OF (relationship name) an EMPLOYEE (top entity type).
- **To change this to read from top to bottom, we could rename the relationship type to HAS_DEPENDENTS**, which would then read as follows: An EMPLOYEE entity (top entity type) HAS_DEPENDENTS (relationship name) of type DEPENDENT (bottom entity type).

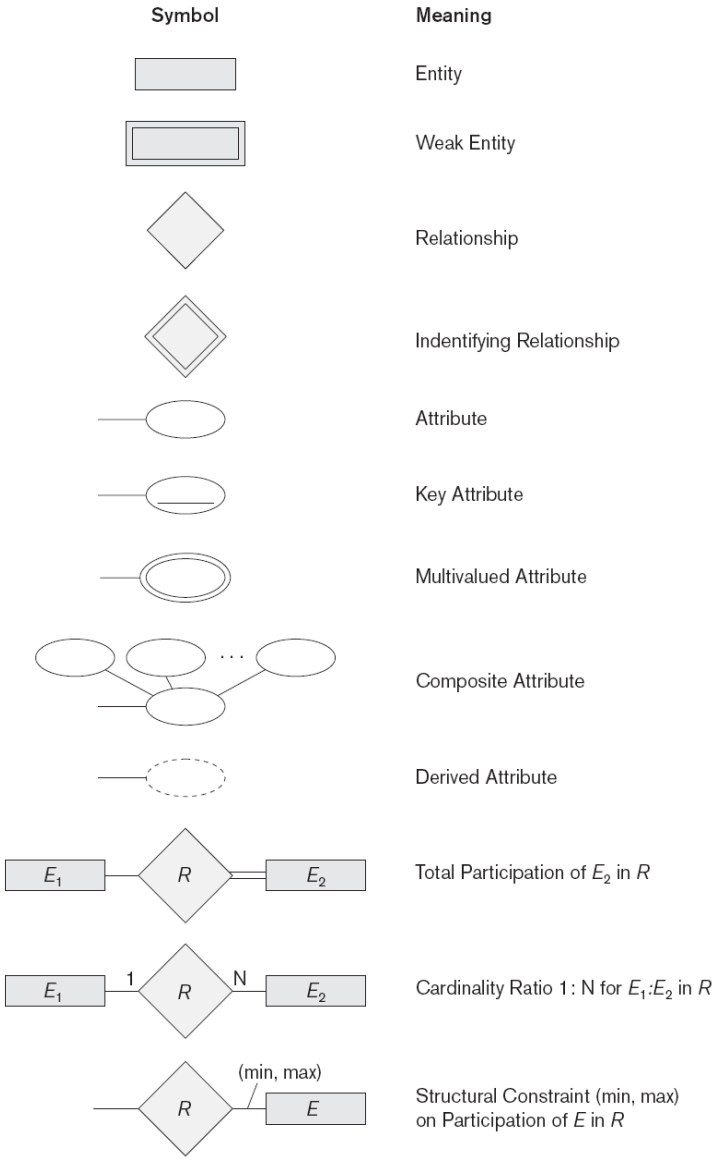


Figure 3.14 Summary of the notation for ER diagrams

3.7.3 Design Choices for ER Conceptual Design

- It is occasionally difficult to decide whether a particular concept in the miniworld should be modeled as an entity type, an attribute, or a relationship type. In this section, we give some brief guidelines as to which construct should be chosen in particular situations.
- In general, the schema design process should be considered an iterative refinement process, where an initial design is created and then iteratively refined until the most suitable design is reached. Some of the refinements that are often used include the following:
    * A concept may be first modeled as an attribute and then refined into a relationship because it is determined that the attribute is a reference to another entity type. It is often the case that pair of such attributes that are inverses of one another are refined into a binary relationship. It is important to note that in our notation, **once an attribute is replaced by a relationship, the attribute itself should be removed from the entity type to avoid duplication** and redundancy.
    * Similarly, **an attribute that exists in several entity types may be elevated or promoted to an independent entity type**. For example, suppose that several entity types in a UNIVERSITY database, such as STUDENT, INSTRUCTOR, and COURSE, each has an attribute Department in the initial design; the designer may then choose to create an entity type DEPARTMENT with a single attribute Dept_name and relate it to the three entity types (STUDENT, INSTRUCTOR, and COURSE) via appropriate relationships.
    * An inverse refinement to the previous case may be applied—for example, if an entity type DEPARTMENT exists in the initial design with a single attribute Dept_name and is related to only one other entity type, STUDENT. In this case, DEPARTMENT may be reduced or demoted to an attribute of STUDENT.

3.7.4 Alternative Notations for ER Diagrams

- There are many alternative diagrammatic notations for displaying ER diagrams.
- In this section, we describe one alternative ER notation for specifying structural constraints on relationships, which replaces the cardinality ratio (1:1, 1: N, M: N) and single/double line notation for participation constraints.
- This notation involves associating a pair of integer numbers (min, max) with each participation of an entity type E in a relationship type R, where $0 \leq \min \leq \max$ and $\max \geq 1$.
- The numbers mean that for each entity e in E, e must participate in at least min and at most max relationship instances in R at any point in time. In this method, **min = 0 implies partial participation, whereas min > 0 implies total participation**.
- Figure 3.15 displays the COMPANY database schema using the (min, max) notation.
- It also displays all the role names for the COMPANY database schema.

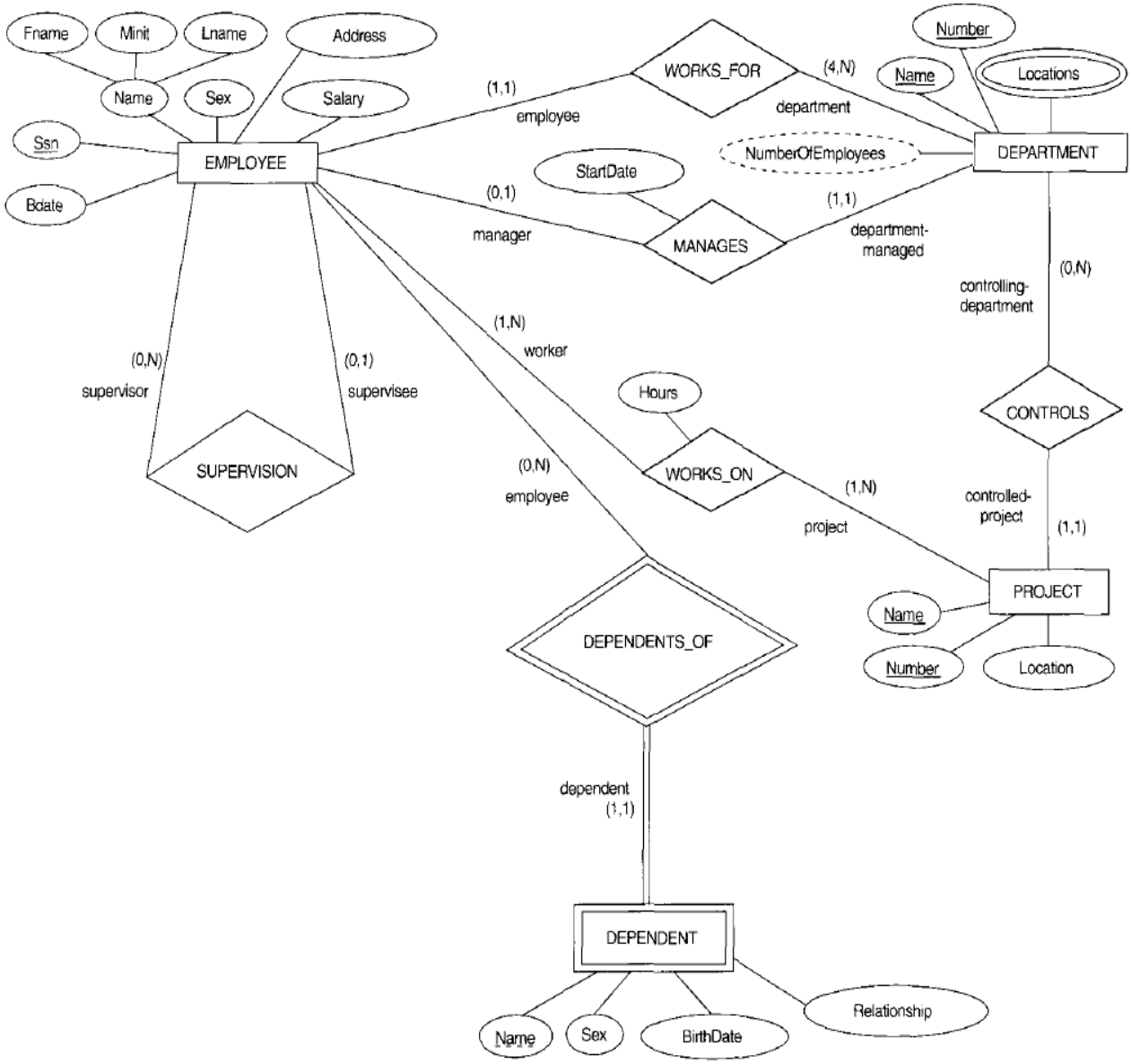


FIGURE 3.15 ER diagrams for the COMPANY schema, with structural constraints specified using (min, max) notation.

3.8 Relationship Types of Degree Higher than Two

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

- The ER diagram notation for a ternary relationship type is shown in Figure 3.17(a), which displays the schema for the SUPPLY relationship type that was displayed at the entity set/relationship set or instance level in Figure 3.10. Recall that the relationship set of SUPPLY is a set of relationship instances (s, j, p), where s is a SUPPLIER who is currently supplying a PART p to a PROJECT j.
- In general, a relationship type R of degree n will have n edges in an ER diagram, one connecting R to each participating entity type.

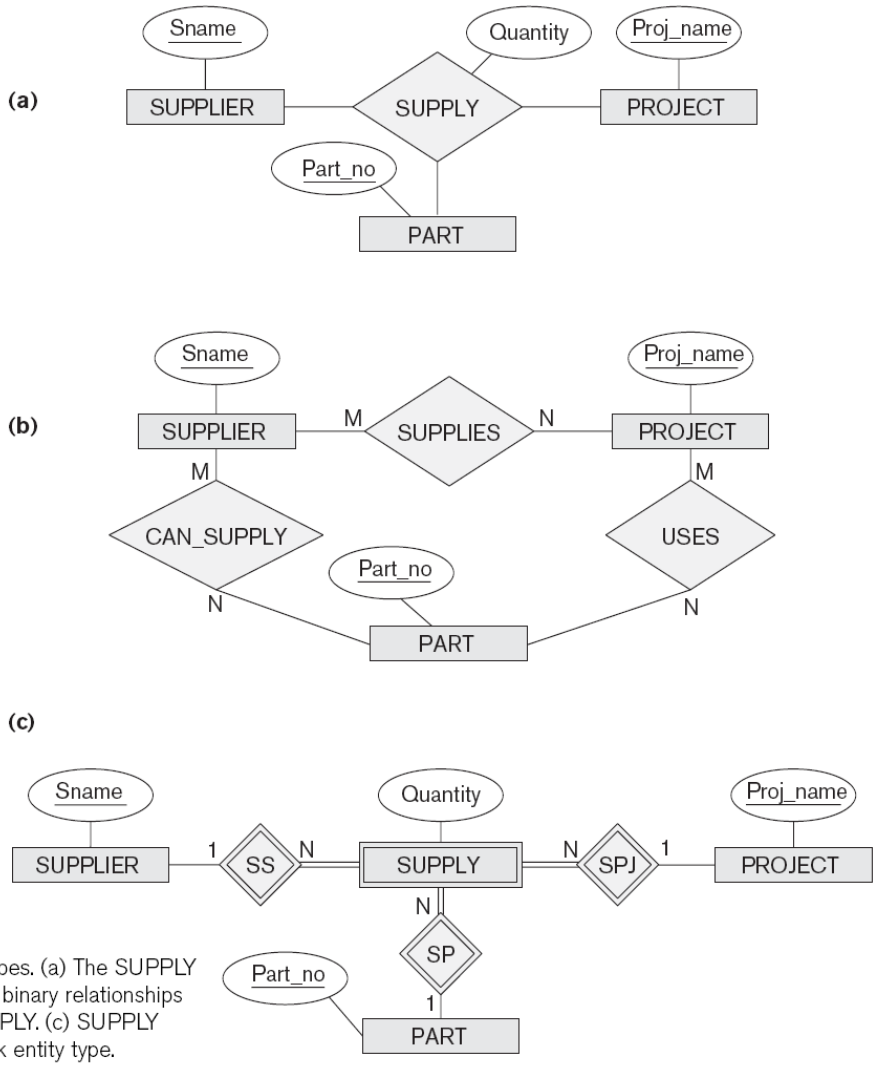
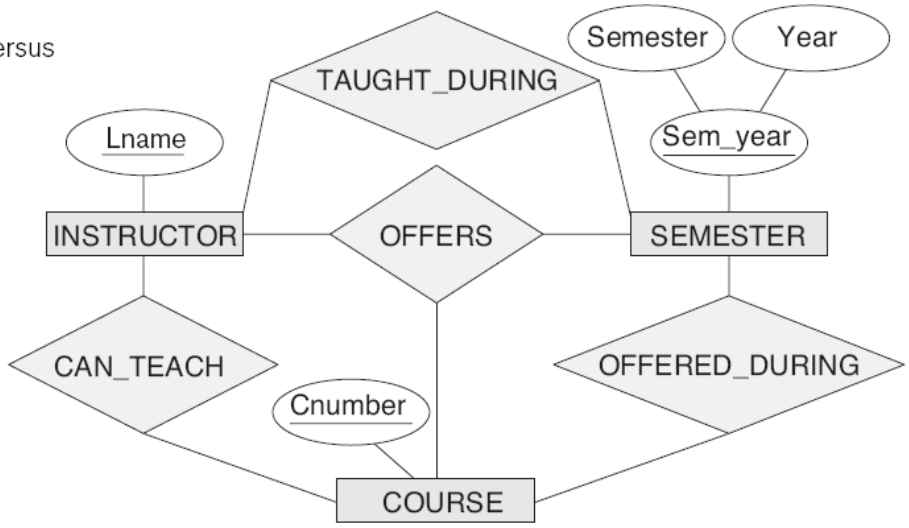


Figure 3.17
Ternary relationship types. (a) The SUPPLY relationship. (b) Three binary relationships not equivalent to SUPPLY. (c) SUPPLY represented as a weak entity type.

- Figure 3.17(b) shows an ER diagram for three binary relationship types CAN_SUPPLY, USES, and SUPPLIES. In general, a ternary relationship type represents different information than do three binary relationship types. Consider the three binary relationship types CAN_SUPPLY, USES, and SUPPLIES. Suppose that CAN_SUPPLY, between SUPPLIER and PART, includes an instance (s, p) whenever supplier s can supply part p (to any project); USES, between PROJECT and PART, includes an instance (j, p) whenever project j uses part p; and SUPPLIES, between SUPPLIER and PROJECT, includes an instance (s, j) whenever supplier s supplies some part to project j. The existence of three relationship instances (s, p), (j, p), and (s, j) in CAN_SUPPLY, USES, and SUPPLIES, respectively, does not necessarily imply that an instance (s, j, p) exists in the ternary relationship SUPPLY, because the meaning is different.
- It is often tricky to decide whether a particular relationship should be represented as a relationship type of degree n or should be broken down into several relationship types of smaller degrees. The designer must base this decision on the semantics or meaning of the particular situation being represented.
- The typical solution is to include the ternary relationship plus one or more of the binary relationships, if they represent different meanings and if all are needed by the application.

- Some database design tools are based on variations of the ER model that permit only binary relationships. In this case, a ternary relationship such as SUPPLY must be represented as a weak entity type, with no partial key and with three identifying relationships. The three participating entity types SUPPLIER, PART, and PROJECT are together the owner entity types (see Figure 3.17(c)). Hence, an entity in the weak entity type SUPPLY in Figure 3.17(c) is identified by the combination of its three owner entities from SUPPLIER, PART, and PROJECT.
- Another example is shown in Figure 3.18. The ternary relationship type OFFERS represents information on instructors offering courses during particular semesters; hence it includes a relationship instance (i, s, c) whenever INSTRUCTOR i offers COURSE c during SEMESTER s. The three binary relationship types shown in Figure 3.18 have the following meanings: CAN_TEACH relates a course to the instructors who can teach that course, TAUGHT_DURING relates a semester to the instructors who taught some course during that semester, and OFFERED_DURING relates a semester to the courses offered during that semester by any instructor.
- These ternary and binary relationships represent different information, but certain constraints should hold among the relationships. For example, a relationship instance (i, s, c) should not exist in OFFERS unless an instance (i, s) exists in TAUGHT_DURING, an instance (s, c) exists in OFFERED_DURING, and an instance (i, c) exists in CAN_TEACH. However, the reverse is not always true; we may have instances (i, s), (s, c), and (i, c) in the three binary relationship types with no corresponding instance (i, s, c) in OFFERS.
- The schema designer must analyze the meaning of each specific situation to decide which of the binary and ternary relationship types are needed.

Figure 3.18
Another example of ternary versus binary relationship types.



- Notice that it is possible to have a weak entity type with a ternary (or n-ary) identifying relationship type. In this case, the weak entity type can have several owner entity types. An example is shown in Figure 3.19. This example shows part of a database that keeps track of candidates interviewing for jobs at various companies, and may be part of an employment agency database, for example. In the requirements, a candidate can have multiple interviews with the same company (for example, with different company departments or on separate dates), but a job offer is made based on one of the interviews. Here, INTERVIEW is represented as a weak entity with two owners CANDIDATE and

COMPANY, and with the partial key Dept_date. An INTERVIEW entity is uniquely identified by a candidate, a company, and the combination of the date and department of the interview.

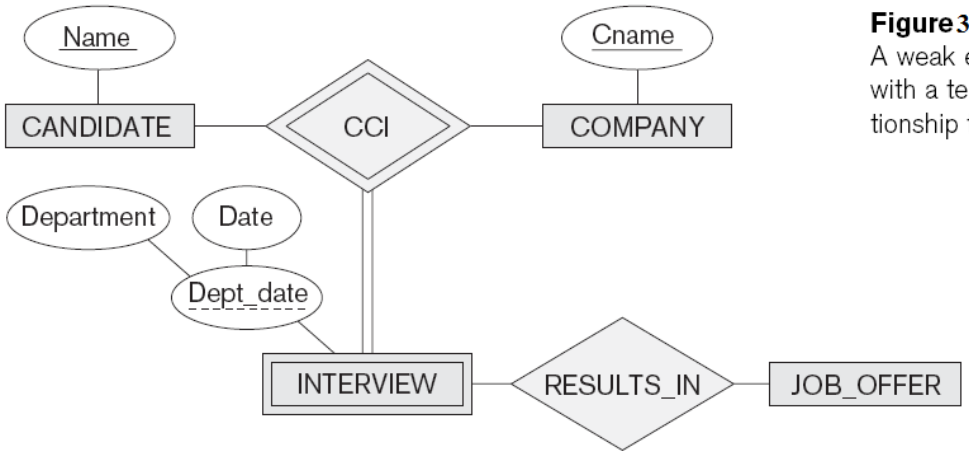


Figure 3.19
A weak entity type INTERVIEW with a ternary identifying relationship type.

3.9.2 Constraints on Ternary (or Higher-Degree) Relationships

- There are two notations for specifying structural constraints on n-ary relationships, and they specify different constraints. They should thus both be used if it is important to fully specify the structural constraints on a ternary or higher-degree relationship.
- The first notation is based on the cardinality ratio notation of binary relationships displayed in Figure 3.2. Here, a 1, M, or N is specified on each participation arc (both M and N symbols stand for many or any number). Let us illustrate this constraint using the SUPPLY relationship in Figure 3.17.
- Recall that the relationship set of SUPPLY is a set of relationship instances (s, j, p), where s is a SUPPLIER, j is a PROJECT, and p is a PART. Suppose that the constraint exists that for a particular project-part combination, only one supplier will be used (only one supplier supplies a particular part to a particular project). In this case, we place 1 on the SUPPLIER participation, and M, N on the PROJECT, PART participations in Figure 3.17. This specifies the constraint that a particular (j, p) combination can appear at most once in the relationship set because each such (PROJECT, PART) combination uniquely determines a single supplier. Hence, any relationship instance (s, j, p) is uniquely identified in the relationship set by its (j, p) combination, which makes (j, p) a key for the relationship set. In this notation, the participations that have a 1 specified on them are not required to be part of the identifying key for the relationship set. If all three cardinalities are M or N, then the key will be the combination of all three participants.
- The second notation is based on the (min, max) notation displayed in Figure 3.15 for binary relationships. A (min, max) on a participation here specifies that each entity is related to at least min and at most max relationship instances in the relationship set. These constraints have no bearing on determining the key of an n-ary relationship, where n > 2, but specify a different type of constraint that places restrictions on how many relationship instances each entity can participate in.