

DATABASE MANAGEMENT SYSTEM [As per Choice Based Credit System (CBCS) scheme] (Effective from the academic year 2016 -2017) SEMESTER – V			
Subject Code	15CS53	IA Marks	20
Number of Lecture Hours/Week	4	Exam Marks	80
Total Number of Lecture Hours	50	Exam Hours	03
CREDITS – 04			
Course objectives: This course will enable students to			
<ul style="list-style-type: none"> • Provide a strong foundation in database concepts, technology, and practice. • Practice SQL programming through a variety of database problems. • Demonstrate the use of concurrency and transactions in database • Design and build database applications for real world problems. 			
Module – 1			Teaching Hours
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, Specialization and Generalization. Textbook 1: Ch 1.1 to 1.8, 2.1 to 2.6, 3.1 to 3.10			10 Hours
Module – 2			
Relational Model: Relational Model Concepts, Relational Model Constraints and relational database schemas, Update operations, transactions, and dealing with constraint violations. Relational Algebra: Unary and Binary relational operations, additional relational operations (aggregate, grouping, etc.) Examples of Queries in relational algebra. Mapping Conceptual Design into a Logical Design: Relational Database Design using ER-to-Relational mapping. SQL: SQL data definition and data types, specifying constraints in SQL, retrieval queries in SQL, INSERT, DELETE, and UPDATE statements in SQL, Additional features of SQL. Textbook 1: Ch4.1 to 4.5, 5.1 to 5.3, 6.1 to 6.5, 8.1; Textbook 2: 3.5			10 Hours
Module – 3			
SQL : Advances Queries: More complex SQL retrieval queries, Specifying constraints as assertions and action triggers, Views in SQL, Schema change statements in SQL. Database Application Development: Accessing databases from applications, An introduction to JDBC, JDBC classes and interfaces, SQLJ, Stored procedures, Case study: The internet Bookshop. Internet Applications: The three-Tier application architecture, The presentation layer, The Middle Tier Textbook 1: Ch7.1 to 7.4; Textbook 2: 6.1 to 6.6, 7.5 to 7.7.			10 Hours
Module – 4			
Normalization: Database Design Theory – Introduction to Normalization using Functional and Multivalued Dependencies: Informal design guidelines for			10 Hours

<p>relation schema, Functional Dependencies, Normal Forms based on Primary Keys, Second and Third Normal Forms, Boyce-Codd Normal Form, Multivalued Dependency and Fourth Normal Form, Join Dependencies and Fifth Normal Form. Normalization Algorithms: Inference Rules, Equivalence, and Minimal Cover, Properties of Relational Decompositions, Algorithms for Relational Database Schema Design, Nulls, Dangling tuples, and alternate Relational Designs, Further discussion of Multivalued dependencies and 4NF, Other dependencies and Normal Forms</p> <p>Textbook 1: Ch14.1 to 14.7, 15.1 to 15.6</p>	
Module – 5	
<p>Transaction Processing: Introduction to Transaction Processing, Transaction and System concepts, Desirable properties of Transactions, Characterizing schedules based on recoverability, Characterizing schedules based on Serializability, Transaction support in SQL. Concurrency Control in Databases: Two-phase locking techniques for Concurrency control, Concurrency control based on Timestamp ordering, Multiversion Concurrency control techniques, Validation Concurrency control techniques, Granularity of Data items and Multiple Granularity Locking. Introduction to Database Recovery Protocols: Recovery Concepts, NO-UNDO/REDO recovery based on Deferred update, Recovery techniques based on immediate update, Shadow paging, Database backup and recovery from catastrophic failures</p> <p>Textbook 1: 20.1 to 20.6, 21.1 to 21.7, 22.1 to 22.4, 22.7.</p>	10 Hours
Course outcomes: The students should be able to:	
<ul style="list-style-type: none"> Identify, analyze and define database objects, enforce integrity constraints on a database using RDBMS. Use Structured Query Language (SQL) for database manipulation. Design and build simple database systems Develop application to interact with databases. 	
<p>Question paper pattern: The question paper will have TEN questions. There will be TWO questions from each module. Each question will have questions covering all the topics under a module. The students will have to answer FIVE full questions, selecting ONE full question from each module.</p>	
Text Books:	
<ol style="list-style-type: none"> Database systems Models, Languages, Design and Application Programming, RamezElmasri and Shamkant B. Navathe, 7th Edition, 2017, Pearson. Database management systems, Ramakrishnan, and Gehrke, 3rd Edition, 2014, McGraw Hill 	
Reference Books:	
<ol style="list-style-type: none"> Silberschatz Korth and Sudharshan, Database System Concepts, 6th Edition, McGrawHill, 2013. Coronel, Morris, and Rob, Database Principles Fundamentals of Design, Implementation and Management, Cengage Learning 2012. 	

Introduction

- 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.
- A database has the following implicit properties:
 - i. 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.
 - ii. 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.
 - iii. A database is designed, built, and populated with data for a specific purpose. It has an intended group of users and some preconceived applications in which these users are interested.
- A **database management system (DBMS)** 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.
 - i. **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**.
 - ii. **Constructing** the database is the process of storing the data on some storage medium that is controlled by the DBMS.
 - iii. **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.
 - iv. **Sharing** a database allows multiple users and programs to access the database simultaneously.
 - v. An **application program** accesses the database by sending queries or requests for data to the DBMS.
 - vi. A **query** typically causes some data to be retrieved;
 - vii. **transaction** may cause some data to be read and some data to be written into the database.
 - viii. **Protection** includes system protection against hardware or software malfunction (or crashes) and security protection against unauthorized or malicious access.
 - ix. To **maintain** the database system by allowing the system to evolve as requirements change over time.
- **Figure 1.1** Represents the simplified database system environment.

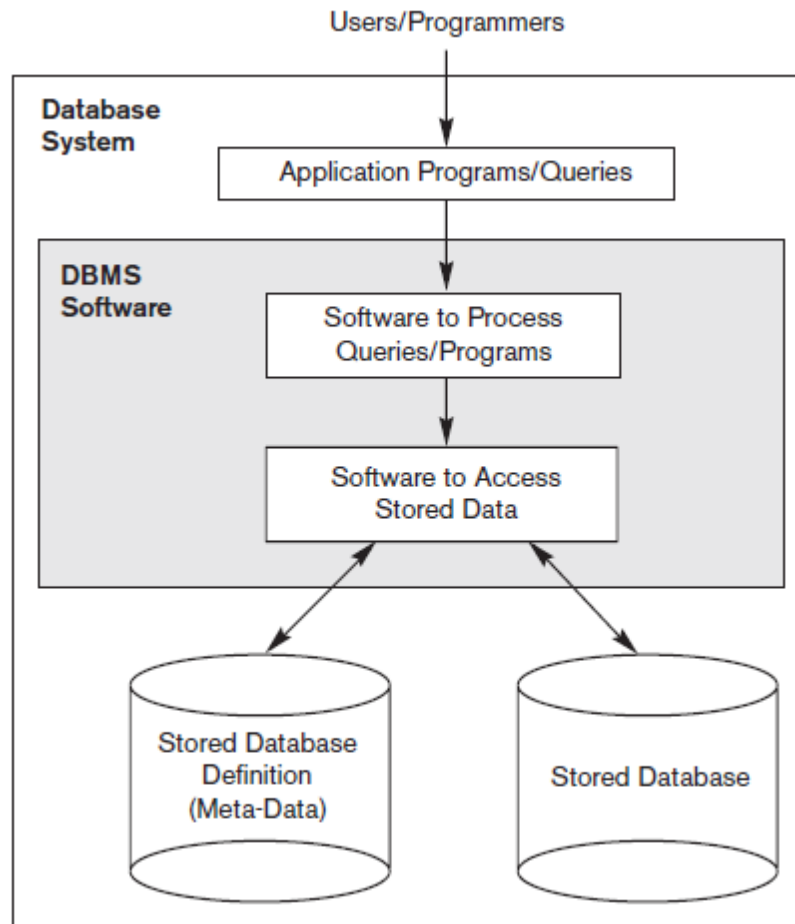


Figure 1.1 A simplified database system environment

1.3: Characteristics of the Database Approach:

- In traditional **file processing**, each user defines and implements the files needed for a specific software application.
For example, one user, the grade reporting office, may keep files on students and their grades. 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. This redundancy in defining and storing data results in wasted storage space and in redundant efforts to maintain common up-to-date information.
- 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 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:
 1. **Self-describing nature of a database system**
 2. **Insulation between programs and data, and data abstraction**

3. Support of multiple views of the data
4. Sharing of data and multiuser transaction processing.

1.3.1 Self-Describing Nature of a Database System.

- The **database system** contains a complete definition 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.
- A **general-purpose DBMS** software package is not written for a specific database application. It refers to the catalog to know the structure of the files in a specific database, such as the type and format of the data.
- In **traditional file processing**, data definition is typically part of the application programs. 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.
- Figure 1.3 ::** An example of a database catalog for the database.

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

Figure 1.3: An example of a database catalog for the database.

1.3.2 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. In DBMS the structure of data files is stored in the **DBMS catalog** separately from the access programs, this property is called **program-data independence**.
- **For example.** Consider a STUDENT record, In a file processing if we want to add another piece of data to STUDENT record, say the Birth_date, such a program will no longer work and must be changed. In a DBMS environment, we only need to change the description of STUDENT records in the catalog to reflect the inclusion of the new data item Birth_date; **no programs are changed**.
- User application programs can operate on the data by invoking the operations through names and arguments, regardless of how the operations are implemented. This is called as **program operation independence**.
- The characteristic that allows program-data independence and program-operation independence is called **data abstraction**. A DBMS provides users with a **conceptual representation** of data that does not include many of the details of how the data is stored or how the operations are implemented.
- A **data model** is a type of data abstraction that is used to provide this conceptual representation. The data model uses logical concepts, such as objects, their properties, and their interrelationships, that may be easier for most users to understand than computer storage concepts. Hence, the data model hides storage and implementation details that are not of interest to most database users.
- **For example,** an operation CALCULATE_GPA can be applied to a STUDENT object to calculate the grade point average. Such operations can be invoked by the user queries or application programs without having to know the details of how the operations are implemented. An abstraction of the miniworld activity is made available to the user as an **abstract operation**.

1.3.3 Support of Multiple Views of the Data.

- A traditional file processing approach supports a single view 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.
- For example, Considering a STUDENT record database, one user of the database may be interested in accessing and printing the transcript of each student (Figure 1.5a). A second user may be interested only in checking that students have taken all the prerequisites of each course for which they register. (Figure 1.5b).

(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 student database.

(a) The TRANSCRIPT view.

(b) The COURSE_PREREQUISITES view.

1.3.4 Sharing of Data and Multiuser Transaction Processing.

- A multiuser DBMS allows 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 includes **concurrency control** software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct
- For example, when several reservation agents try to assign a seat on an airline flight, the DBMS should ensure that each seat can be accessed by only one agent at a time for assignment to a passenger. These types of applications are generally called **online transaction processing (OLTP)** applications.
- A fundamental role of multiuser DBMS software is to ensure that concurrent transactions operate correctly and efficiently. The DBMS enforces 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.

1.4 Actors on the Scene

- The people whose jobs involve the day-to-day use of a large database;

1.4.1 Database Administrators

- The DBA is responsible for authorizing access to the database, coordinating and monitoring its use, and acquiring software and hardware resources as needed. The DBA is accountable for problems such as security breaches and poor system response time.

1.4.2 Database Designers

- **Database designers** are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data.
- Database designers typically interact with each potential group of users and develop **views** of the database that meet the data and processing requirements of these groups. The final database design must be capable of supporting the requirements of all user groups.

1.4.3 End Users

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

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

Example: middle- or high-level managers or other occasional browsers.

- Naive or **parametric end users** make up a sizable portion of database end users. Their main job function revolves around constantly querying and updating the database, using standard types of queries and updates—called **canned transactions**—that have been carefully programmed and tested. The tasks that such users perform are varied:

Example : 1. Bank tellers check account balances and post withdrawals and deposits.

2. Reservation agents for airlines, hotels, and car rental companies check availability for a given request and make reservations.

- **Sophisticated end users** include engineers, scientists, business analysts, and others who thoroughly familiarize themselves with the facilities of the DBMS in order to implement their own applications to meet their complex requirements.

- **Standalone users** maintain personal databases by using ready-made program packages that provide easy-to-use menu-based or graphics-based interfaces.

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.

- **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**.

1.5 Workers behind the Scene

Workers behind the scene are those who 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.

- **DBMS system designers and implementers** design and implement the DBMS modules and interfaces as a software package. 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**—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.

1.6 Advantages of Using the DBMS Approach.

1.6.1 Controlling Redundancy.

- In traditional file processing technique, every user group maintains its own files for handling its data-processing applications, each group independently keeps files which leads to redundancy. **Redundancy** is storing the same data multiple times in the database.
- Redundancy leads to several problems, Firstly 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.
- 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, it is designed to store each logical data item such as students name or birth date in only one place in the database. This is known as data normalization, it ensures consistency and saves storage space. By controlling redundancy the performance of queries is improved.

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 only authorized persons are allowed to access such data.
- A DBMS should provide a **security and authorization subsystem**, which the DBA uses to create accounts and to specify account restrictions.

- For example, only the dba's staff may be allowed to use certain **privileged software**, such as the software for creating new accounts.

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**.
- In Programming languages the values of program variables or objects are discarded once a program terminates, unless the programmer explicitly stores them in permanent files, which often involves converting these complex structures into a format suitable for file storage.
- Traditional old database systems often suffer from **impedance mismatch problem**, since the data structures provided by the traditional DBMS were incompatible with the programming language's data structures.
- **Object-oriented database systems** typically offer data structure **compatibility** with one or more object-oriented programming languages.

1.6.4 Providing Storage Structures and Search Techniques for Efficient Query Processing.

- Database systems provide capabilities for efficiently executing queries and updates.
- The database is typically stored on disk; the DBMS provides specialized data structures and search techniques to speed up disk search for the desired records.
- The DBMS often has a **buffering** or **caching** module that maintains parts of the database in main memory buffers. 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.

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.

1.6.6 Providing Multiple User Interfaces.

- Different 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 standalone users. Both forms-style interfaces and menu-driven interfaces are commonly known as **graphical user interfaces (GUIs)**.

1.6.7 Representing Complex Relationships among Data.

- A database may include numerous varieties of data that are interrelated in many ways.

- A DBMS must have the capability to represent a variety of complex relationships among the data, to define new relationships as they arise, and to retrieve and update related data easily and efficiently.

1.6.8 Enforcing Integrity Constraints

- The simplest type of integrity constraint is specifying a data type for each data item. For example: The value of Name must be a string of no more than 30 alphabetic characters.
- A more complex type of constraint is specifying that a record in one file must be related to records in other files. For example every section record must be related to a course record. This is known as a **referential integrity** constraint.
- Another type of constraint specifies uniqueness on data item values, such as every course record must have a unique value for Course_number. This is known as a **key** or **uniqueness** constraint. It is the responsibility of the database designers to identify integrity constraints during database design.

1.6.9 Permitting Inferencing and Actions Using Rules.

- In a **deductive** database system, one may specify declarative rules that allow the database to infer new data. E.g., Figure out which students are on academic probation.
- A **database trigger** and **active database systems** provide active rules that can automatically initiate actions when certain events and conditions occur.

1.6.10 Additional Implications of Using the Database Approach

- **Flexibility.** It may be necessary to change the structure of a database as requirements change.
- **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.
- **Potential for Enforcing Standards.** The database approach permits the DBA to define and enforce standards among database users in a large organization. Standards can be defined for names and formats of data elements, display formats, report structures, terminology, and so on.

A Brief History of Database Applications

1. Early Database Applications Using Hierarchical and Network Systems:

- Many early database applications maintained records in large organizations, such as corporations, universities, hospitals, and banks. In many of these applications, there were large numbers of records of similar structure.
- 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. These systems did not provide sufficient data abstraction and program-data independence capabilities.
- 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.

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.
- 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.

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.

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 Markup 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.
- A variety of techniques were developed to allow the interchange of data on the Web. Currently, XML (eXtended Markup 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 modeling concepts.

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 some examples of these applications:

- 1. Scientific** applications that store large amounts of data resulting from scientific experiments in areas such as high-energy physics, the mapping of the human genome, and the discovery of protein structures.

2. Storage and retrieval of **images**, including scanned news or personal photographs, satellite photographic images, and images from medical procedures such as x-rays and MRIs (magnetic resonance imaging).
3. Storage and retrieval of **videos**, such as movies, and **video clips** from news or personal digital cameras.
4. **Data mining** applications that analyze large amounts of data searching for the occurrences of specific patterns or relationships, and for identifying unusual patterns in areas such as credit card usage.
5. **Spatial** applications that store spatial locations of data, such as weather information, maps used in geographical information systems, and in automobile navigational systems

1.8 When Not to Use a DBMS

The overhead costs of using a DBMS are due to the following:

- High initial investment in hardware, software, and training.
- The generality that a DBMS provides for defining and processing data.
- Overhead for providing security, concurrency control, recovery, and integrity functions.

Therefore, it may be more desirable to use regular files under the following circumstances:

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

2.1 Data Models, Schemas, and Instances.

- One fundamental characteristic of the database approach is that it provides some level of data abstraction by hiding details of data storage that are irrelevant to database users.
- A **data model** ---a collection of concepts that can be used to describe the conceptual/logical structure of a database--- provides the necessary means to achieve this abstraction.

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, whereas **low-level** or **physical data models** provide concepts that describe the details of how data is stored on the computer storage media.
- Concepts provided by low-level data models are generally meant for computer specialists, not for end users.
- **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.

- **Representational or implementation data models** are used most frequently in traditional commercial DBMSs. These include the widely used **relational data model**, as well as the so-called legacy data models—the **network** and **hierarchical models**— Representational data models represent data by using record structures and hence are sometimes called **record-based data models**.
- **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.

2.1.2 Schemas, Instances, and Database State.

Figure 2.1
Schema diagram for the
database in Figure 1.2.

STUDENT

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

COURSE

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

PREREQUISITE

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

SECTION

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

GRADE_REPORT

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

- The description of a database is called the **database schema**, which is specified during database design and is not expected to change frequently. Figure 2.1 represents the schema diagram for a database
- 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;
- The DBMS is partly responsible for ensuring that every state of the database is a **valid state**—that is, a state that satisfies the structure and constraints specified in the schema.
- The DBMS stores the descriptions of the schema constructs and constraints—also called the **meta-data**—in the DBMS catalog 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.

2.2 Three-Schema Architecture and Data Independence

2.2.1 The Three-Schema Architecture:

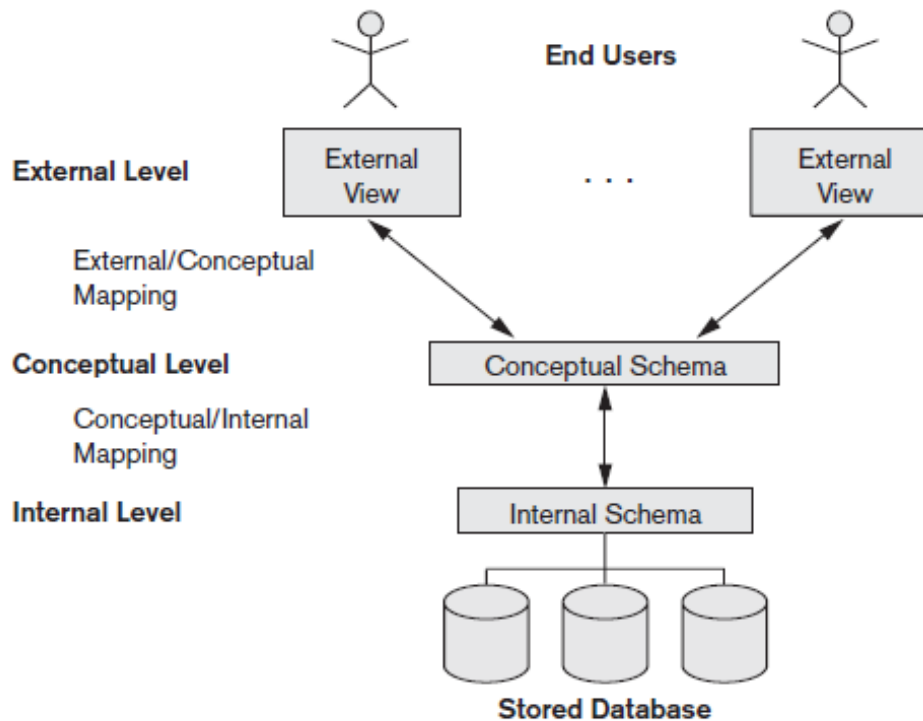


Figure 2.2 The three-schema architecture.

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

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. A representational data model is used to describe the conceptual schema when a database system is implemented.
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.

2.2.2 Data Independence:

The ability to modify a scheme definition in one level without affecting a scheme definition in a higher level is called **data independence**

The two types of data independence are:

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

- The conceptual schema could be changed 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).
- The view definition and the mappings need to be changed in a DBMS that supports logical data independence.
- After the conceptual schema undergoes a logical reorganization, application programs that reference the external schema constructs must work as before. Changes to constraints can be applied to the conceptual schema without affecting the external schemas or application programs.

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

2.3 Database Languages and Interfaces.

2.3.1 DBMS Languages

- DDL – the **data definition language**, used by the DBA and database designers to define the conceptual and internal schemas.
- The DBMS has a DDL compiler to process DDL statements in order to identify the schema constructs, and to store the description in the catalogue.
- In databases where there is a separation between the conceptual and internal schemas, DDL is used to specify the conceptual schema, **storage definition language**, is used to specify the internal schema.
- Here are the lists of tasks that come under DDL:
CREATE – used to create objects in the database
ALTER – used to alters the structure of the database
DROP – used to delete objects from the database
TRUNCATE – used to remove all records from a table, including all spaces allocated for the records are removed
- For a true three-schema architecture, VDL, **view definition language**, is used to specify the user views and their mappings to the conceptual schema. But in most DBMSs, the DDL is used to specify both the conceptual schema and the external schemas.

- Once the schemas are compiled, and the database is populated with data, users need to manipulate the database. Manipulations include retrieval, insertion, deletion and modification. The DBMS provides operations using the DML, **data manipulation language**.
- Here are the lists of tasks that come under DML:

SELECT – It retrieve data from the a database

INSERT – It inserts data into a table

UPDATE – It updates existing data within a table

DELETE – It deletes all records from a table, the space for the records remain.

- **There are two main types of Data Manipulation Languages (DMLs).**

1. High-level/Non procedural

- Can be used on its own to specify complex database operations.
- DBMSs allow DML statements to be entered interactively from a terminal, or to be embedded in a programming language.
- High-level DMLs, such as SQL can specify and retrieve many records in a single DML statement, and are called set at a time or set oriented DMLs.
- High-level languages are often called declarative, because the DML often specifies what to retrieve, rather than how to retrieve it.

2. Low Level/Procedural

- Must be embedded in a general purpose programming language.
- Typically retrieves individual records or objects from the database and processes each separately. Therefore it needs to use programming language constructs such as loops.

2.3.2 DBMS Interfaces.

Types of interfaces provided by the DBMS include:

1. Menu-Based Interfaces for Web Clients or Browsing

- Present users with list of options (menus) that lead user through formulation of request.
- Query is composed of selection options from a menu displayed by system.
- 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.

2. Forms-Based Interfaces

- Displays a form to each user.
- User can fill out form to insert new data or fill out only certain entries.
- Designed and programmed for naïve users as interfaces to canned transactions.

3. Graphical User Interfaces

- Displays a schema to the user in diagram form.
- The user can specify a query by manipulating the diagram. GUIs use both forms and menus.
- GUIs use a **pointing device**, such as a mouse, to select certain parts of the displayed schema diagram.

4. Natural Language Interfaces

- Accept requests written in English, or other languages and attempt to understand them.
- Interface has its own schema, and a dictionary of important words. Uses the schema and dictionary to interpret a natural language request.
- If the interpretation is successful, the interface generates a high-level query corresponding to the natural language request and submits it to the DBMS for processing;

5. Speech Input and Output.

- Speech as an input query and speech as an answer to a question or result of a request.
- 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.

6. Interfaces for Parametric Users

- Parametric users have small set of operations they perform.
- Analysts and programmers design and implement a special interface for each class of naïve users.
- Often a small set of commands included to minimize the number of keystrokes required. (I.e. function keys)

7. Interfaces for the DBA

- Systems contain privileged commands only for DBA staff.
- Include commands for creating accounts, setting parameters, authorizing accounts, changing the schema, reorganizing the storage structures etc.

2.4 The Database System Environment

2.4.1 DBMS Component Modules.

- The typical DBMS components are represented in Figure 2.3. It 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. The **stored data manager** module of the DBMS controls access to DBMS information that is stored on disk.
- The top part of Figure shows interfaces for the **DBA staff** who defines the database and manipulate its definition using the DDL and other privileged commands, **casual users** who work with interactive interfaces to formulate queries, **application programmers** who create programs using some host programming languages, and **parametric users** who do data entry work by supplying parameters to predefined transactions.
- The **DDL compiler** processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta-data) in the DBMS catalog.
- Casual users interact with the **interactive query** interface. The queries are parsed and validated for correctness of the query syntax, the names of files and data elements by a **query compiler**.

- This internal query is subjected to query optimization. The **query optimizer** is concerned with the ordering of operations, elimination of redundancies, and use of correct algorithms and indexes during execution.

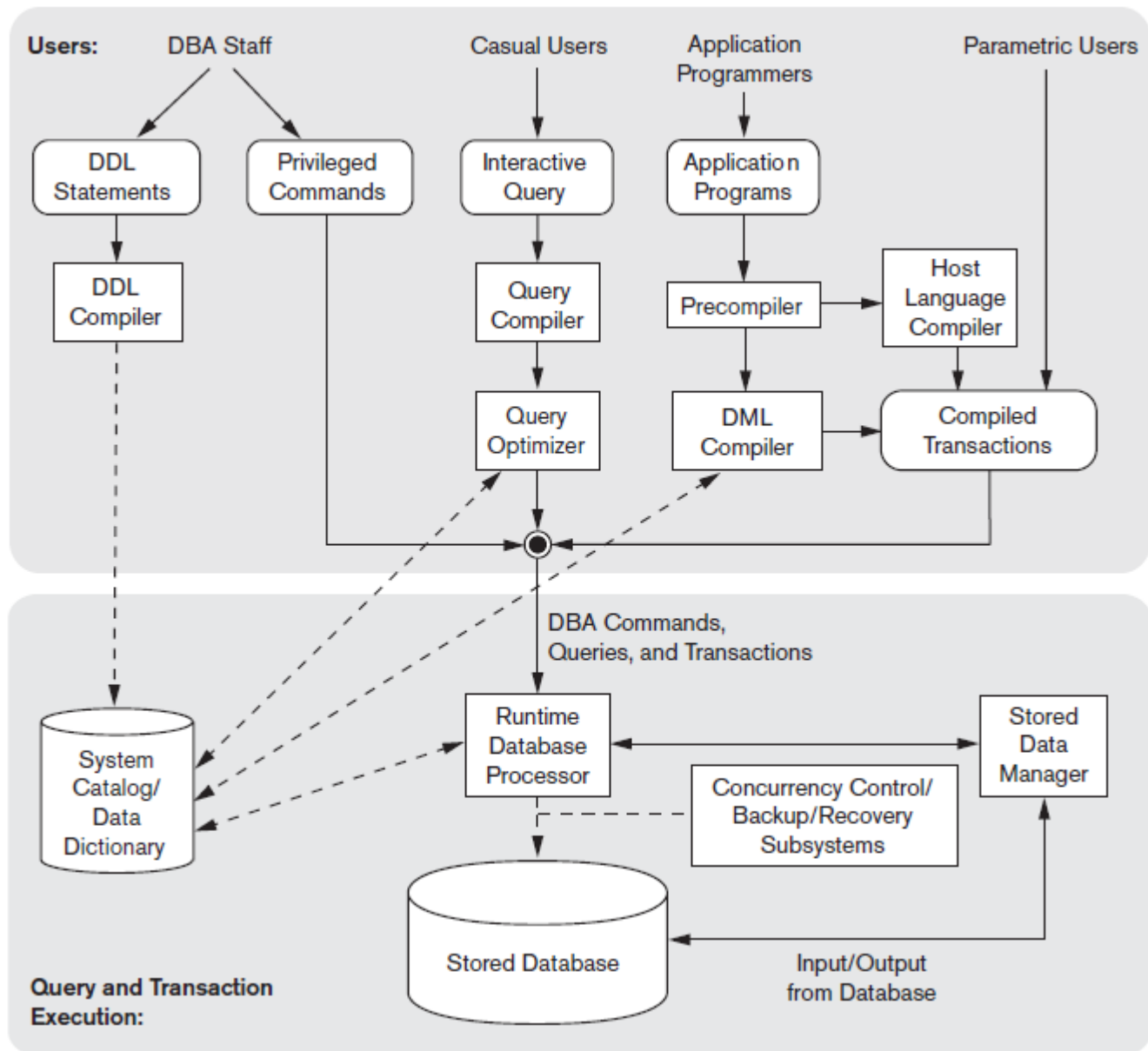


Figure 2.3 Component modules of a DBMS and their interactions.

- The **query optimizer** consults the system catalog for statistical and other physical information about the stored data and generates executable code that performs the necessary operations for the query and makes calls on the **runtime processor**.
- **Application programmers** write programs in host languages such as Java, C, or C++ that are submitted to a precompiler. 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**.
- **Canned transactions** are executed repeatedly by parametric users, who simply supply the parameters to the transactions. Each execution is considered to be a separate transaction.
Example: Bank withdrawal transaction where the account number and the amount is supplied as parameters.
- In the lower part of Figure, the **runtime database processor** executes
 1. The privileged commands,
 2. The executable query plans,
 3. The canned transactions with runtime parameters.

It works with the **system catalog** and update it with statistics. It also works with the **stored data manager**, which in turn uses basic operating system services for carrying out low-level input/output (read/write) operations between the disk and main memory.

- The **concurrency control** and **backup and recovery systems** are integrated into the working of the runtime database processor for purposes of transaction management.

2.4.2 Database System Utilities:

DBMSs have database utilities that help the DBA manage the system. Functions include:

- **Loading** – A loading utility is used to load existing text/sequential files into the database. Source format and desired target file are specified to the utility, and the utility reformats the data to load into a table.
- **Backup** – A backup utility creates a backup copy of the database, usually by dumping database onto tape. Can be used to restore the database in case of failure. Incremental backup can be used which records only the changes since the last backup.
- **File Reorganization** – This utility reorganize database files into different file organizations to improve performance.
- **Performance Monitoring** – monitors database usage and provides statistics to the DBA. DBA uses the statistics for decision-making. 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.

2.4.3 Tools, Environments and Communication Facilities:

- **CASE Tools** – used in the design phase to help speed up the development process.
- **Data dictionary system** – stores catalog information about schemas and constraints, as well as design decisions, usage standards, application program descriptions, user information. Also called an information repository. Can be accessed directly by DBA or users when needed.
- **Application development environments** – (i.e. JBuilder) provide environment for developing database applications, and include facilities to help in database design, GUI development, querying and updating and application development.
- **Communication software** – allow users at remote locations to access the database through computer terminals, workstations or personal computers. Connected to the database through data communications hardware such as phone lines, local area networks etc.

2.5 Centralized and Client Server Architectures for DBMSs

2.5.1 Centralized DBMS Architecture

- Uses mainframes to provide main processing for user application programs, user interface programs and DBMS functionality
- User accessed systems via 'dumb' computer terminals that only provided display capabilities, with no processing capabilities.
- All processing was performed remotely on the computer system, and only display information was sent to the terminals, connected via a network.
- Dumb terminals were replaced with workstations, which lead to the client/server architecture.

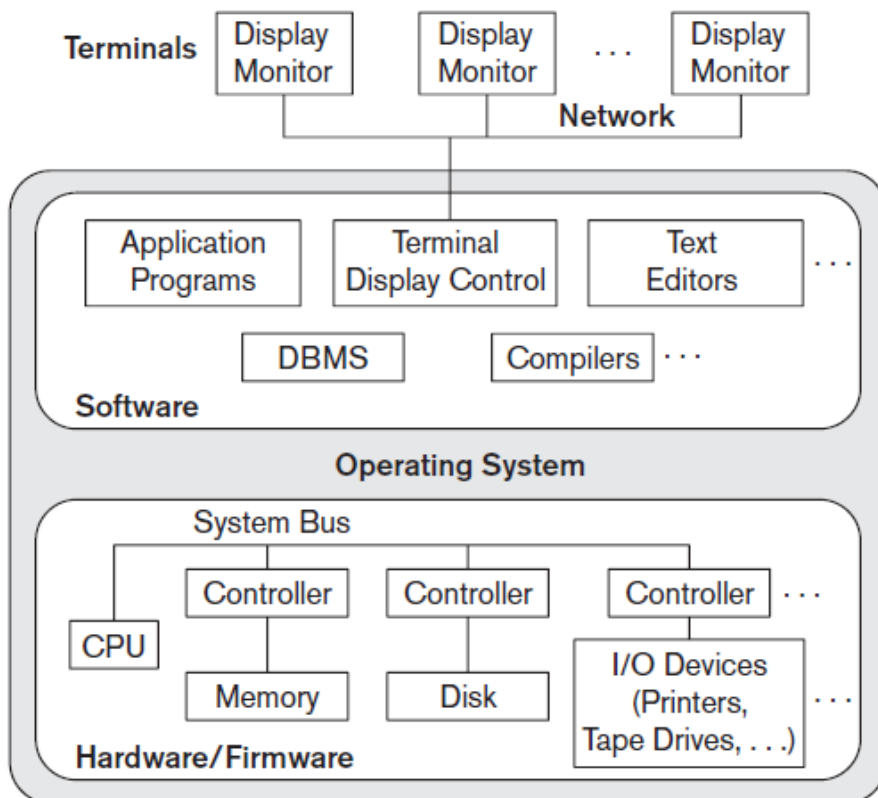


Figure 2.4
A physical centralized architecture.

2.5.2 Basic Client/Server Architectures

- Define specialized servers with specific functionalities (file servers, print servers, web servers, database servers)
- Many client machines can access resources provided by specialized server.
- Client machines provide user with the appropriate interfaces to utilize servers, as well as with local processing power to run local applications.
- Some machines are client sites, with client software installed and other machines are dedicated servers.
- **Client** – a user machine that provides user interface capabilities and local processing.
- **Server** – machine that provides services to client machines such as file access, printing, and database access.

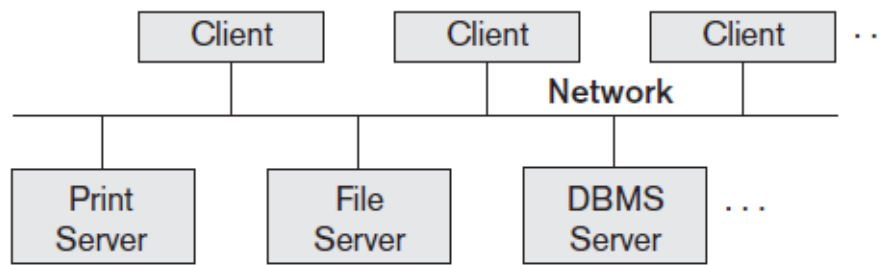


Figure 2.5
Logical two-tier
client/server
architecture.

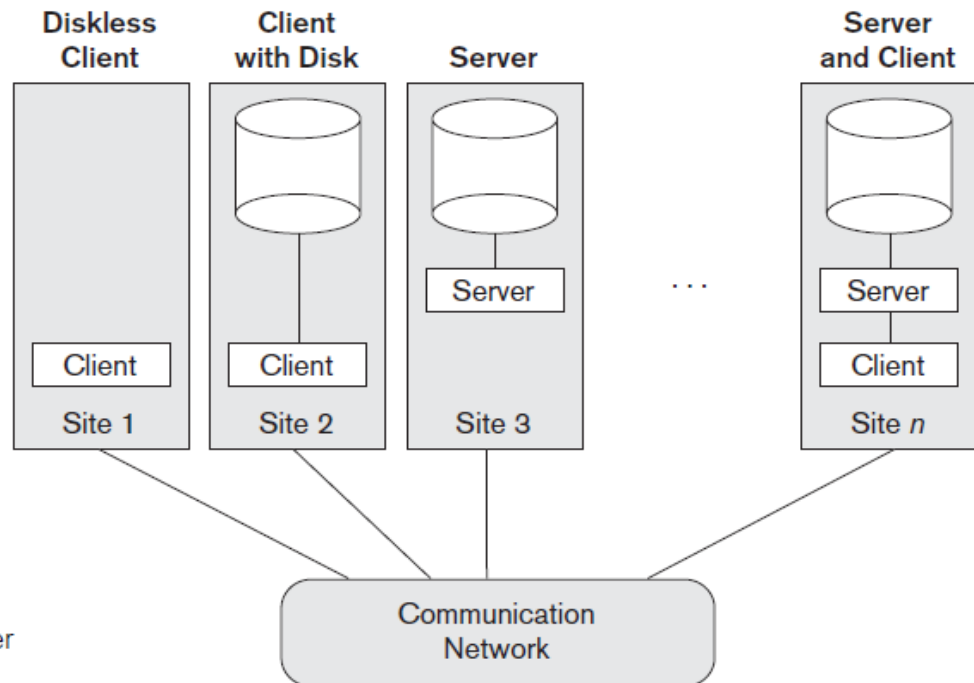


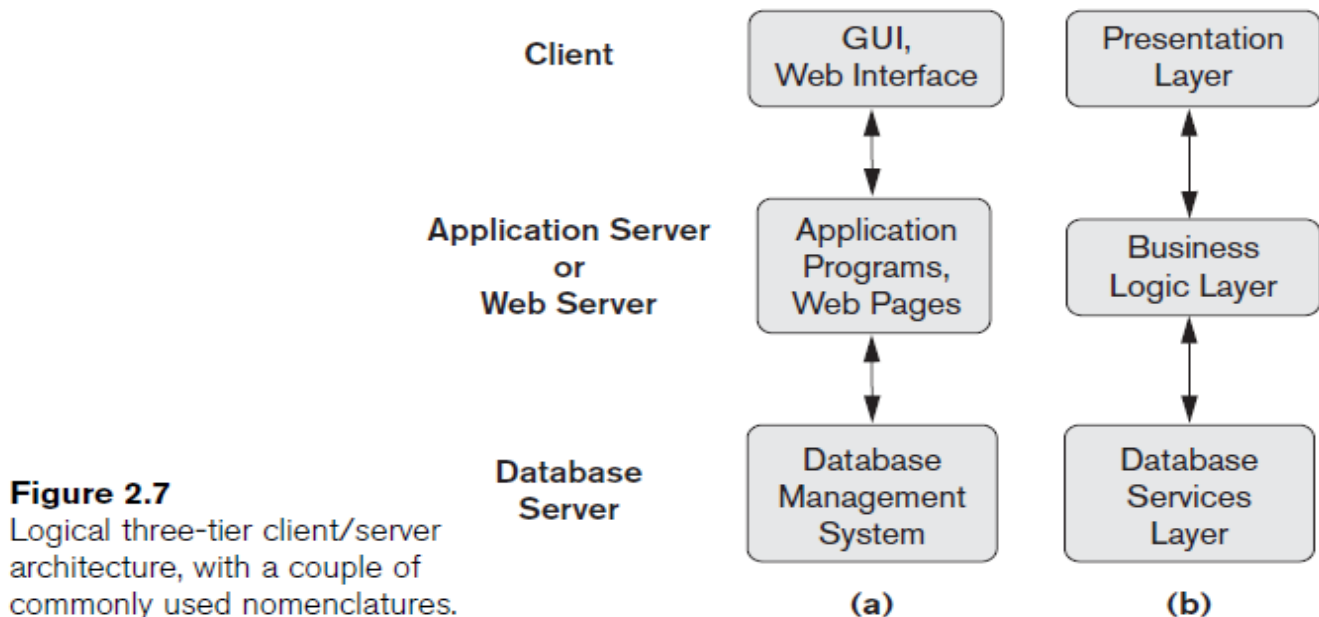
Figure 2.6
Physical two-tier client/server
architecture.

2.5.3 Two-Tier Client/Server Architectures for DBMSs

Two Tier Client/Server Architecture for DBMSs

- In relational DBMSs, user interfaces and application programs were first moved to the client side.
- SQL provided a standard language, which was a logical dividing point between client and server.
- Query and transaction functionality remained on server side. In this architecture, the server is called a query server, or transaction server.
- In relational DBMSs, the server is called an SQL server, because most RDBMSs use SQL.
- ODBC (Open Database Connectivity) is a standard that provides an application processing interface which allows client side programs to call the DBMS as long as both sides have the required software. Most database vendors provide ODBC drivers for their systems.
- Client programs can connect to several RDBMS and send query and transaction requests using the ODBC API
- Query requests are sent from the client to the server, and the server processes the request and sends the result to the client.
- A related Java standard is JDBC, which allows Java programs to access the DBMS through a standard interface.

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



Three-Tier Client Server Architecture for Web Applications

- Many web applications use three-tier architecture, which adds an intermediate layer between the client and the database server.
- The middle tier is called the application server, or the web server. Plays an intermediate role, by storing business rules (procedures/constraints) used to access data from database.
- Can improve database security by checking the clients credentials before forwarding request to database server.
- Clients contain GUI interfaces and application specific rules.
- The intermediate server accepts the requests from the client, processes the request and sends the database commands to the db server, then passes the data from the database server to the client, where it may be processes further and filtered. The three tiers are: user interface, application rules, and data access.

2.6 Classification of Database Management Systems.

1. **Based on the data model Relational database** . It is based on the SQL. Few examples are MYSQL(Oracle, open source).
 - Object oriented database –. It adds the database functionality to object programming languages.
 - Object relational database – Relational DBMS are evolving to a new class called extended relational database or object relational database.
 - Hierarchical database –The information about the groups of parent or child relationships is present in the records which is similar to the structure of a tree.
 - Network database –If there are more connections, then this database is efficient.
2. **Based on the number of users:**
 - Single user – support only one user at a time and are mostly used with Pc.
 - Multiple users –Includes majority of DBMS It supports multiple users concurrently.

3. Based on the number of sites over which database is distributed

- **Centralized database system** – The DBMS and database are stored at the single site that is used by several other systems too.
- **Distributed database system** – In this data and the DBMS software are distributed over several sites but connected to the single computer.

Further they are classified as

- 1.Homogeneous DBMS – They use same DBMS software at multiple sites.
- 2.Heterogeneous DBMS – They use different DBMS software at each site.

4. Based on the cost.

- It is difficult to propose classification on cost. There is open source dbms products like MySQL and PostgreSQL.
- Dbms software 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.

5. Based on the usage

- Online transaction processing(OLTP) DBMS – They manage the operational data. Database server must be able to process lots of simple transactions per unit of time.
- Online analytical processing(OLAP) DBMS – They use the operational data for tactical and strategical decision making. They have limited users deal with huge amount of data, complex queries.

7.1 Using High-Level Conceptual Data Models for Database Design

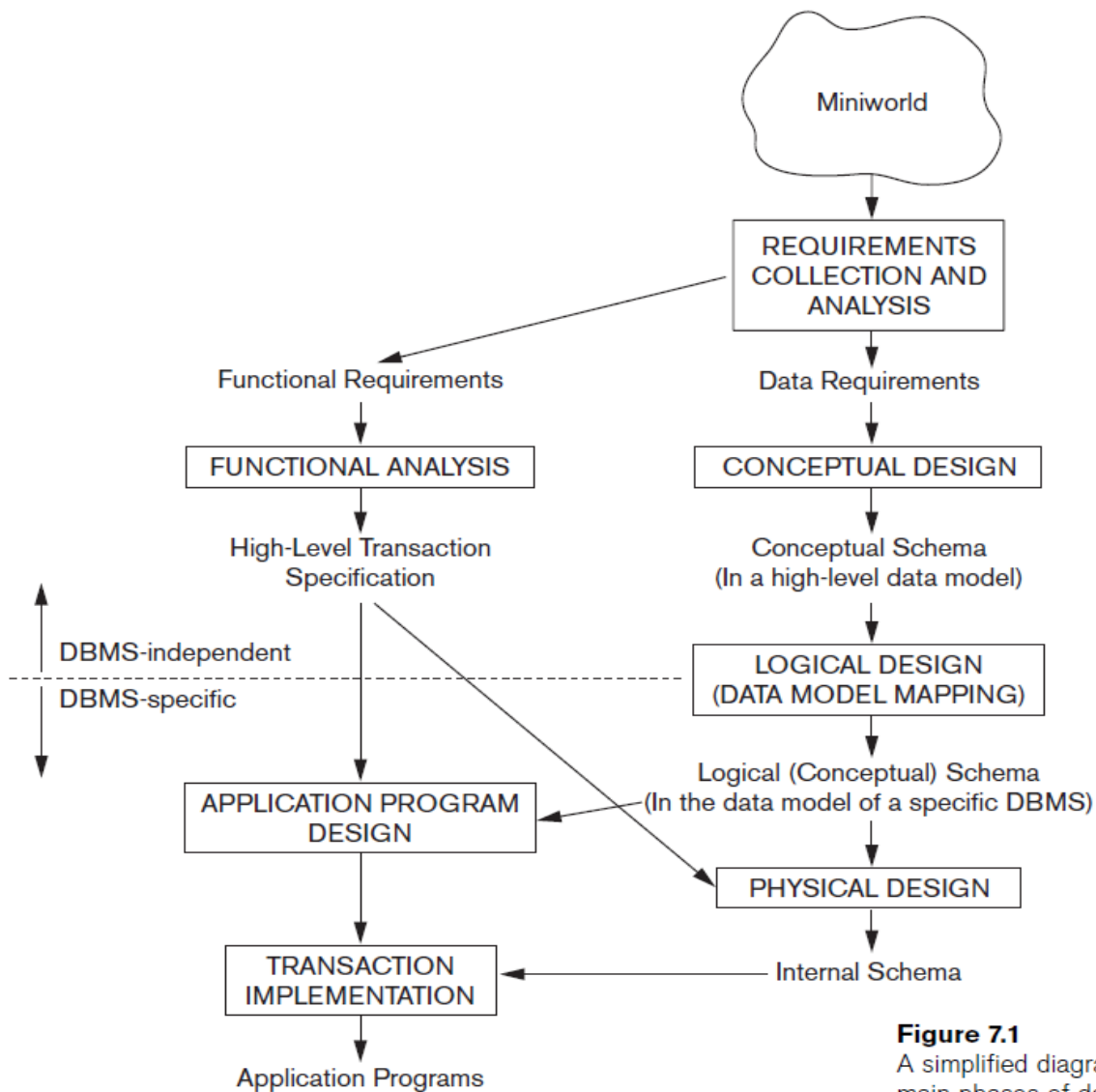


Figure 7.1

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

Database Design Process

The database design process consists of a number of steps listed below.

Step 1: Requirements Collection and Analysis

- Prospective users are interviewed to understand and document data requirements
- This step results in a concise set of user requirements, which should be detailed and complete.
- The functional requirements should be specified, as well as the data requirements. Functional requirements consist of user operations that will be applied to the database, including retrievals and updates.
- Functional requirements can be documented using diagrams such as sequence diagrams, data flow diagrams, scenarios, etc.

Step 2: Conceptual Design

- Once the requirements are collected and analyzed, the designers go about creating the conceptual schema.
- Conceptual schema: concise description of data requirements of the users, and includes a detailed description of the entity types, relationships and constraints.
- The concepts do not include implementation details; therefore the end users easily understand them, and they can be used as a communication tool.
- The conceptual schema is used to ensure all user requirements are met, and they do not conflict.

Step 3: Database Implementation

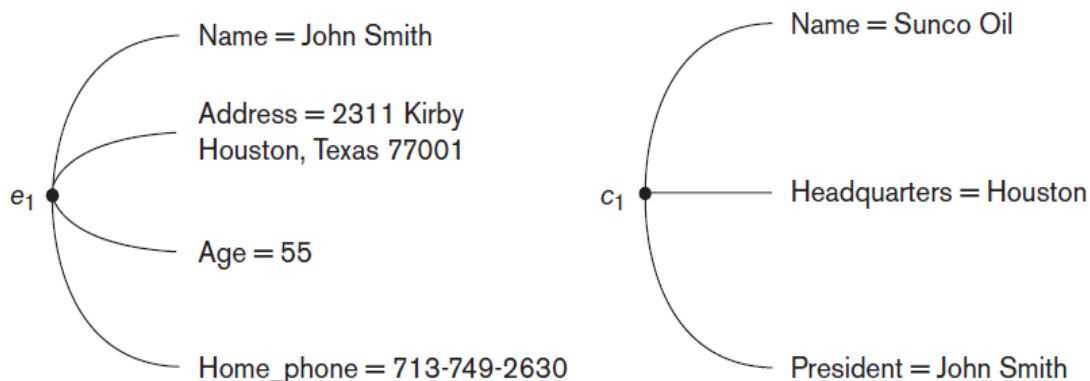
- Many DBMS systems use an implementation data model, so the conceptual schema is transformed from the high-level data model into the implementation data model.
- This step is called logical design or data model mapping, which results in the implementation data model of the DBMS.

Step 4: Physical Design

- Internal storage structures, indexes, access paths and file organizations are specified.
- Application programs are designed and implemented.

7.3 Entity Types, Entity Sets, Attributes and Keys**7.3.1 Entities and Attributes.**

- The basic concept of an ER diagram is the entity. **An entity represents a ‘thing’ or ‘object’ in the real world.**
- Examples of entities might be an object with physical existence, such as a student, a house, a product etc, or conceptual entities such as a company, a job position, a course, etc.
- Entities have **attributes, which basically are the properties/characteristics of a particular entity.**

**Figure 7.3**

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

Figure 7.3 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:

1. **Simple versus composite,**
2. **Singlevalued versus multivalued,**
3. **Stored versus derived.**

1. Simple vs. Composite Attributes

- **Composite attributes can be divided into smaller subparts, which represent more basic attributes, independent meanings.**
- A common example of a composite attribute is Address. Address can be broken down into a number of subparts, such as Street Address, City, Postal Code. Street Address may be further broken down by Number, Street Name and Apartment/Unit number.

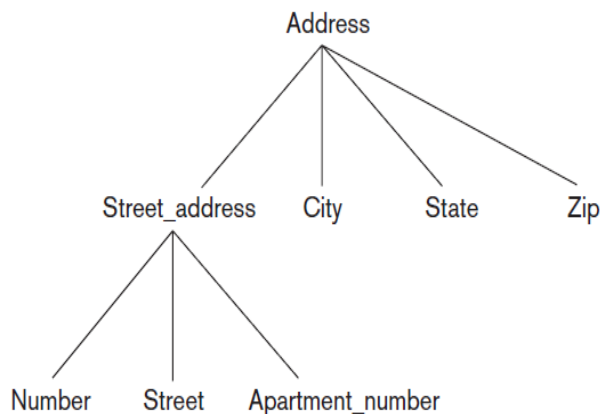


Figure 7.4

A hierarchy of composite attributes.

- **Attributes that are not divisible into subparts are called simple or atomic attributes.**
- Composite attributes can be used if the attribute is referred to as the 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.

2. Single-Valued vs. Multi-valued Attributes

- **The attributes that have a single value for a particular entity, such a attribute are called Single valued.** For example **Age** is a single valued attribute of a person.
- **An attribute can have multiple values for a single entity, These attributes are called multi-valued attributes.** For example, a doctor may have more than one specialty (or may have only one specialty), a customer may have more than one mobile phone number, or they may not have one at all.

- Multi-valued attributes may have a lower and upper bounds to constrain the number of values allowed. For example, a doctor must have at least one specialty, but no more than 3 specialties.

3. Stored vs. Derived Attributes

- **If an attribute can be determined using the value of another attribute, they are called derived attributes.**
- **The attribute that is used to derive the attribute is called a stored attribute.**
- 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.
- Derived attributes are not stored in the file, but can be derived when needed from the stored attributes.

4. Null Valued Attributes

- There are cases where an attribute does not have an applicable value for an attribute. For these situations, the value null is created.
- A person who does not have a mobile phone would have null stored at the value for the Mobile Phone Number attribute.
- Null can also be used in situations where the attribute value is unknown. There are two cases where this can occur. 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.

5. Complex Attributes

- Complex attributes are attributes that are nested in an arbitrary way.
- For example a person can have more than one residence, and each residence can have more than one phone, therefore it is a complex attribute that can be represented as:

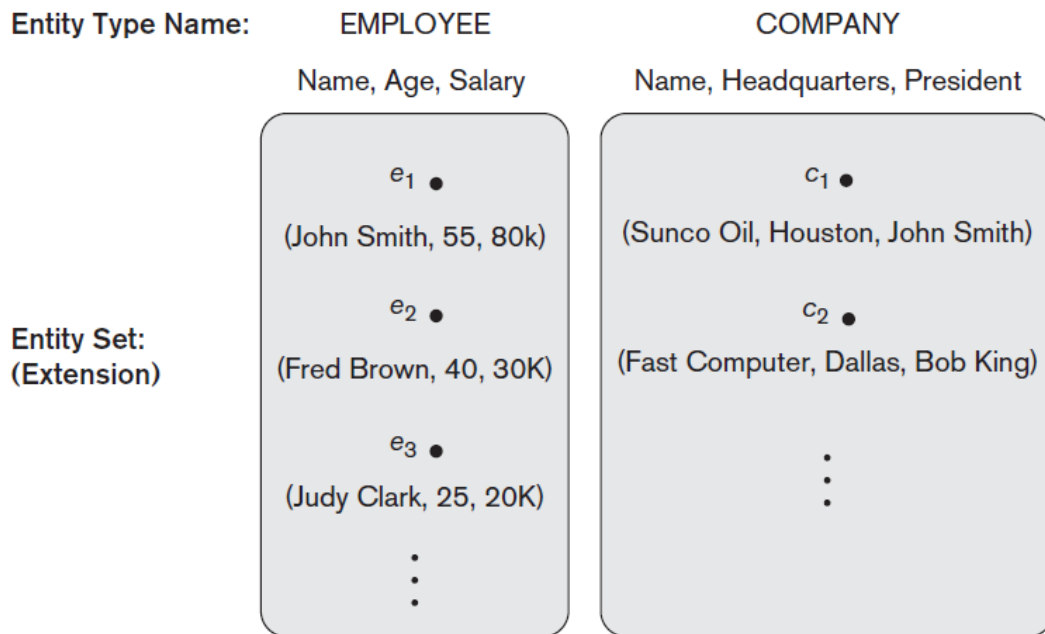
```
{Address_phone( {Phone(Area_code,Phone_number)},Address(Street_address  
(Number,Street,Apartment_number),City,State,Zip) )}
```

Figure 7.5

A complex attribute:
Address_phone.

- The composite and multivalued attributes can be nested arbitrarily by grouping components of a composite attribute between parentheses () and separating the components with commas, and by displaying multivalued attributes between braces { }. Such attributes are called **complex attributes**.

7.3.2 Entity Types, Entity Sets, Keys, and Value Sets

**Figure 7.6**

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

1. Entity Types and Entity Sets

- An **Entity Type** defines a collection or set of entities that have the same attributes. Entities are the instances of people, places, things, or events that are of interest. Each entity type in the database is described by its name and attributes. The entity share the same attributes, but each entity has its own value for each attribute.
- Figure 7.6 shows two entity types: EMPLOYEE and COMPANY, and a list of some of the attributes for each.
- An entity type named Student defines a collection of student entities.
- The collection of all entities of a particular entity type in the database at any point in time is called an **entity set**.

Entity Set Example:

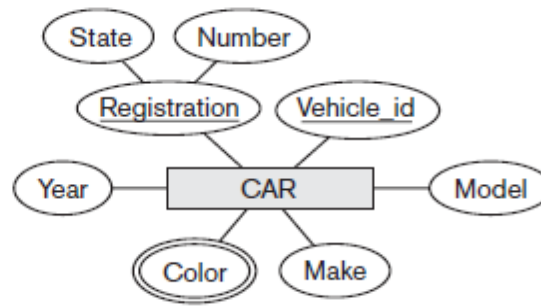
- Entity Type: Student
- Entity Set:
 - [123, John, Smith, 12/01/1981, Computer Technology]
 - [456, Jane, Doe, 05/02/1979, Mathematics]
 - [789, Semra, Aykan, 02/08/1980, Linguistics]

- An entity type is represented in ER diagrams as a **rectangular box**. Attribute names are enclosed in **ovals** and are attached to their entity type by straight lines. Composite attributes are attached to their component attributes by **straight lines**. Multivalued attributes are displayed in **double ovals**. Figure 7.7(a) shows a CAR entity type in this notation.
- The entity type describes the **intension**, or schema for a set of entities that share the same structure. The collection of entities of a particular entity type is grouped into the entity set, called the **extension**.

Figure 7.7

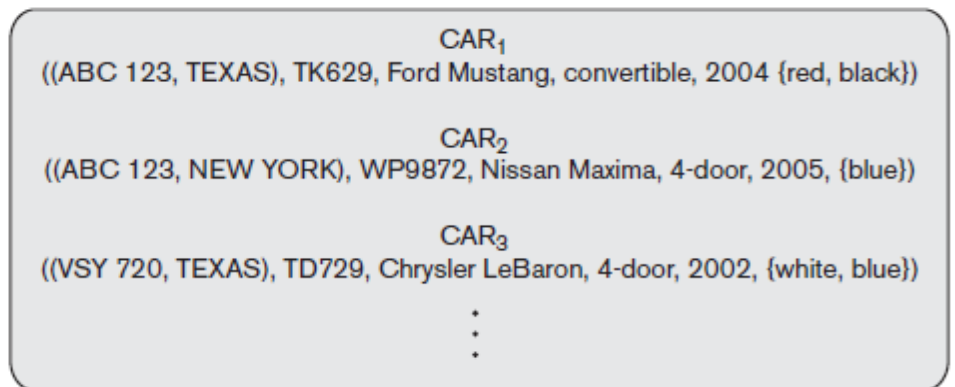
The CAR entity type with two key attributes, Registration and Vehicle_id. (a) ER diagram notation. (b) Entity set with three entities.

(a)



(b)

CAR
Registration (Number, State), Vehicle_id, Make, Model, Year, {Color}



2. Key Attributes of an Entity Type.

- An important constraint on entities of an entity type is the uniqueness constraint.
- A **key attribute** is an attribute whose values are distinct for each individual entity in the entity set.
- The values of the key attribute can be used to identify each entity uniquely.
- Example : The Name attribute is a key of the COMPANY entity type because no two companies are allowed to have the same name. For the PERSON entity type, a typical key attribute is Ssn.
- In ER diagrammatic notation, each key attribute has its name **underlined** inside the oval.
- Sometimes a key can consist of several attributes together, where the combination of attributes is unique for a given entity. This is called a composite key. Composite keys should be minimal, meaning that all attributes must be included to have the uniqueness property.
- An entity can have more than one key attribute. For example, Vehicle_id and Registration attributes of the entity type CAR (Figure 7.7) is a key
- Those entities with no key attribute are called **weak entity types**.

3. Value Sets (Domains) of Attributes.

- Each simple attribute of an entity is associated with a domain of values, or value set, which specifies the set of values that may be assigned to that attribute for each entity.
- For example, date of birth must be before today's date, and after 01/01/1900, or the Student Name attribute must be a string of alphabetic characters.
- Value sets are not specified in ER diagrams.

7.4 Relationship Types, Relationship Sets, Roles, & Structural Constraints

7.4.1 Relationship Types, Sets, and Instances

- A **relationship type** R among n entity types E_1, E_2, \dots, E_n defines a set of associations among entities from these entity types.
- Relationship instance: Each relationship instance r_i in R is an association of entities, where the association includes exactly one entity from each participating entity type.
- For example, consider a relationship type **WORKS_FOR** between the two entity types **EMPLOYEE** and **DEPARTMENT**. Each relationship instance in the relationship set **WORKS_FOR** associates one **EMPLOYEE** entity and one **DEPARTMENT** entity.

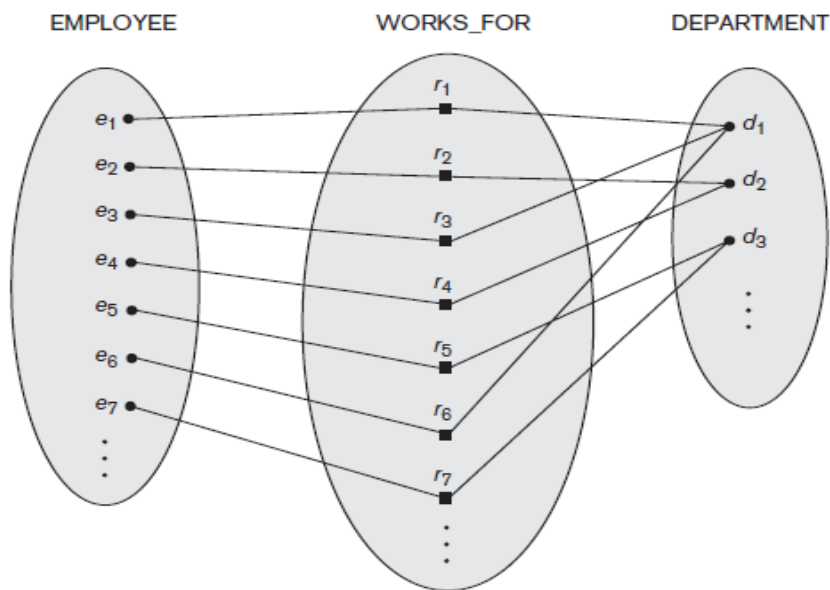


Figure 7.9
Some instances in the **WORKS_FOR** relationship set, which represents a relationship type **WORKS_FOR** between **EMPLOYEE** and **DEPARTMENT**.

- 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.

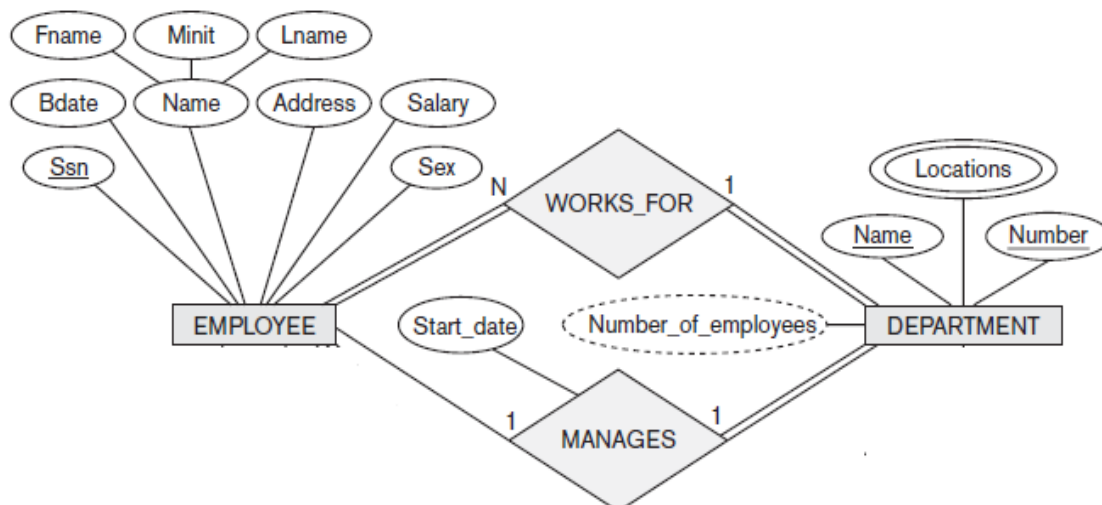
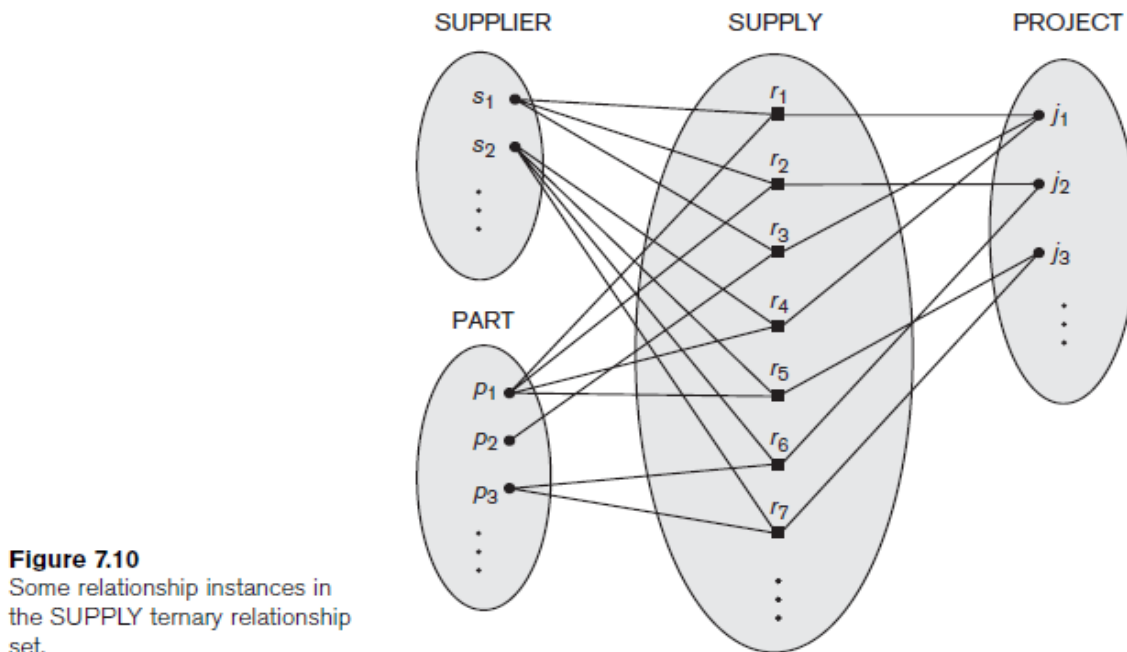


Figure: Representation of ER diagram for relationship.

7.4.2 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. Example 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 7.10, 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 .



- For example, if we have three entities, Supplier, Project and Part. Each part is supplied by a unique supplier, and is used for a given project within a company; the relationship “Supplies” is a ternary (degree of three) between Supplier, Project and Part, meaning all three participate in the supplies relationship.

Role Names and Recursive Relationships.

- Each entity type in a relationship plays a particular role. The **role name** specifies the role that a participating entity type plays in the relationship and explains 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.
- 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**.
- **For example**, in the Company schema, each employee has a supervisor, we need to include the relationship “Supervises”, however a supervisor is also an employee, therefore the employee entity

type participates twice in the relationship, once as an employee and once as a supervisor, therefore we can specify two roles, employee and supervisor.

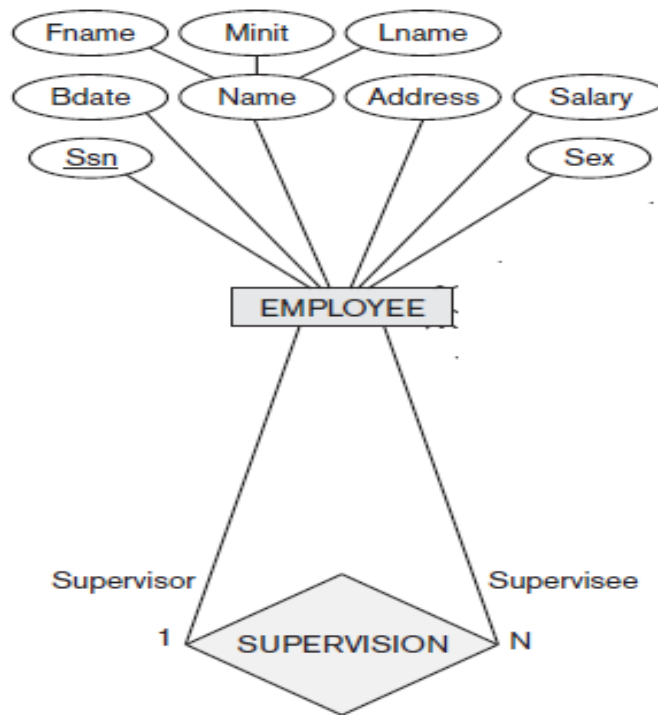


Figure 7.11 A recursive relationship SUPERVISION between EMPLOYEE in the supervisor role (1) and EMPLOYEE in the subordinate role (2).

7.4.3 Constraints on Binary Relationship Types

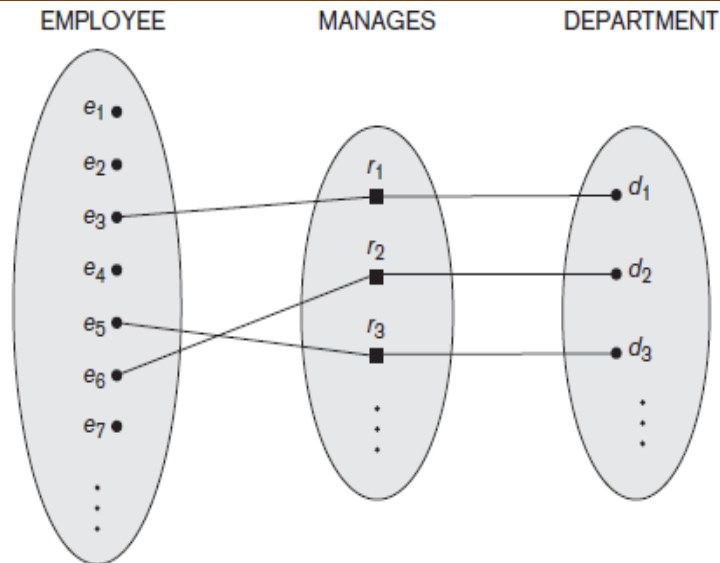
- Relationship types have certain constraints that limit the possible combination of entities that may participate in relationship.
- There are two main types of relationship constraints **cardinality ratio and participation**.

1. Cardinality for Binary Relationship

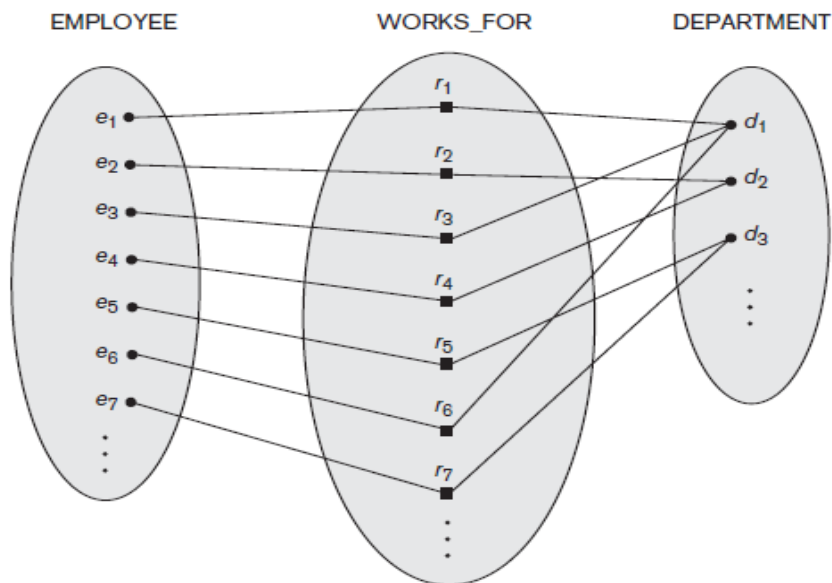
- The cardinality ratio specifies the maximum number of relationship instances that an entity can participate in.
- The possible cardinality ratios for binary relationship types are: **1:1, 1:N, N:1, M:N**.
- One-to-one** – When only one instance of an entity is associated with the relationship, it is marked as '1:1'. An example of a 1:1 binary relationship is MANAGES (Figure 7.12), which relates a department entity to the employee who manages that department.

Figure 7.12

A 1:1 relationship,
MANAGES.

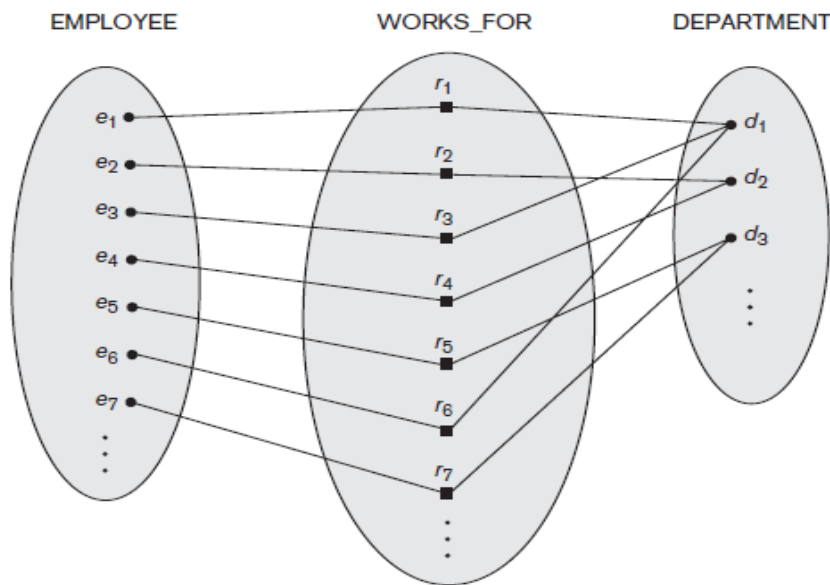


- **One-to-many** – When more than one instance of an entity is associated with a relationship, it is marked as '1:N'. For example, in the WORKS_FOR binary relationship type, **DEPARTMENT:EMPLOYEE** is of cardinality ratio 1:N, meaning that each department can have any number of employees, but an employee work for only one department.



DEPARTMENT:EMPLOYEE 1:N relationship

- **Many-to-one** – When more than one instance of entity is associated with the relationship, it is marked as 'N:1'.
- For example, in the WORKS_FOR binary relationship type, **EMPLOYEE: DEPARTMENT** is of cardinality ratio N:1, meaning that many number of employees work for one department.



EMPLOYEE: DEPARTMENT N:1 relationship

- **Many-to- Many** The relationship type WORKS_ON (Figure 7.13) is of cardinality ratio M:N, an employee can work on several projects and in a project there might be several employees.

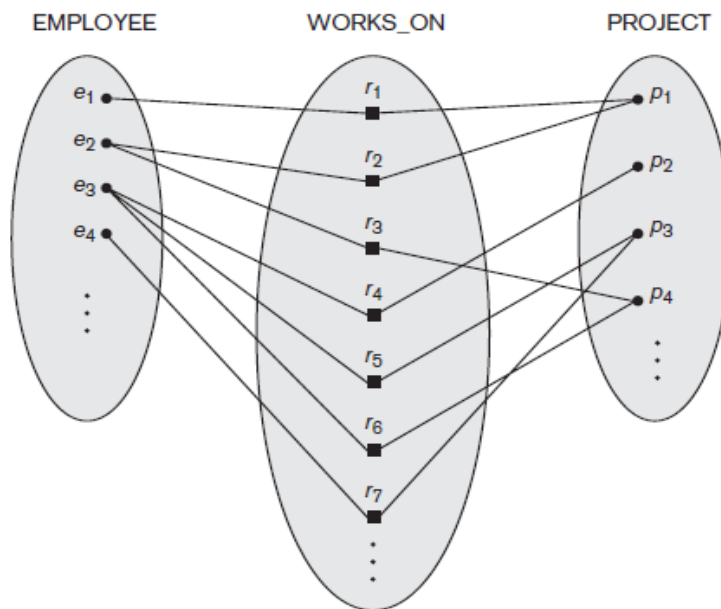


Figure 7.13
An M:N relationship,
WORKS_ON.

2. 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.
- The constraint specifies the minimum number of relationship instances that each entity can participate in.
- There are two types of participation constraints:
 1. **Total Participation.**
 2. **Partial Participation.**

Total Participation.

- Total Participation is when each entity in the entity set occurs in at least one relationship in that relationship set.
- If a company policy states that *every* employee must work for a department, then an employee entity can exist only if it participates in at least one WORKS_FOR relationship instance .
- Thus, the participation of EMPLOYEE in WORKS_FOR is called **total participation**, meaning that every entity in *the total set* of employee entities must be related to a department entity via WORKS_FOR.
- It is also sometimes called an **existence dependency**.
- Total participation is represented by a double line, going from the relationship to the dependent entity.

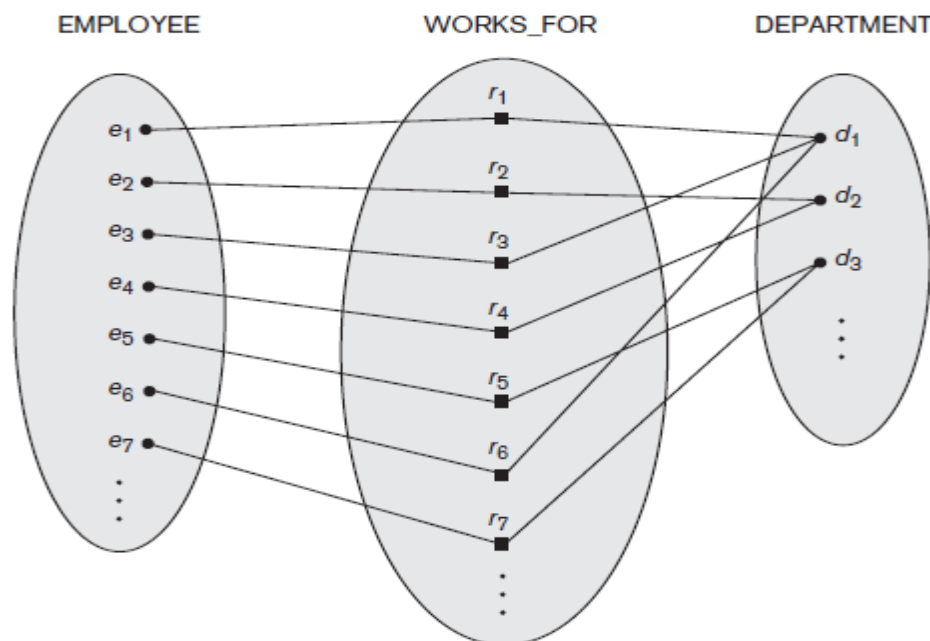


Figure : Total Participation

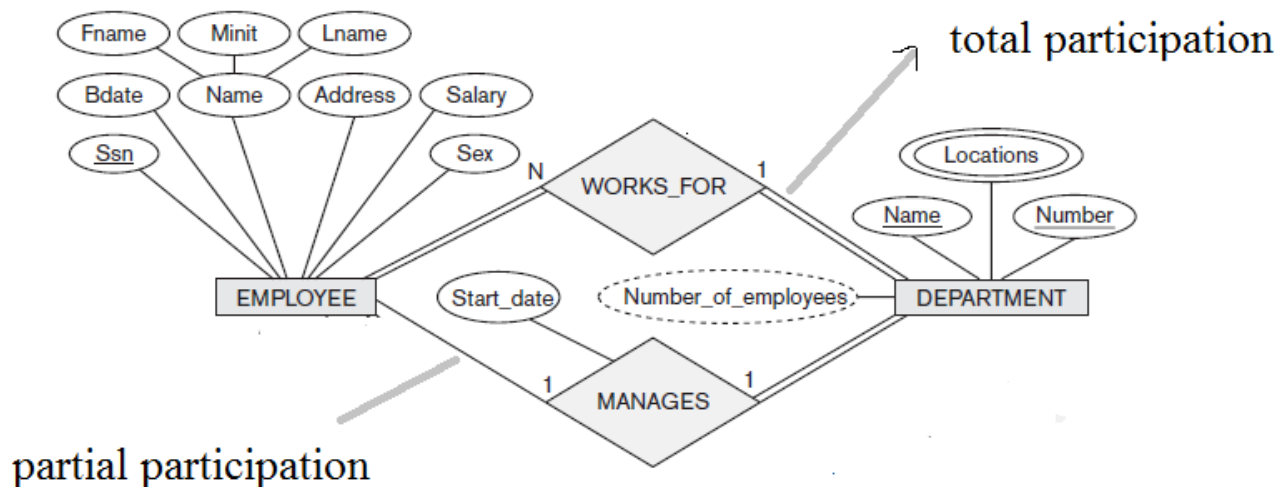


Figure: Representation of total and partial participation in ER diagram

Partial Participation.

- If only a part of the set of entities participate in a relationship, then it is called **partial participation**.
- Using the Company example, every employee will not be a manager of a department, so the participation of an employee in the “Manages” relationship is partial.
- Partial participation is represented by a **single line**.
- In Figure 7.12 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.

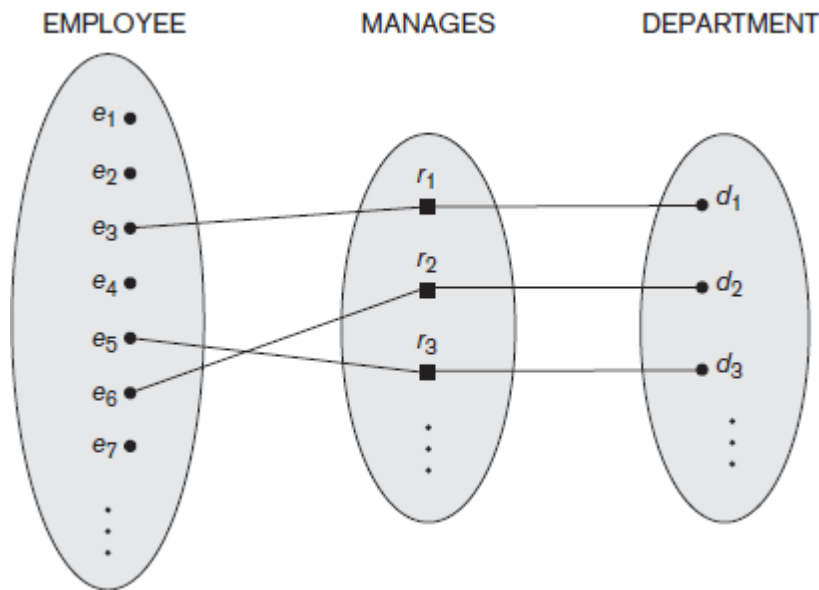


Figure : Partial Participation

7.4.3 Attributes of Relationship Types:

- Relationships can have attributes similar to entity types.
- For example, in the relationship Works_On, between the Employee entity and the Department entity we would like to keep track of the number of hours an employee works on a project. Therefore we can include Number of Hours as an attribute of the relationship.
- Another example is for the “manages” relationship between employee and department, we can add Start Date as an attribute of the Manages relationship.
- For some relationships (1:1, or 1:N), the attribute can be placed on one of the participating entity types. For example the “Manages” relationship is 1:1, StartDate can either be migrated to Employee or Department.

7.5 Weak Entity Types

- Entity types that do not have key attributes are called **weak entity types**.
- Entities that belong to a weak entity type are identified by being related to specific entities from another entity type in combination with one of their attribute values. This entity type is called an **identifying or owner entity type**.
- The relationship that relates the identifying entity type with the weak entity type is called an **identifying relationship**.
- A weak entity type always has a total participation constraint with respect to the identifying relationship, because a weak entity cannot exist without its owner.
- A weak entity type usually has a partial key, which is the set of attributes that can uniquely identify weak entities that are **related to the same owner entity**
- Consider the entity type DEPENDENT, related to EMPLOYEE, the attributes of DEPENDENT are Name, Birth_date, Sex, and Relationship (to the employee). The dependents are identified as distinct entities only after determining the particular employee entity to which each dependent is related.
- A weak entity type normally has a **partial key**, which is the attribute that can uniquely identify weak entities that are related to the same owner entity. For example, the attribute Name of DEPENDENT is 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**. The partial key attribute is underlined with a **dashed or dotted line**.

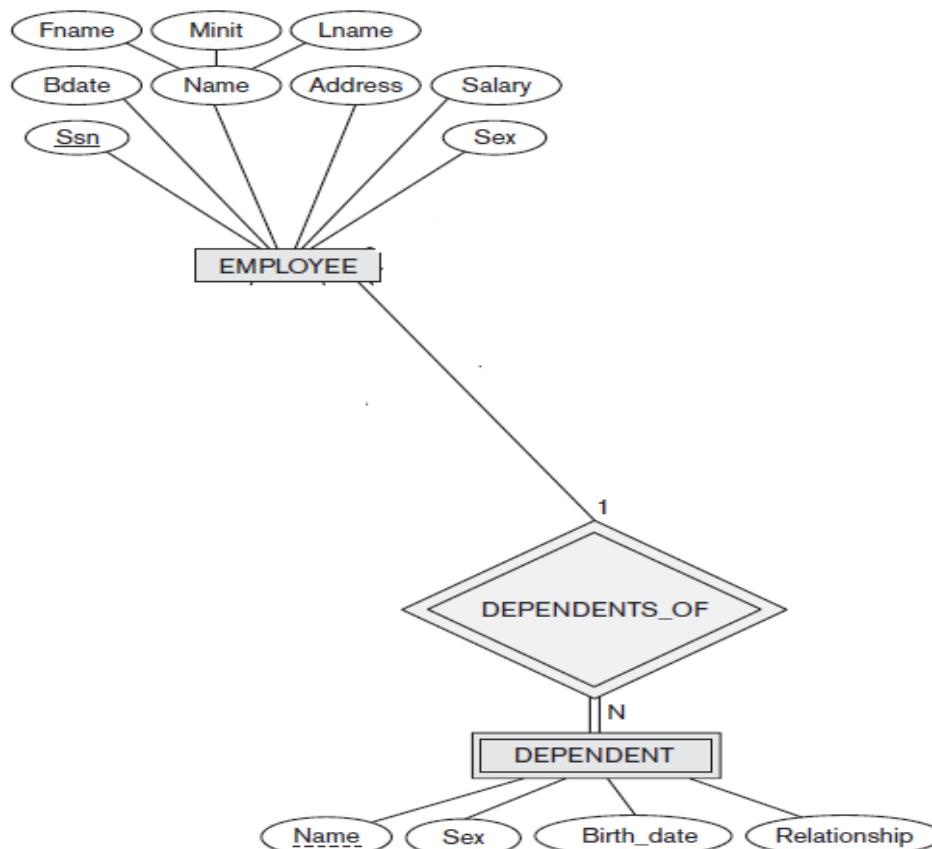


Figure: Weak entity: DEPENDENT, Partial key : Name

7.9 Relationship Types of Degree Higher than Two

- Relationship types of degree 2 are called binary
- Relationship types of degree 3 are called ternary and of degree n are called n -ary .
- An n -ary relationship is not equivalent to n binary relationships
- In general, 3 binary relationships can represent different information than a single ternary relationship
- In Figure 7.17(a), 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 .
- Figure 7.17(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.
- The designer must decide whether a relationship of degree n or should be broken down into smaller degrees based on the semantics or meaning of the particular situation

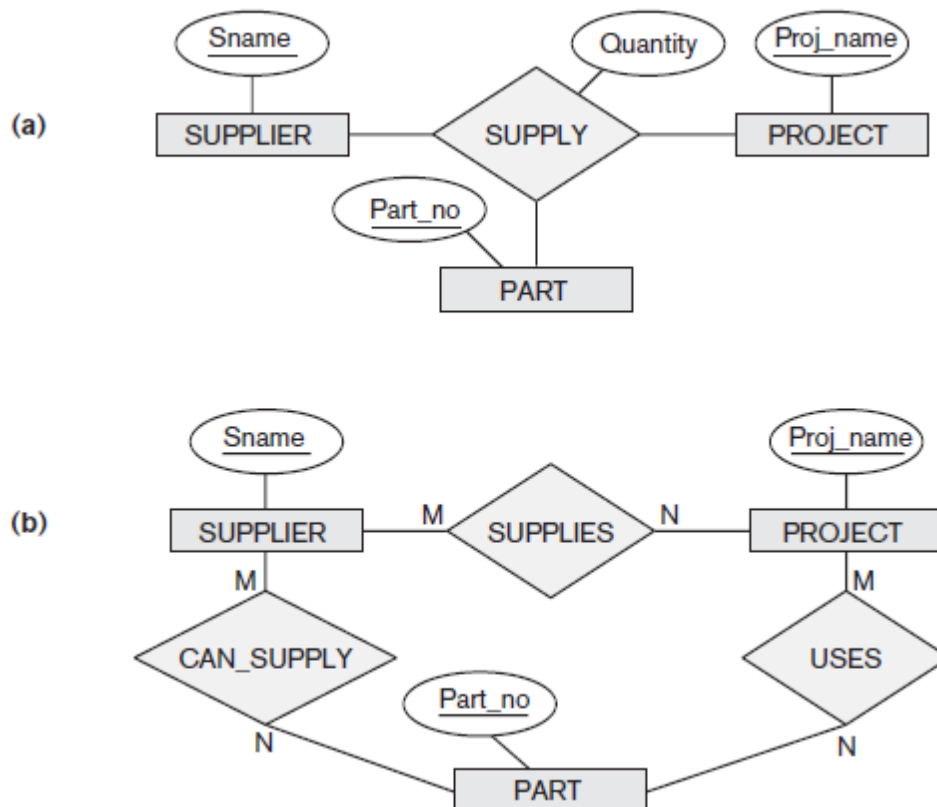


Figure 7.17
Ternary relationship types. (a) The SUPPLY relationship. (b) Three binary relationships not equivalent to SUPPLY.

ER Diagrams, Naming Conventions, and Design Issues.


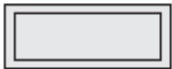






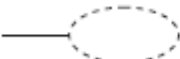
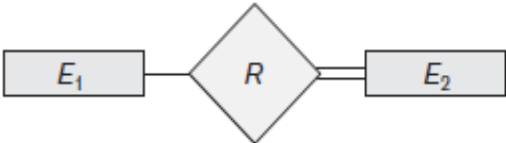
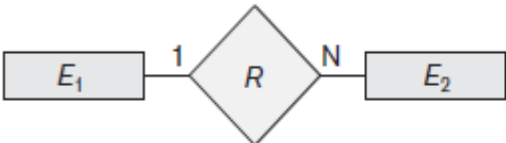
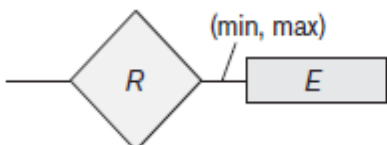
Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Identifying Relationship
	Attribute
	Key Attribute
	Multivalued Attribute
	Composite Attribute
	Derived Attribute
	Total Participation of E_2 in R
	Cardinality Ratio 1 : N for $E_1:E_2$ in R
	Structural Constraint (min, max) on Participation of E in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

Figure 7.14

Summary of the notation for ER diagrams.

A Sample Database Application

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

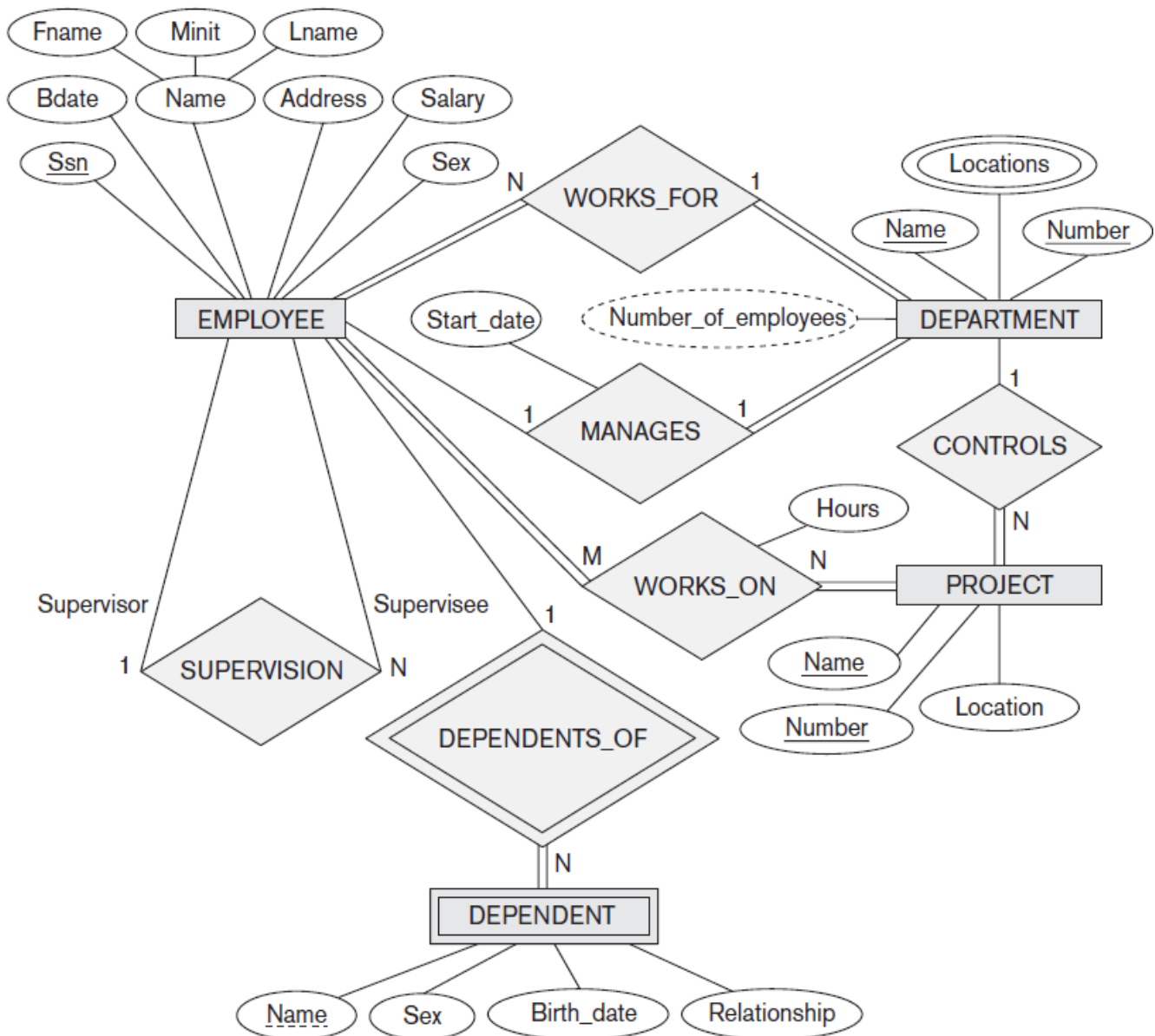


Figure 7.2

An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter and is summarized in Figure 7.14.

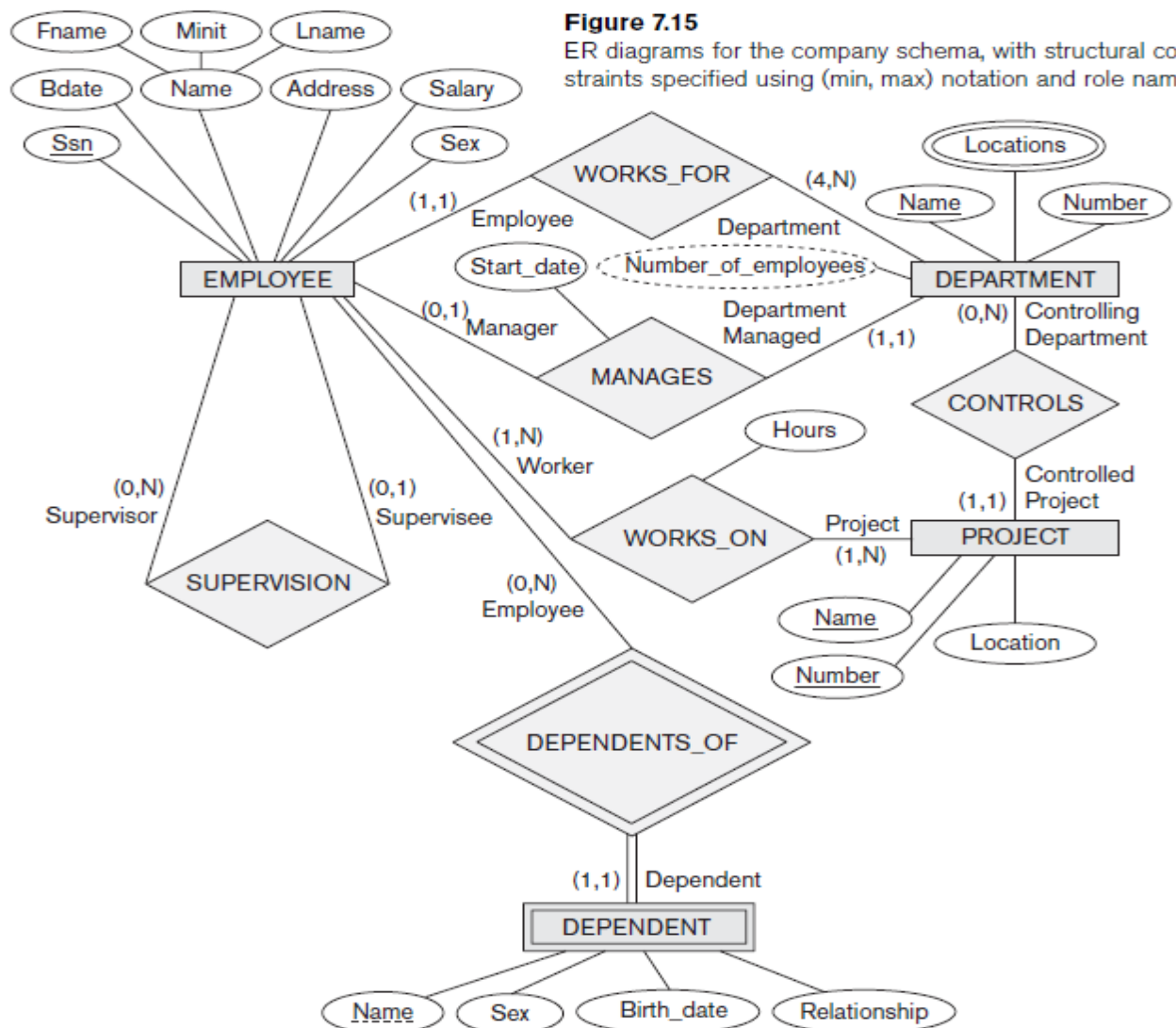
In this example, we specify the following relationship types:

- MANAGES, a 1:1 relationship type between EMPLOYEE and DEPARTMENT. EMPLOYEE participation is partial.
- 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

Refining the ER Design for the COMPANY Database

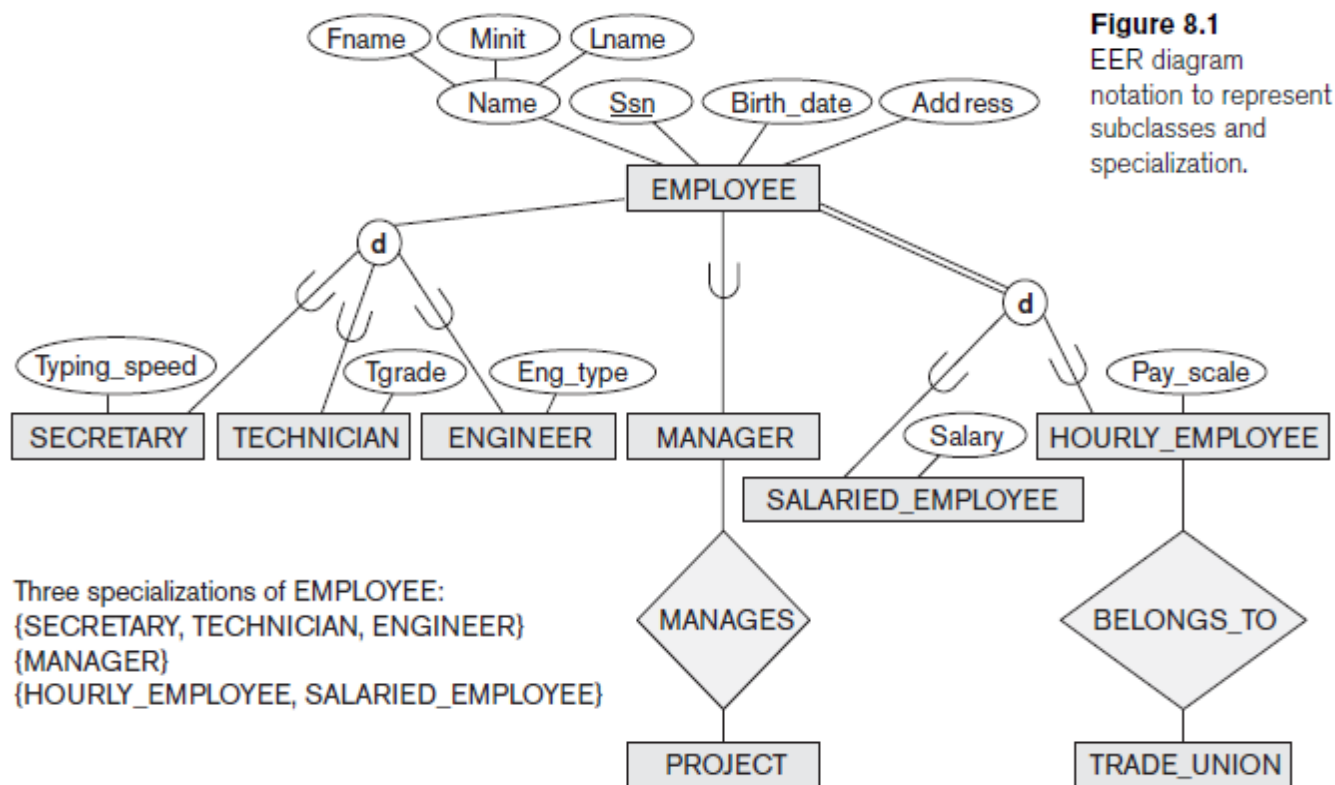
- There is one alternative ER notation for specifying structural constraints on relationships, which replaces the cardinality ratio (1:1, 1:N, M:N) and single/double line notation for participation constraints.
- This notation involves associating a pair of integer numbers (min, max) with each participation 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 7.15 displays the COMPANY database schema using the (min, max) notation.



8.2 Specialization and Generalization

8.2.1 Specialization

- **Specialization** is the process of defining a set of subclasses of an entity type; this entity type is called the **superclass** of the specialization.
- The set of subclasses that forms a specialization is defined on the basis of some distinguishing characteristic of the entities in the superclass
- For example, the set of subclasses {SECRETARY, ENGINEER, TECHNICIAN} is a specialization of the superclass EMPLOYEE that distinguishes among employee entities based on the job type of each employee entity.
- Figure 8.1 Represents a specialization diagrammatically in an EER diagram. The subclasses that define a specialization are attached by lines to a **circle** that represents the specialization, which is connected in turn to the superclass. The subset symbol on each line connecting a subclass to the circle indicates the direction of the superclass/subclass relationship.



- Attributes that apply only to entities of a particular subclass such as TypingSpeed of SECRETARY are attached to the rectangle representing that subclass. These are called **specific attributes** of the subclass.
- Figure 8.2 shows entity instances that belong to subclasses of the {SECRETARY, ENGINEER, TECHNICIAN} specialization.

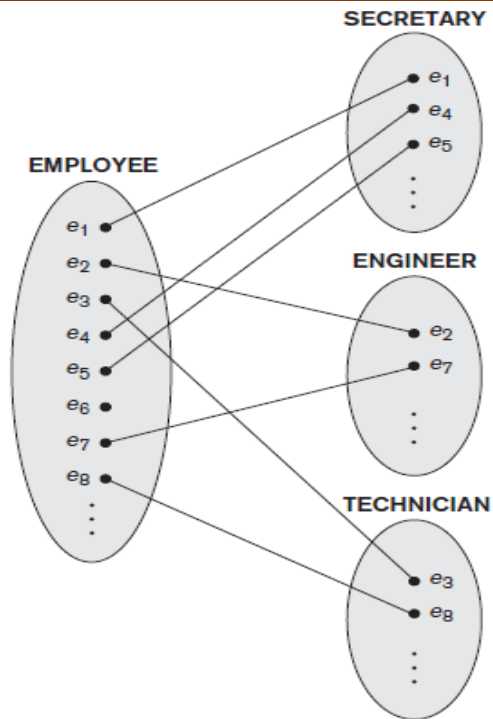


Figure 8.2
Instances of a specialization.

- There are two main reasons for including class/subclass relationships and specializations in a data model.
- The first is that certain attributes may apply to some but not all entities of the superclass. For example, in Figure 8.1 the SECRETARY subclass has the specific attribute Typing_speed, whereas the ENGINEER subclass has the specific attribute Eng_type, but SECRETARY and ENGINEER share their other inherited attributes from the EMPLOYEE entity type.
- The second reason for using subclasses is that some relationship types may be participated in only by entities that are members of the subclass. For example, if only HOURLY_EMPLOYEES can belong to a trade union, we can represent that fact by creating the subclass HOURLY_EMPLOYEE of EMPLOYEE and relating the subclass to an entity type TRADE_UNION via the BELONGS_TO relationship type,
- In summary, the specialization process allows us to do the following:
 - Define a set of subclasses of an entity type
 - Establish additional specific attributes with each subclass
 - Establish additional specific relationship types between each subclass and other entity types or other subclasses

8.2.2 Generalization

- The reverse process of abstraction suppresses the differences among entity types, identify their common features, and **generalize** them into a single **superclass** of which the original entity types are special **subclasses**.

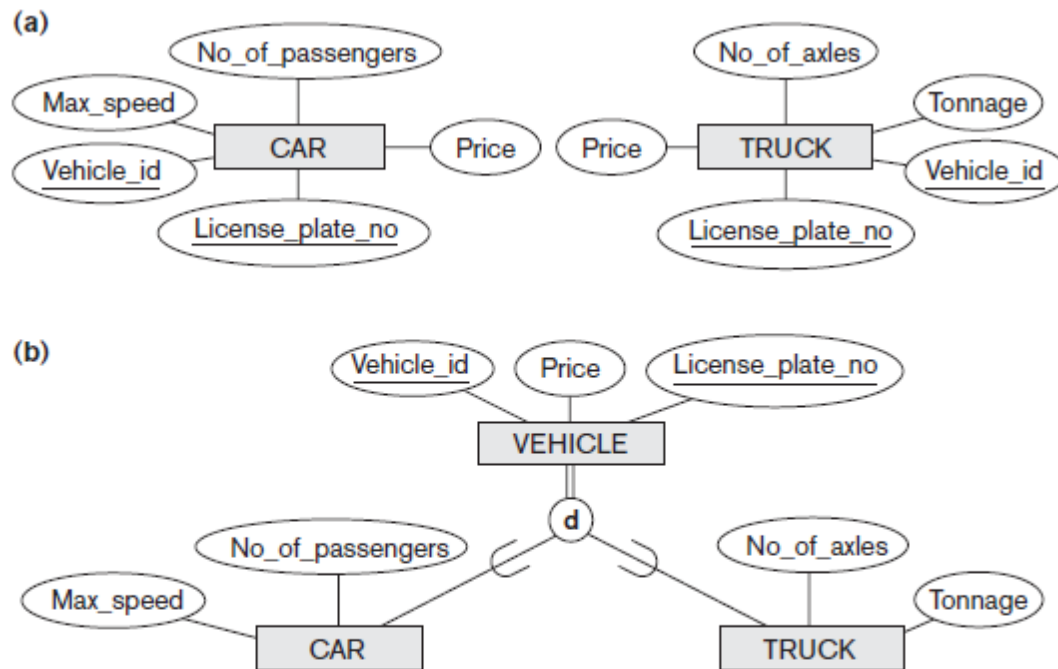
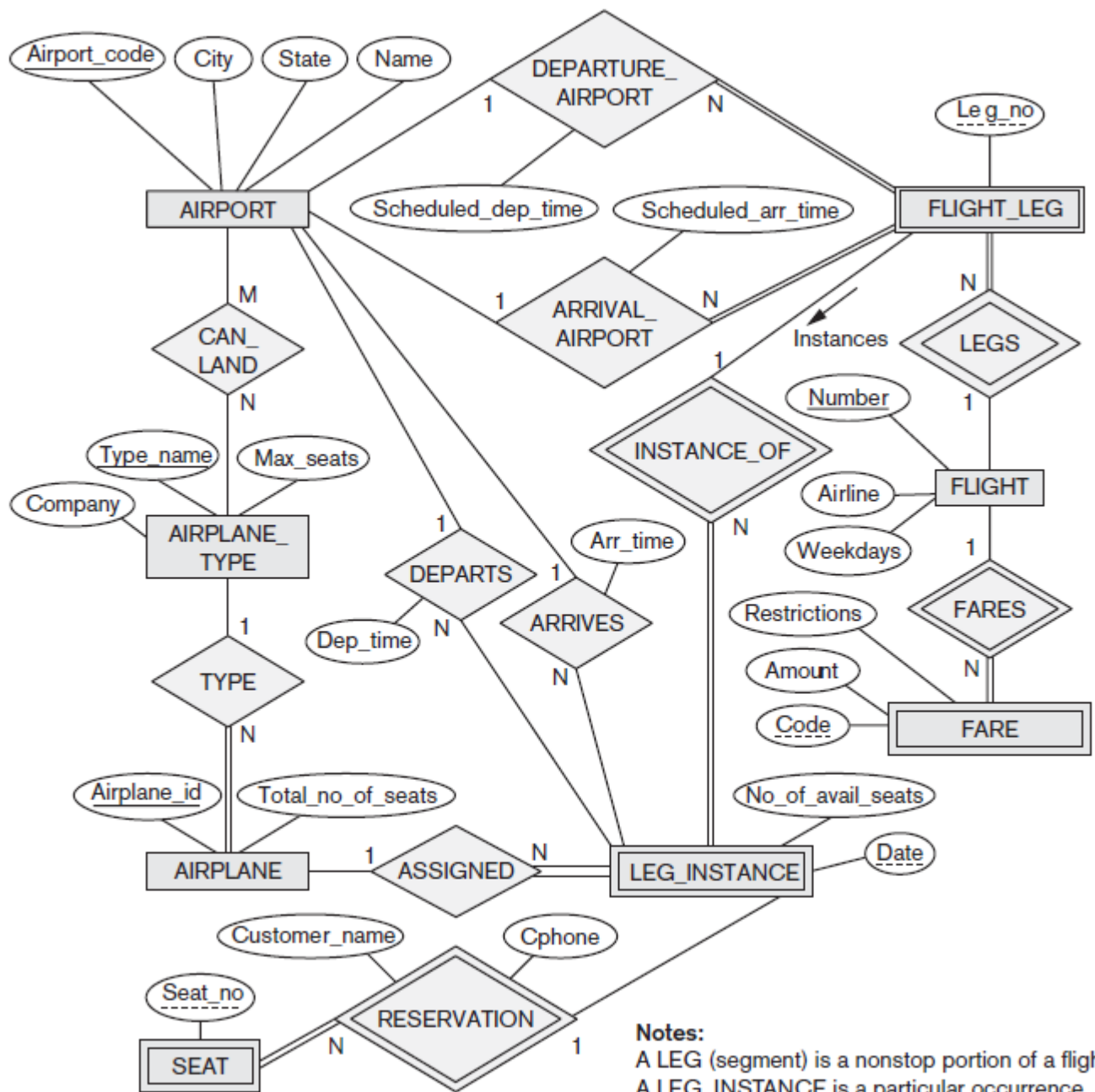


Figure 8.3

Generalization. (a) Two entity types, CAR and TRUCK. (b) Generalizing CAR and TRUCK into the superclass VEHICLE.

- For example, consider the entity types CAR and TRUCK. Because they have several common attributes, they can be generalized into the entity type VEHICLE, as shown in Figure 8.3(b). Both CAR and TRUCK are now subclasses of the **generalized superclass** VEHICLE.
- An arrow pointing to the generalized superclass represents a generalization, whereas arrows pointing to the specialized subclasses represent a specialization.

An ER diagram for an AIRLINE database schema.



An ER diagram for an university database schema.

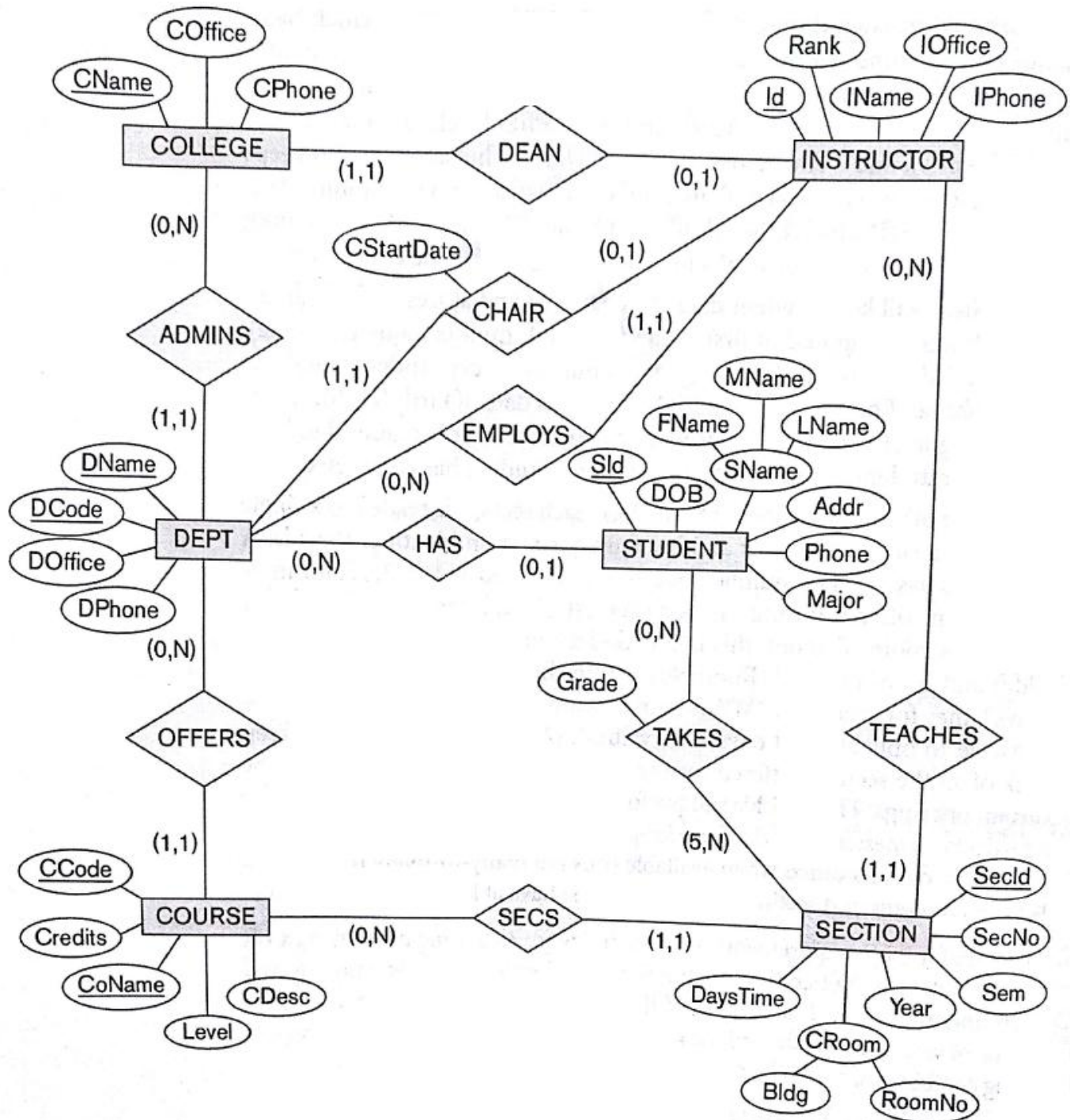


Figure 3.20
An ER diagram for a UNIVERSITY database schema.