UNIT-1

Overview of Database

What is Data?

Data is a raw and unorganized fact that required to be processed to make it meaningful. Generally, data comprises facts, observations, perceptions numbers, characters, symbols, image, etc.

What is Information?

Data is always interpreted, by a human or machine, to derive meaning. Information assigns meaning and improves the reliability of the data. It helps to ensure undesirability and reduces uncertainty. So, when the data is transformed into information, it never has any useless details.

Example: Look at the examples given for data:

- 4,8,12,16
- Dog, cat, cow, cockatoo
- 161.2, 175.3, 166.4, 164.7, 169.3

When we assign a context or meaning to the data, only then the data become information. It all becomes meaningful when you are told:

- 4, 8, 12 and 16 are the first four answers in the 4 x table
- Dog, cat, cow is a list of household pets
- 165, 175.2, 186.3, 164.3, 169.3 are the height of 14-year old students.

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, names, telephone numbers, and addresses of the people.

A **database management system** (DBMS) is a collection of programs that enables users to create and maintain a database. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient.
The DBMS is hence 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 for the data to be stored in the database.
- **Constructing** the database is the process of storing the data itself on some storage medium that is controlled by the DBMS.
- **Manipulating** a database includes such functions 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 concurrently.

Other important functions provided by the DBMS include *protecting* the database and *maintaining* it over a long period of time.

- **Protection** includes both *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.

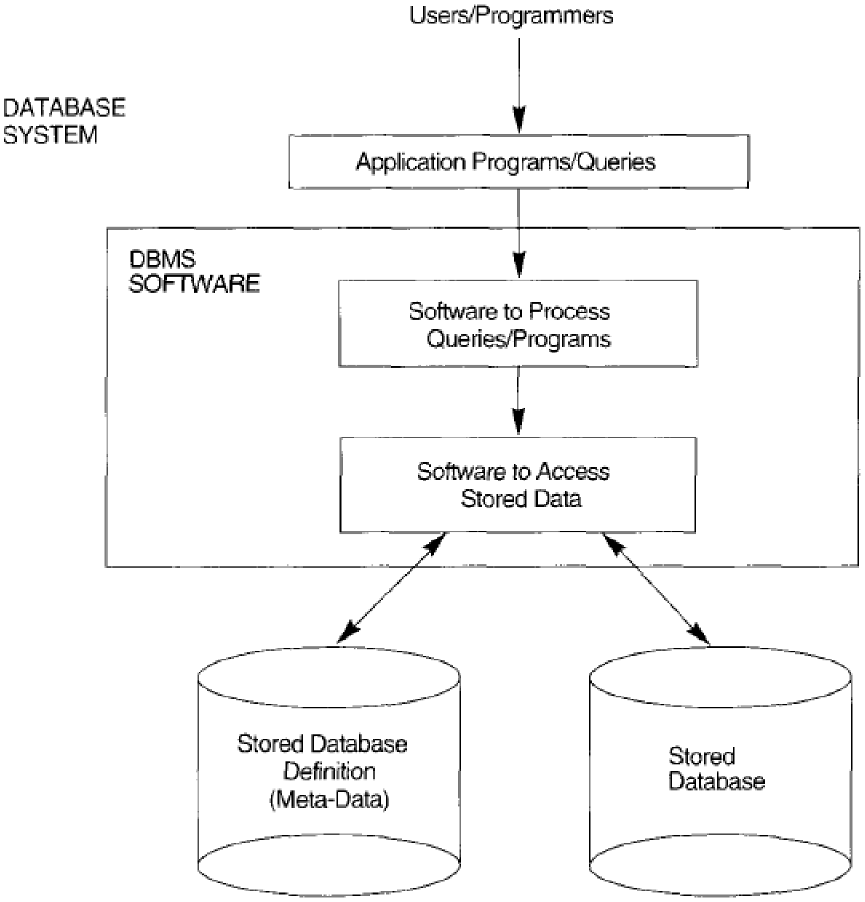

Users/Programmers
DATABASE SYSTEM
Application Programs/Queries
DBMS SOFTWARE
Software to Process Queries/Programs
Software to Access Stored Data
Stored Database Definition (Meta-Data)
Stored Database

FIGURE 1.1 A simplified database system environment.

Database System Applications

Databases are widely used. Here are some representative applications:

- **Banking:** For customer information, accounts, and loans, and banking transactions.
- **Airlines:** For reservations and schedule information.
- **Universities:** For student information, course registrations, and grades.
- **Credit card transactions:** For purchases on credit cards and generation of monthly statements.
- **Telecommunication:** For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.
- **Finance:** For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds.
- **Sales:** For customer, product, and purchase information.
- **Manufacturing:** For management of supply chain and for tracking production of items in factories, inventories of items in warehouses/stores, and orders for items.
- **Human resources:** For information about employees, salaries, payroll taxes and benefits, and for generation of paychecks.

Advantages of DBMS

1. Improved data sharing

An advantage of the database management approach is, the DBMS helps to create an environment in which end users have better access to more and better-managed data. Such access makes it possible for end users to respond quickly to changes in their environment.

2. Improved data security

The more users access the data, the greater the risks of data security breaches. A DBMS provides a framework for better enforcement of data privacy and security policies.

3. Better data integration

Wider access to well-managed data promotes an integrated view of the organization's operations and a clearer view of the big picture. It becomes much easier to see how actions in one segment of the company affect other segments.

4. Minimized data inconsistency

Data inconsistency exists when different versions of the same data appear in different places. All data appears consistently across the database and the data is same for all the users viewing the database. Moreover, any changes made to the database are immediately reflected to all the users and there is no data inconsistency.

5. Improved data access

The DBMS makes it possible to produce quick answers to ad hoc queries. From a database perspective, a query is a specific request issued to the DBMS for data manipulation. The DBMS sends back an answer (called the query result set) to the application.

6. Improved decision making

Better-managed data and improved data access make it possible to generate better-quality information, on which better decisions are based. While the DBMS does not guarantee data quality, it provides a framework to facilitate data quality initiatives.

7. Increased end-user productivity

The availability of data, combined with the tools that transform data into usable information, empowers end users to make quick, informed decisions that can make the difference between success and failure in the global economy.

Disadvantages of DBMS

1. Increased costs

One of the disadvantages of DBMS is Database systems require sophisticated hardware and software and highly skilled personnel. The cost of maintaining the hardware, software, and personnel required to operate and manage a database system can be substantial.

2. Management complexity

Database systems interface with many different technologies and have a significant impact on a company's resources and culture. The changes introduced by the adoption of a database system must be properly managed to ensure that they help advance the company's objectives.

3. Maintaining currency

To maximize the efficiency of the database system, you must keep your system current. Therefore, you must perform frequent updates and apply the latest patches and security measures to all components.

Because database technology advances rapidly, personnel training costs tend to be significant.

4. Frequent upgrade/replacement cycles

DBMS vendors frequently upgrade their products by adding new functionality. Such new features often come bundled in new upgrade versions of the software. Some of these versions require hardware upgrades. Not only do the upgrades themselves cost money, but it also costs money to train database users and administrators to properly use and manage the new features.

Database Systems versus File Systems

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 a file 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-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 data up to date.
Keeping organizational information in a file-processing system has a number of major disadvantages:

- **Data redundancy and inconsistency:** Since different programmers create the files and application programs the same information may be duplicated in several places (files). In addition, it may lead to data inconsistency; that is, the various copies of the same data may no longer agree.
- **Difficulty in accessing data**
- **Data isolation:** Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.
- **Integrity problems:** The data values stored in the database must satisfy certain types of consistency constraints.
- **Atomicity problems:** All the updations must be done in its entirety or not at all.
- **Concurrent-access anomalies:** For the sake of overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously. In such an environment, interaction of concurrent updates may result in inconsistent data.
- **Security problems:** Not every user of the database system should be able to access all the data.

Difference between File System and DBMS:

S.NO.	FILE SYSTEM	DBMS
1.	File system is a software that manages ad organizes the files in a storage medium within a computer.	DBMS is a software for managing the database.
2.	Redundant data can be present in a file system.	In DBMS there is no redundant data.
3.	It doesn't provide backup and recovery of data if it is lost.	It provides backup and recovery of data even if it is lost.
4.	There is no efficient query processing in file system.	Efficient query processing is there in DBMS.
5.	There is less data consistency in file system.	There is more data consistency because of the process of normalization.
6.	It is less complex as compared to DBMS.	It has more complexity in handling as compared to file system.
7.	File systems provide less security in comparison to DBMS.	DBMS has more security mechanisms as compared to file system.
8.	It is less expensive than DBMS.	It has a comparatively higher cost than a file system.

CHARACTERISTICS OF THE DATABASE APPROACH

In the database approach, a single repository of data is maintained that is defined once and then is accessed by various users. The main characteristics of the database approach versus the file-processing approach are the following:

• **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. The information stored in the catalog is called meta-data, and it describes the structure of the primary database.

• Insulation between programs and data, and data abstraction
In traditional file processing, the structure of data files is embedded in the application programs, so any changes to the structure of a file may require changing all programs that access this file. By contrast, DBMS access programs do not require such changes. The structure of data files is stored in the DBMS catalog separately from the access programs. This is called **program-data independence**. Users can define operations on data as part of the database definitions. This is called **program-operation independence.** The characteristic that allows program-data independence and program-operation independence is called **data abstraction**.

• **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.

• Sharing of data and multiuser transaction processing
A multiuser DBMS, must allow multiple users to access the database at the same time.

Data Abstraction

For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database. Developers hide the complexity from users through several levels of abstraction, to simplify users' interactions with the system:

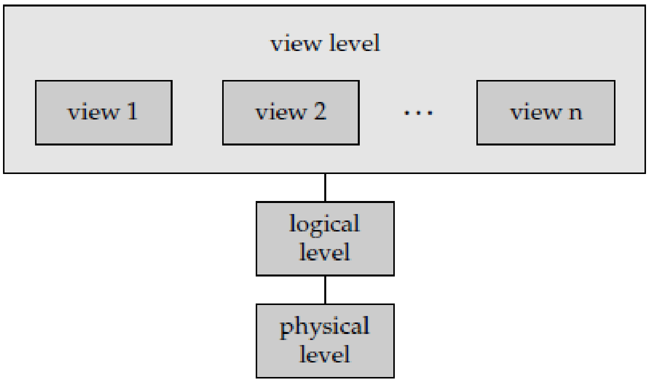


Figure 1.1 The three levels of data abstraction.

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

• **Logical level.** The next-higher level of abstraction describes what data are stored in the database, and what relationships exist among those data. The logical level thus describes the entire database in terms of a small number of relatively simple structures

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

Database system concept and architecture

DATA MODELS

A data model is a collection of concepts that can be used to describe the structure of a database which provides the necessary means to achieve this abstraction. Structure of a database includes data types, relationships, and constraints that should hold for the data. Most data models also include a set of basic operations for specifying retrievals and updates on the database.

Categories of Data Models

Categories of Data Models are:

- **High-level or conceptual data models:** It provides concepts that are close to the way many users perceive data. 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, 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 two or more entities, for example, a works-on relationship between an employee and a project.

- **Low-level or physical data models:** It provides concepts that describe the details of how data is stored in the computer.

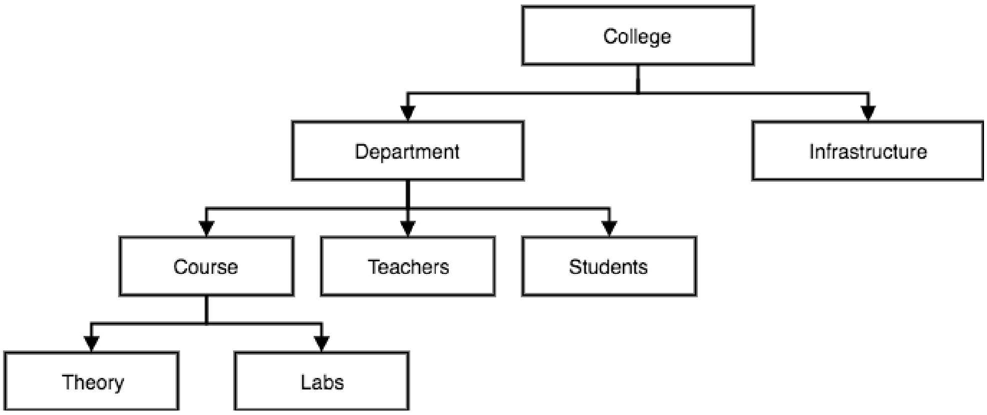
- **Representational (or implementation) data models:** It provides concepts that may be understood by end users but that are not too far removed from the way data is organized within the computer. Representational data models hide some details of data storage but can be implemented on a computer system in a direct way.

Types of Data Models

1. Hierarchical Data Model

This database model organises data into a tree-like-structure, with a single root, to which all the other data is linked. The hierarchy starts from the Root data, and expands like a tree, adding child nodes to the parent nodes. In this model, a child node will only have a single parent node.

In hierarchical model, data is organised into tree-like structure with one one-to-many relationship between two different types of data, for example, one department can have many courses, many professors and of-course many students.

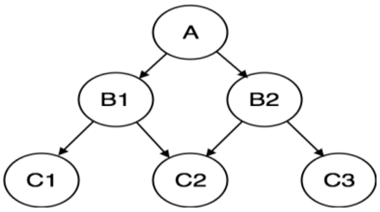


The main advantage of hierarchical data model is that data access is quite predictable in structure and retrieval & updates can be highly optimized by a DBMS. The disadvantage is that the link is permanently established and cannot be modified which makes this model rigid.

2. Network Data Model

This is an extension of the Hierarchical model. In this model data is organised more like a graph, and are allowed to have more than one parent node.

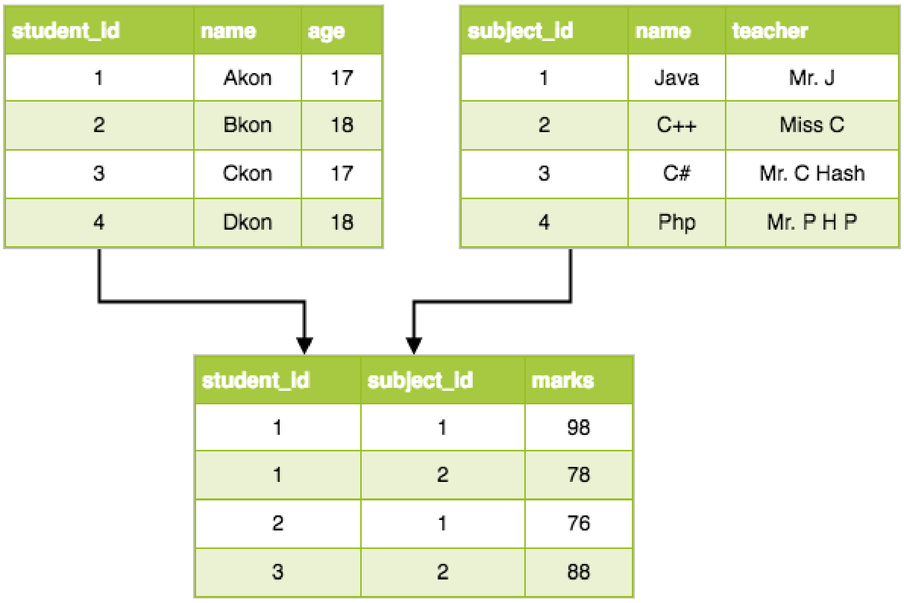
In this database model data is more related as more relationships are established in this database model. Also, as the data is more related, hence accessing the data is also easier and fast. This database model was used to map many-to-many data relationships.



Supporting multiple paths in the data structure eliminates some of the drawbacks of the hierarchical model; the network model is not very practical. The primary drawback of networked databases is that it can be quite complicating to maintain all the links.

3. Relational Data Model

The key differences between previous database models and relational database model is in terms of flexibility. A relational database model represents all data in the database as simple two-dimensional tables called relations. All the information related to a particular type is stored in rows of that table with columns representing attributes (fields). The allowable values of these attributes are called the domain.



student_id	name	age
1	Akon	17
2	Bkon	18
3	Ckon	17
4	Dkon	18

subject_id	name	teacher
1	Java	Mr. J
2	C++	Miss C
3	C#	Mr. C Hash
4	Php	Mr. P H P

student_id	subject_id	marks
1	1	98
1	2	78
2	1	76
3	2	88

4. Object Oriented Data Model

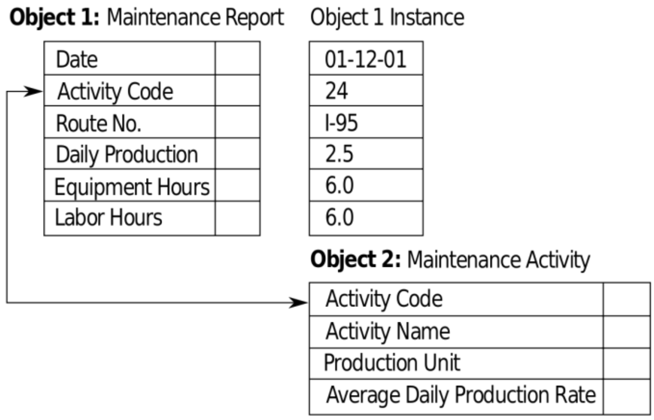
This model defines a database as a collection of objects, or reusable software elements, with associated features and methods. There are several kinds of object-oriented databases:

A **multimedia database** incorporates media, such as images, that could not be stored in a relational database.

A **hypertext database** allows any object to link to any other object. It’s useful for organizing lots of disparate data, but it’s not ideal for numerical analysis.

The object-oriented database model is the best known post-relational database model, since it incorporates tables, but isn’t limited to tables. Such models are also known as hybrid database models.

Object-Oriented Model



Object 1: Maintenance Report

Date	
Activity Code	
Route No.	
Daily Production	
Equipment Hours	
Labor Hours	

Object 1 Instance

01-12-01
24
I-95
2.5
6.0
6.0

Object 2: Maintenance Activity

Activity Code	
Activity Name	
Production Unit	
Average Daily Production Rate	

Schemas, Instances, and Database State

Schema: 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. Most data models have certain conventions for displaying schemas as diagrams. A displayed schema is called a schema diagram.

Instance: 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.
We get the **initial state** of the database when the database is first populated or loaded with the initial data. The DBMS is 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 Three-Schema Architecture

The goal of the three-schema architecture is to separate the user applications and 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. 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.

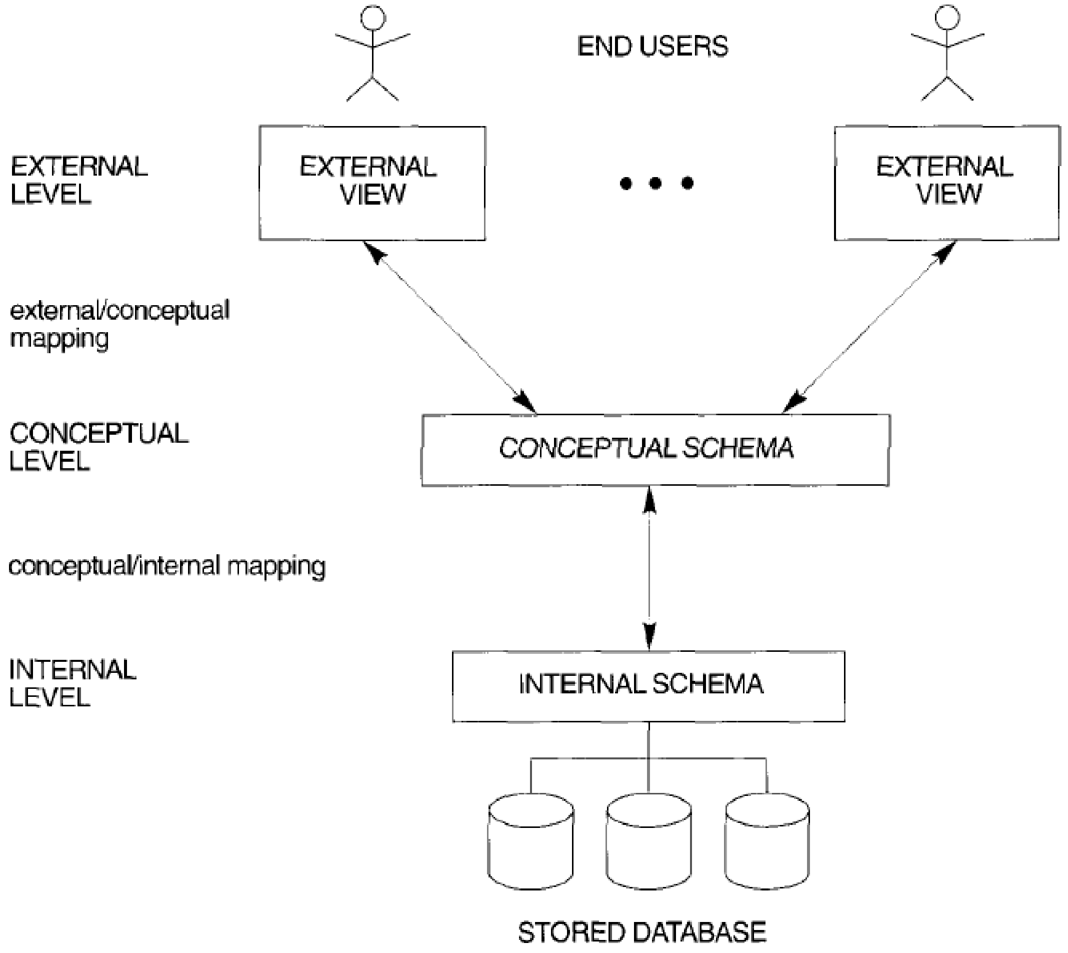


FIGURE 2.2 The three-schema architecture.

The three schemas are only descriptions of data; the only data that actually exists is at the physical level. In a DBMS based on the three-schema architecture, each user group refers only 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. The processes of transforming requests and results between levels are called **mappings**.

Data Independence

Data independence can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level. We can define two types of data independence:

1. **Logical data independence**
   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). External schemas that refer only to the remaining data should not be affected. Changes to constraints can be applied to the conceptual schema without affecting the external schemas or application programs.

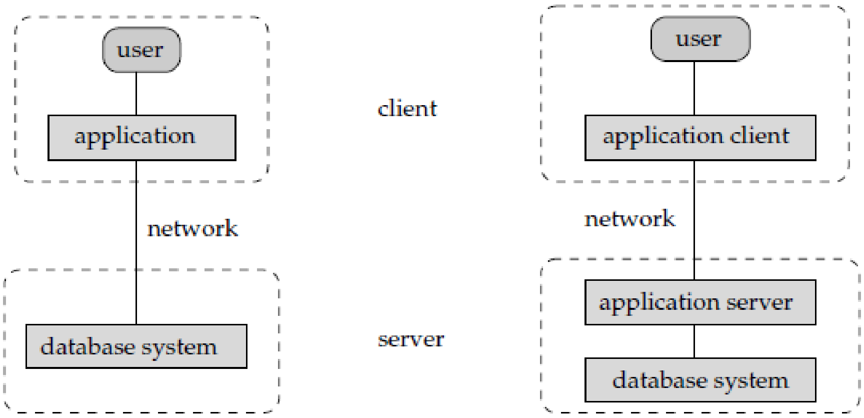
2. **Physical data independence**

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 had to be reorganized.

Data independence occurs because when the schema is changed at some level, the schema at the next higher level remains unchanged; only the mapping between the two levels is changed. Hence, application programs referring to the higher-level schema need not be changed.

Application Architecture of DBMS

Basically, there are two types of application architecture of DBMS:
a) Two-Tier Architecture
b) Three-Tier Architecture



a. two-tier architecture b. three-tier architecture

a) In a **two-tier architecture**, the application is partitioned into a component that resides at the client machine, which invokes database system functionality at the server machine through query language statements. Application program interface standards like ODBC and JDBC are used for interaction between the client and the server.

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

Database language and interfaces

Database Languages:

A database system provides a **data definition language** to specify the database schema and a **data manipulation language** to express database queries and updates.

1. **Data-Definition Language**
   We specify a database schema by a set of definitions expressed by a special language called a **data-definition language (DDL).**
   For instance, the following statement in the SQL language defines the account table:

```
create table account
(account-number char(10),
balance integer)
```

Execution of the above DDL statement creates the account table. it updates a special set of tables called the **data dictionary or data directory**.

A data dictionary contains **metadata**. A database system consults the data dictionary before reading or modifying actual data.

The storage structure and access methods used by the database system by a set of statements in a special type of DDL called a data **storage and definition language**. These statements define the implementation details of the database schemas, which are usually hidden from the users.

Example: Create, alter, drop.

2. **Data-Manipulation Language**

   Data manipulation is
   - The retrieval of information stored in the database
   - The insertion of new information into the database
   - The deletion of information from the database
   - The modification of information stored in the database

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

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

A **query** is a statement requesting the retrieval of information. The portion of a DML that involves information retrieval is called a **query language**.

This query in the SQL language finds the name of the customer whose customer-id is 192-83-7465:

```
select customer.customer-name
from customer
where customer.customer-id = 192-83-7465
```

3. **Data Control Language**
   It is the component of SQL statement that control access to data and to the database.
   Example: Commit, Savepoint, Rollback, etc.

4. **Data Query Language**
   It is the component of SQL statement that allows getting data from the database and imposing ordering upon it.
   Example: Select.

DBMS Interfaces:

User-friendly interfaces provided by a DBMS include the following:

1. **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.

2. **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 fill out only certain entries, in which case the DBMS will retrieve matching data for the remaining entries. Many DBMSs have forms **specification languages**, which are special languages that help programmers specify such forms.

3. **Graphical User Interfaces**
   A graphical interface (CUI) typically displays a schema to the user in diagrammatic form. The user can then specify a query by manipulating the diagram. In many cases, CUIs utilize both menus and forms.

4. **Natural Language Interfaces**
   These interfaces accept requests written in English or some other language and attempt to "understand" them. A natural language interface usually has its own "schema," which is similar to the database conceptual schema, as well as a dictionary of important words. The natural language interface refers to the words in its schema, as well as to the set of standard words in its dictionary, to interpret the request. If the interpretation is successful, the interface generates a high-level query corresponding to the natural language request and submits it to the DBMS for processing; otherwise, a dialogue is started with the user to clarify the request.

5. **Interfaces for Parametric Users**
   Parametric users, such as bank tellers, often have a small set of operations that they must perform repeatedly. 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.

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

Database Users

In large organizations, many people are involved in the design, use, and maintenance of a large database with hundreds of users. We identify the people whose jobs involve the day-to-day use of a large database; we call them the actors on the scene.

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, and acquiring software and hardware resources as needed. The DBA is accountable for problems such as security breaches and poor system response time.

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. These tasks are mostly undertaken before the database is actually implemented and populated with data. It is the responsibility of database designers to communicate with all prospective database users in order to understand their requirements and to create a design that meets these requirements.

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.

- **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:

    - 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.

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.

Overall Database Structure

A database system is partitioned into modules that deal with each of the responsibilities of the overall system. The functional components of a database system can be broadly divided into the storage manager and the query processor components.

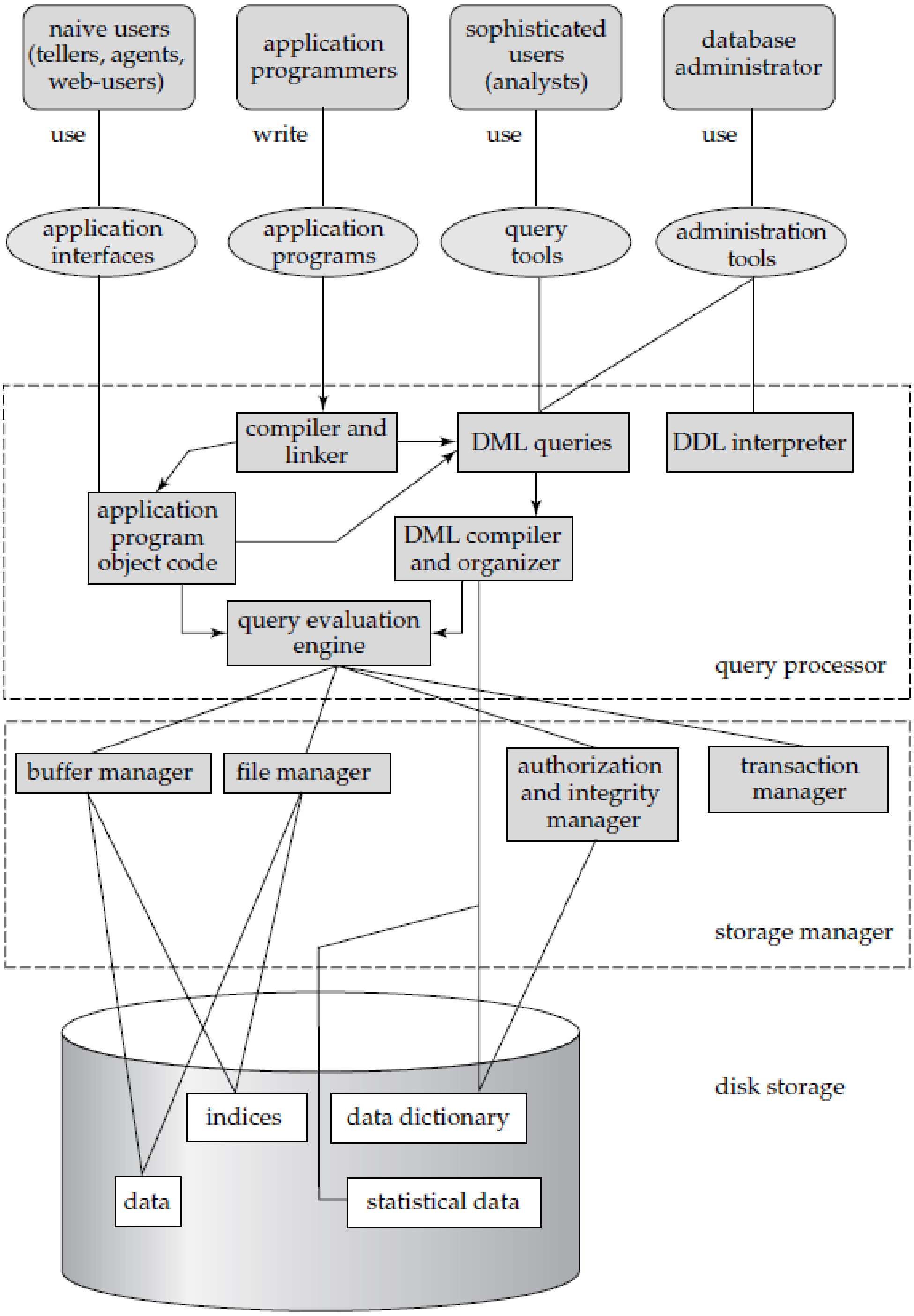
1. Storage Manager

A storage manager is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system. The storage manager is responsible for the interaction with the file manager. The storage manager translates the various DML statements into low-level file-system commands. Thus, the storage manager is responsible for storing, retrieving, and updating data in the database. The storage manager components include:

- **Authorization and integrity manager**, which tests for the satisfaction of integrity constraints and checks the authority of users to access data.
- **Transaction manager**, which ensures that the database remains in a consistent (correct) state despite system failures, and that concurrent transaction executions proceed without conflicting.
- **File manager**, which manages the allocation of space on disk storage and the data structures used to represent information stored on disk.
- Buffer manager, which is responsible for fetching data from disk storage into main memory, and deciding what data to cache in main memory.

The storage manager implements several data structures as part of the physical system implementation:

- Data files, which store the database itself.
- Data dictionary, which stores metadata about the structure of the database, in particular the schema of the database.
- Indices, which provide fast access to data items that hold particular values.



2. The Query Processor

The query processor components include:

- **DDL interpreter**, which interprets DDL statements and records the definitions in the data dictionary.
- **DML compiler**, which translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands. A query can usually be translated into any of a number of alternative evaluation plans that all give the same result. The DML compiler also performs query optimization, that is, it picks the lowest cost evaluation plan from among the alternatives.
- **Query evaluation engine,** which executes low-level instructions generated by the DML compiler.

Data Modeling using the Entity Relationship Model

The **entity-relationship** (E-R) data model perceives the real world as consisting of basic objects, called entities, and relationships among these objects. It was developed to facilitate database design by allowing specification of an enterprise schema, which represents the overall logical structure of a database. The E-R model is very useful in mapping the meanings and interactions of real-world enterprises onto a conceptual schema.

The E-R data model employs three basic notions: entity sets, relationship sets, and attributes.

Entity Sets: An **entity** is a "thing" or "object" in the real world that is distinguishable from all other objects. An entity has a set of properties, and the values for some set of properties may uniquely identify an entity. For instance, a person may have a person-id property whose value uniquely identifies that person.
An **entity set** is a set of entities of the same type that share the same properties, or attributes.
For example, the entity set customer.

Attributes: An entity is represented by a set of **attributes**. Attributes are descriptive properties possessed by each member of an entity set.
For example, attributes of the customer entity set are customer-id, customer-name, customerstreet, and customer-city.
Each entity has a **value** for each of its attributes. For each attribute, there is a set of permitted values, called the domain, or **value set**, of that attribute.

An attribute can be characterized by the following attribute types:

- **Simple and composite attributes**
  Simple attributes are the attributes which are not divided into subparts such as "age".
  Composite attributes are the attributes which can be divided into subparts (that is, other attributes). For example, an attribute name could be structured as a composite attribute consisting of first-name, middle-initial, and last-name.

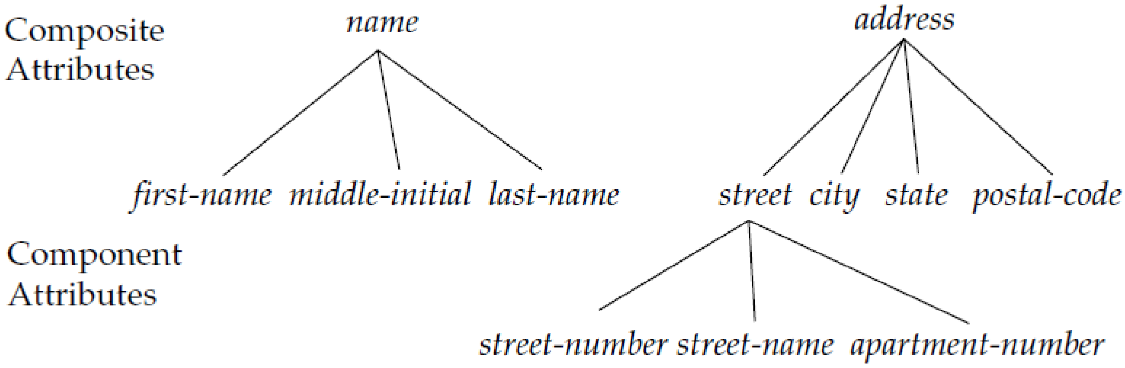


Figure 2.2 Composite attributes *customer-name* and *customer-address.*

- **Single-valued and multi-valued attributes**
  The attributes which contain only one value is said to be single-valued. For instance, the loan-number attribute for a specific loan entity refers to only one loan number.
  An attribute that can have more than one value is said to be multi-valued. For instance, An employee may have zero, one, or several phone numbers.

- **Derived and Stored attribute**
  The value for this type of attribute can be derived from the values of other related attributes or entities. For instance, suppose that the customer entity set has an attribute age, which indicates the customer's age. If the customer entity set also has an attribute date-of-birth, we can calculate age from date-of-birth and the current date. Here date-of-birth is a stored attribute, from which age is derived.

An attribute takes a **null** value when an entity does not have a value for it. that is, that the value does not exist for the entity. For example, one may have no middle name.

Relationship Sets: A **relationship** is an association among several entities. A **relationship set** is a set of relationships of the same type. Consider the two entity sets customer and loan. We define the relationship set borrower to denote the association between customers and the bank loans that the customers have.

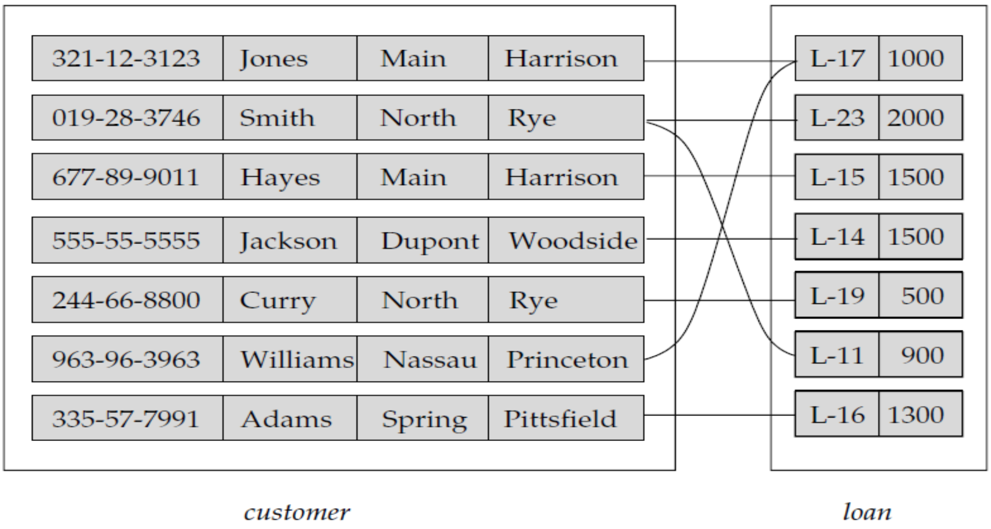
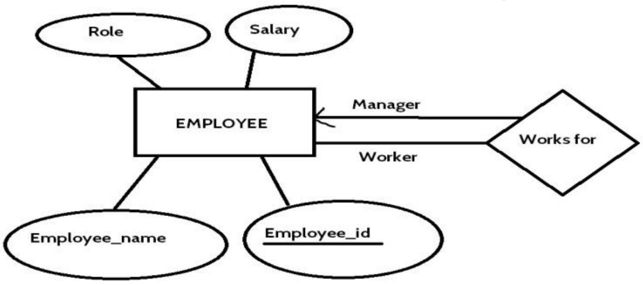

321-12-3123 Jones Main Harrison
019-28-3746 Smith North Rye
677-89-9011 Hayes Main Harrison
555-55-5555 Jackson Dupont Woodside
244-66-8800 Curry North Rye
963-96-3963 Williams Nassau Princeton
335-57-7991 Adams Spring Pittsfield
L-17 1000
L-23 2000
L-15 1500
L-14 1500
L-19 500
L-11 900
L-16 1300
customer
loan

Figure 2.3 Relationship set *borrower*.

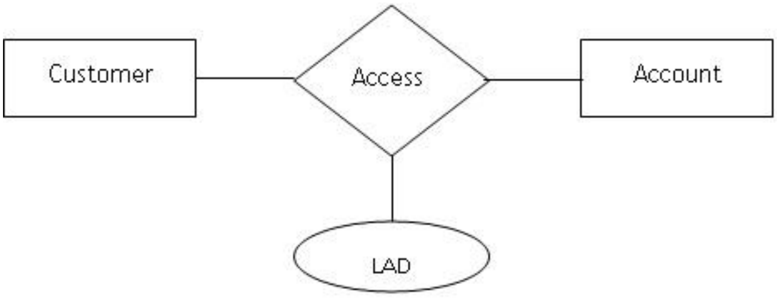
A **relationship instance** in an E-R schema represents an association between the named entities in the real-world enterprise that is being modeled. For example, the individual customer entity Hayes, who has customer identifier 677-89-9011, and the loan entity L-15 participate in a relationship instance of borrower.

The function that an entity plays in a relationship is called that entity's **role**.

When the same entity set participates in a relationship set more than once, in different roles, then it is called **recursive relationship set**. For example, consider an entity set employee, we may have a relationship set works-for that is modelled by ordered pairs of employee entities. The first employee of a pair takes the role of worker, whereas the second takes the role of manager.



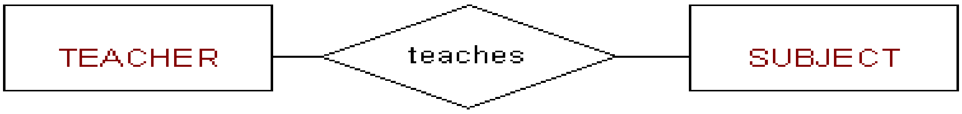
A relationship may also have attributes called **descriptive attributes**. For example, Consider a relationship set depositor with entity sets customer and account. We could associate the attribute access-date to that relationship to specify the most recent date on which a customer accessed an account.



The number of entity sets that participate in a relationship set is also the **degree** of the relationship set. A binary relationship set is of degree 2; a ternary relationship set is of degree 3. The three most common relationships in ER models are Binary, Unary and Ternary.

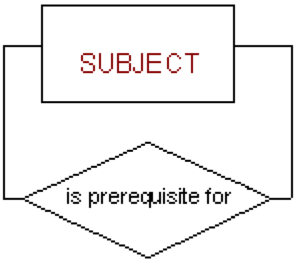
A binary relationship is when two entities participate, and is the most common relationship degree.

For Example:

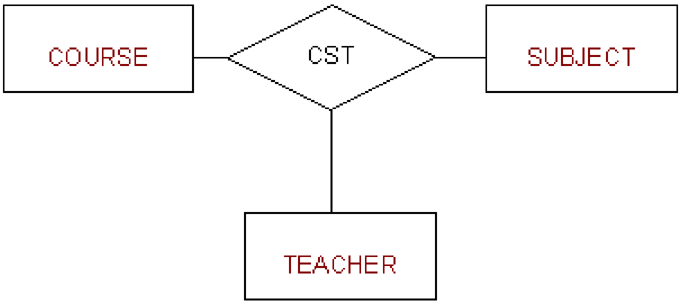


A unary relationship is when both participants in the relationship are the same entity.

For Example:



A ternary relationship is when three entities participate in the relationship.
For Example:
The University might need to record which teachers taught which subjects in which courses.



Constraints

An E-R enterprise schema may define certain constraints to which the contents of a database must conform.

1. Mapping Cardinalities

Mapping cardinalities, or cardinality ratios, express the number of entities to which another entity can be associated via a relationship set. Mapping cardinalities are most useful in describing binary relationship sets.

For a binary relationship set R between entity sets A and B, the mapping cardinality must be one of the following:

- **One to one**. An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A. (See Figure 2.4a.)
- **One to many**. An entity in A is associated with any number (zero or more) of entities in B. An entity in B, however, can be associated with at most one entity in A. (See Figure 2.4b.)
- **Many to one**. An entity in A is associated with at most one entity in B. An entity in B, however, can be associated with any number (zero or more) of entities in A. (See Figure 2.5a.)
- **Many to many**. An entity in A is associated with any number (zero or more) of entities in B, and an entity in B is associated with any number (zero or more) of entities in A. (See Figure 2.5b.)

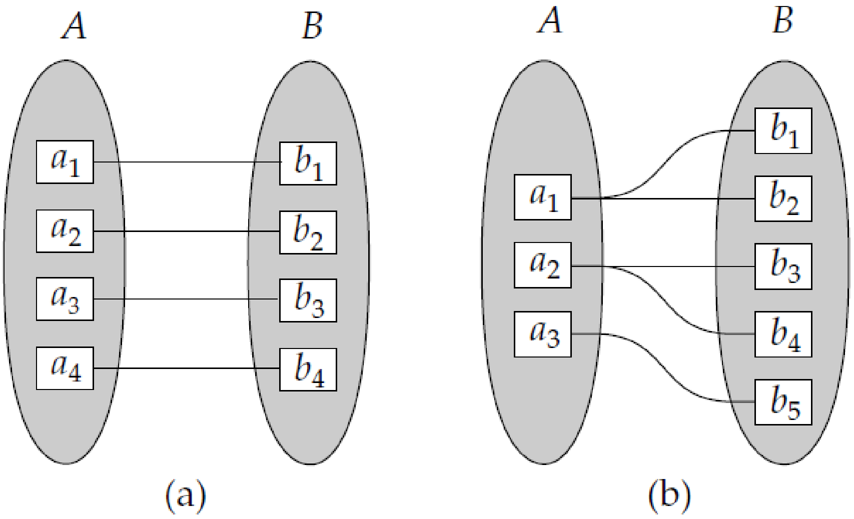


Figure 2.4 Mapping cardinalities. (a) One to one. (b) One to many.

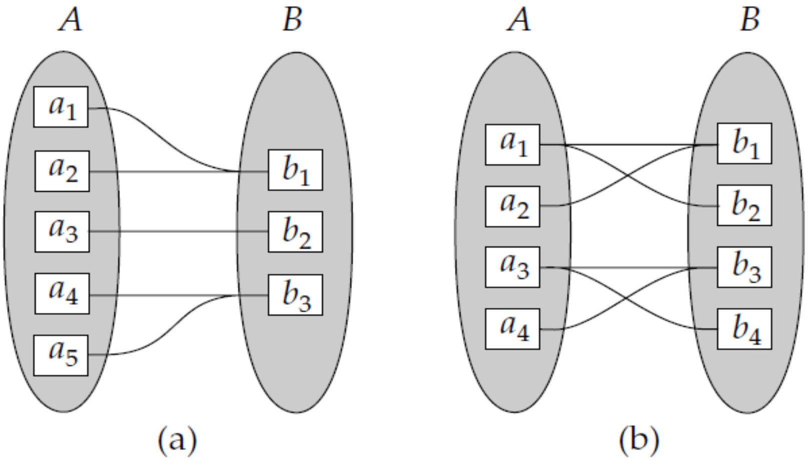
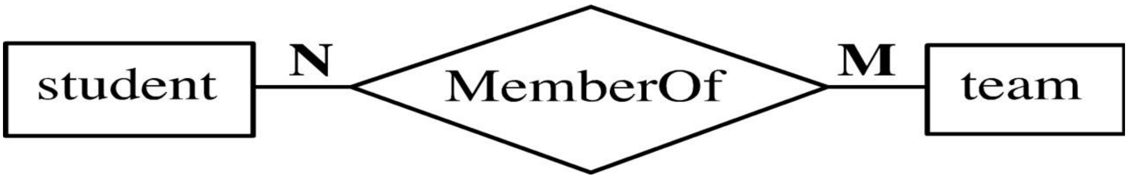
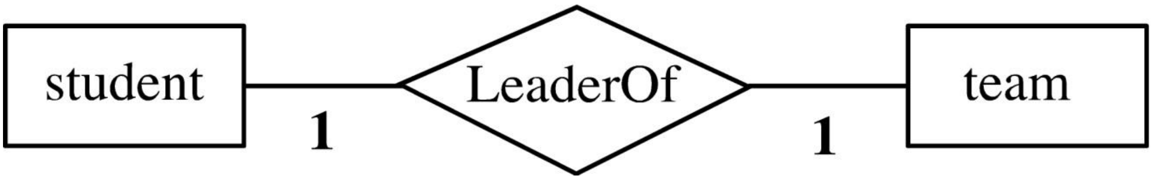


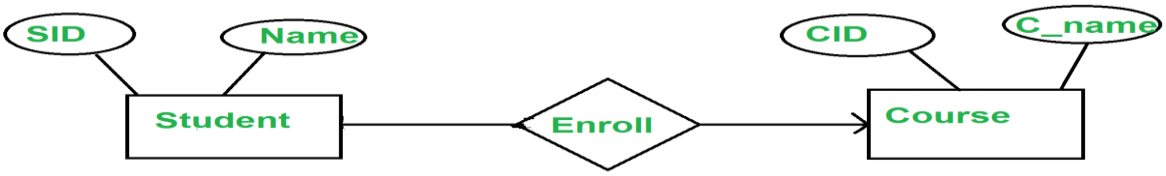
Figure 2.5 Mapping cardinalities. (a) Many to one. (b) Many to many.



Many to Many



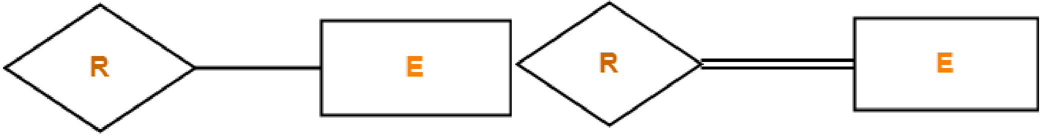
One to One



Many To One

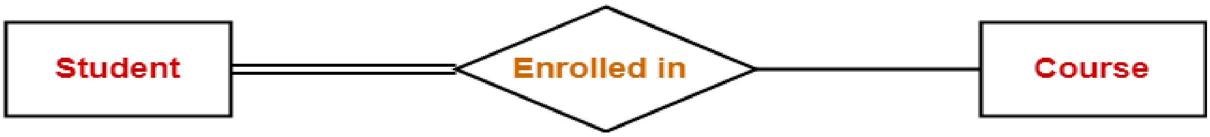
2. Participation Constraints

The participation of an entity set E in a relationship set R is said to be **total** if every entity in E participates in at least one relationship in R. If only some entities in E participate in relationships in R, the participation of entity set E in relationship R is said to be **partial**.



Partial Participation

Total Participation



Keys

A key allows us to identify a set of attributes that suffice to distinguish entities from each other. Keys also help uniquely identify relationships, and thus distinguish relationships from each other. A key is normally correlated with one column in table, although it may be associated with multiple columns.

1. **Super Key**
   A superkey is a set of one or more attributes that, taken collectively, allow us to identify uniquely an entity in the entity set. For example, the customer-id attribute of the entity set customer is sufficient to distinguish one customer entity from another. Thus, customer-id is a superkey. Similarly, the combination of customer-name and customer-id is a superkey for the entity set customer. The customer-name attribute of customer is not a superkey, because several people might have the same name.

2. **Candidate key**
   The concept of a superkey is not sufficient for our purposes, since, a superkey may contain extraneous attributes. If K is a superkey, then so is any superset of K. We are often interested in superkeys for which no proper subset is a superkey. Such minimal superkeys are called candidate keys. For example, Suppose that a combination of customer-name and customer-street is sufficient to distinguish among members of the customer entity set. Then, both {customer-id} and {customer-name, customer-street} are candidate keys. Although the attributes customerid and customer-name together can distinguish customer entities, their combination does not form a candidate key, since the attribute customer-id alone is a candidate key.

3. **Primary Key**
   It is a candidate key that is chosen by the database designer as the principal means of identifying entities within an entity set. A Primary Key is a column or a combination of columns that **uniquely identify a record**. Only one Candidate Key can be Primary Key. The primary key should be chosen such that its attributes are never changed. For example, student's Roll No.

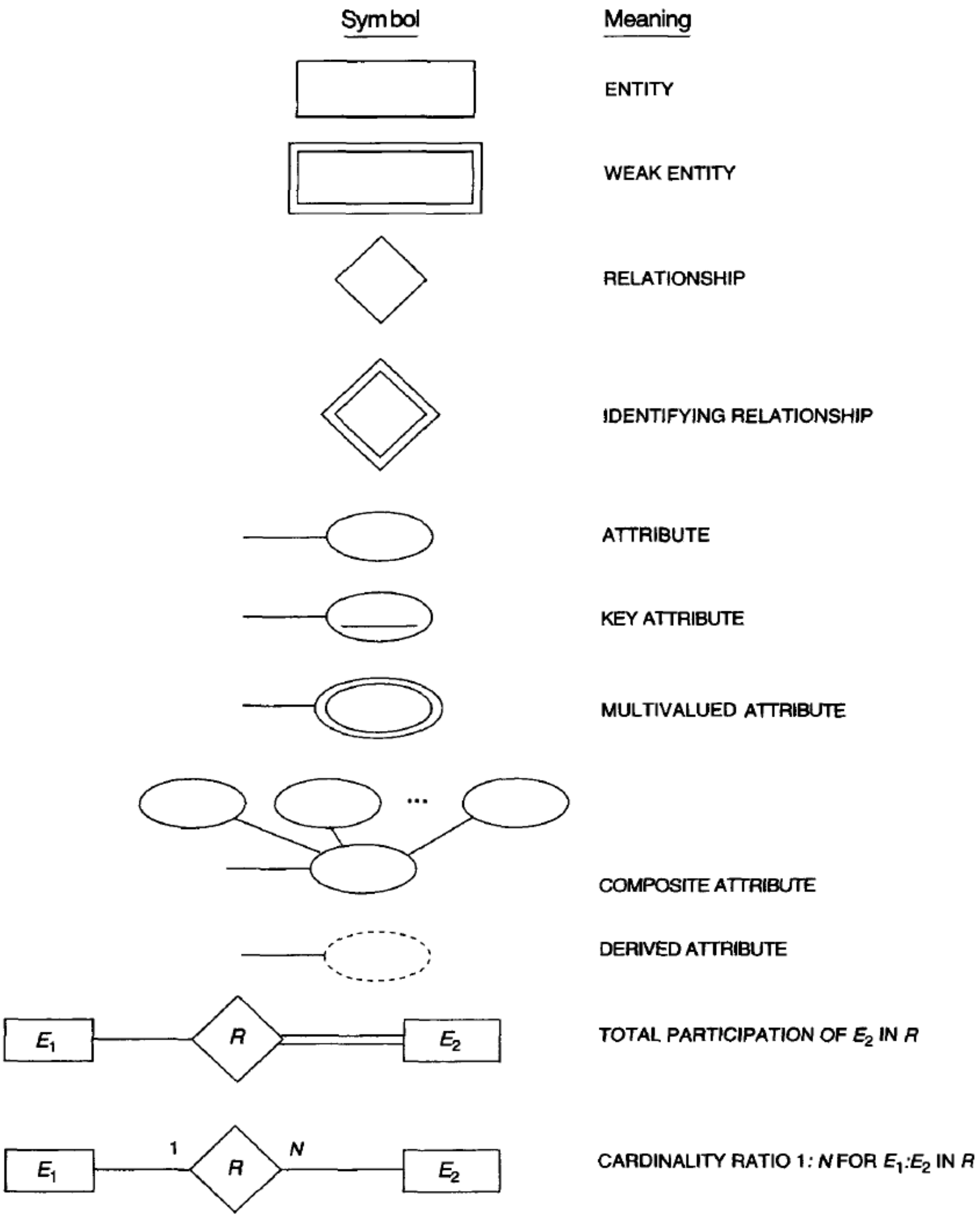
4. **Composite Key**
   If we use multiple attributes to create a primary key then that primary key is called Composite key (also called compound key or concatenated key).

5. **Foreign Key**
   A foreign key is a column or group of columns in a relational database table that provides a link between data in two tables. It acts as a cross-reference between tables because it references the primary key of another table, thereby establishing a link between them.

   For instance, if there are two tables, customer and order, a relationship can be created between them by introducing a foreign key into the order table that refers to the CUSTOMER_ID in the customer table. The CUSTOMER_ID column exists in both customer and order tables.
   The CUSTOMER_ID in the order table becomes the foreign key, referring to the primary key in the customer table. To insert an entry into the order table, the foreign key constraint must be satisfied.
   An attempt to enter a CUSTOMER_ID that is not present in the customer table fails, thus maintaining the table's referential integrity.

6. **Alternate Key**
   Any candidate key other than one chosen as a primary key is known as Alternate key.

Notations for ER- Diagram



Weak Entity Sets

An entity set may not have sufficient attributes to form a primary key. Such an entity set is termed a **weak entity set**. An entity set that has a primary key is termed a **strong entity set**. For example, consider the entity set payment, which has the three attributes: payment-number, payment-date, and payment-amount. Although each payment entity is distinct, payments for different loans may share the same payment number. Thus, this entity set does not have a primary key; it is a weak entity set.

For a weak entity set to be meaningful, it must be associated with another entity set, called the **identifying or owner entity set**. Every weak entity must be associated with an identifying entity; that is, the weak entity set is said to be **existence dependent** on the identifying entity set. The identifying entity set is said to **own** the weak entity set that it identifies. The relationship associating the weak entity set with the identifying entity set is called the **identifying relationship**. The identifying relationship is many to one from the weak entity set to the identifying entity set, and the participation of the weak entity set in the relationship is total.

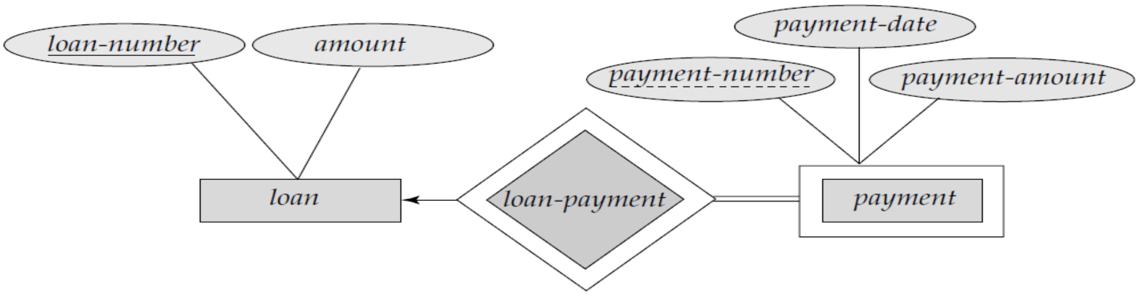
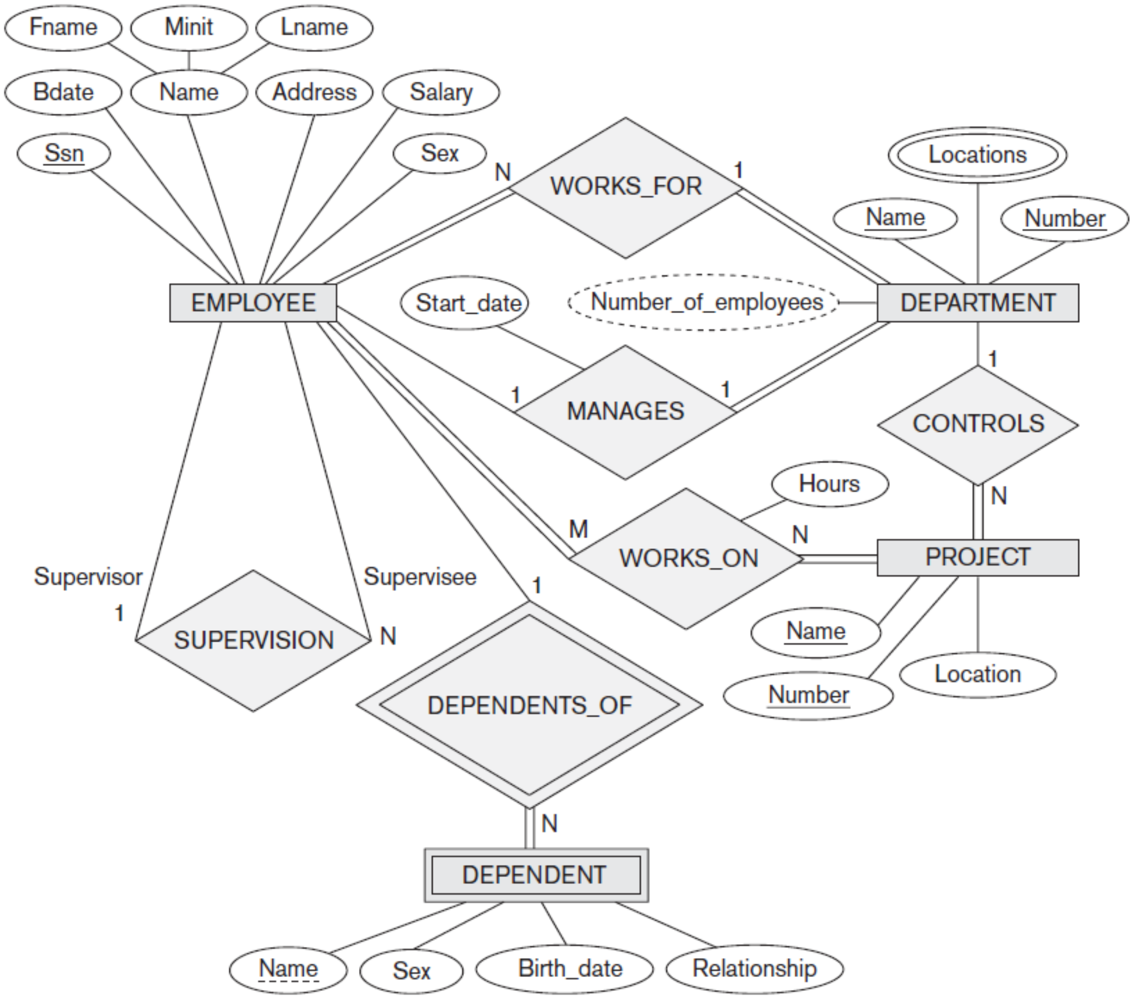


Figure 2.16 E-R diagram with a weak entity set.

EXAMPLE: COMPANY database ER Diagram

The ER conceptual schema diagram for the COMPANY database.



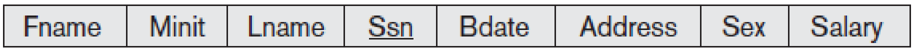
Reduction of ER Diagram into Relations

Database can be represented with the help of ER notations and these notations (diagrams) can be reduced to collection of tables. For every entity set, relationship set can be represented in tabular form.
The name of table is the name of corresponding entity set of relationship. Each table have columns which is the attributes each entity set or relationship have.

Step 1: Mapping of Regular Entity Types. For each regular (strong) entity type E in the ER schema, create a relation R that includes all the simple attributes of E. Include only the simple component attributes of a composite attribute. Choose one of the key attributes of E as the primary key for R. If the chosen key of E is a composite, then the set of simple attributes that form it will together form the primary key of R.

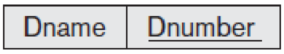
For example: we create the relations EMPLOYEE, DEPARTMENT, and PROJECT for COMPANY database schema.

EMPLOYEE



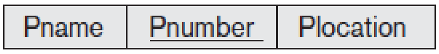
Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary

DEPARTMENT



Dname	Dnumber

PROJECT

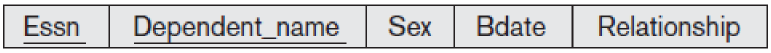


Pname	Pnumber	Plocation

Step 2: Mapping of Weak Entity Types. For each weak entity type W in the ER schema with owner entity type E, create a relation R and include all simple attributes (or simple components of composite attributes) of W as attributes of R. Also, include as foreign key attributes of R, the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s). The primary key of R is the combination of the primary key(s) of the owner(s) and the partial key of the weak entity type W, if any.

For example: we create the relation DEPENDENT in this step to correspond to the weak entity type DEPENDENT. We include the primary key Ssn of the EMPLOYEE relation—which corresponds to the owner entity type. The primary key of the DEPENDENT relation is the combination {Essn, Dependent_name}.

DEPENDENT



Essn	Dependent_name	Sex	Bdate	Relationship

Step 3: Mapping of Binary 1:1 Relationship Types.
For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R.

1. Foreign key approach: Choose one of the relations—S, say—and include as a foreign key in S the primary key of T. It is better to choose an entity type with total participation in R in the role of S. Include all the simple attributes (or simple components of composite attributes) of the 1:1 relationship type R as attributes of S.

For example: we map the 1:1 relationship type MANAGES from Figure by choosing the participating entity type DEPARTMENT to serve in the role of S because its participation in the MANAGES relationship type is total (every department has a manager). We include the primary key of the

EMPLOYEE relation as foreign key in the DEPARTMENT relation and rename it Mgr_ssn.We also include the simple attribute Start_date of the MANAGES relationship type in the DEPARTMENT relation and rename it Mgr_start_date.

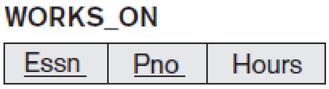
2. Merged relation approach: An alternative mapping of a 1:1 relationship type is to merge the two entity types and the relationship into a single relation. This is possible when both participations are total, as this would indicate that the two tables will have the exact same number of tuples at all times.

Step 4: Mapping of Binary 1:N Relationship Types. For each regular binary 1:N relationship type R, identify the relation S that represents the participating entity type at the N-side of the relationship type. Include as foreign key in S the primary key of the relation T that represents the other entity type participating in R.

For example: map the 1:N relationship types WORKS_FOR, CONTROLS, and SUPERVISION from Figure. For WORKS_FOR we include the primary key Dnumber of the DEPARTMENT relation as foreign key in the EMPLOYEE relation and call it Dno. For SUPERVISION we include the primary key of the EMPLOYEE relation as foreign key in the EMPLOYEE relation itself—because the relationship is recursive— and call it Super_ssn. The CONTROLS relationship is mapped to the foreign key attribute Dnum of PROJECT, which references the primary key Dnumber of the DEPARTMENT relation.

Step 5: Mapping of Binary M:N Relationship Types. For each binary M:N relationship type R, create a new relation S to represent R. Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types; their combination will form the primary key of S.

For example: we map the M:N relationship type WORKS_ON from Figure by creating the relation WORKS_ON in Figure 9.2.We include the primary keys of the PROJECT and EMPLOYEE relations as foreign keys in WORKS_ON and rename them Pno and Essn, respectively. We also include an attribute Hours in WORKS_ON to represent the Hours attribute of the relationship type. The primary key of the WORKS_ON relation is the combination of the foreign key attributes {Essn, Pno}.

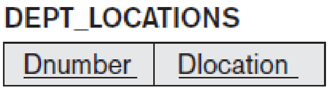


WORKS_ON

Essn	Pno	Hours

Step 6: Mapping of Multivalued Attributes. For each multivalued attribute A, create a new relation R. This relation R will include an attribute corresponding to A, plus the primary key attribute K—as a foreign key in R

For example: we create a relation DEPT_LOCATIONS. The attribute Dlocation represents the multivalued attribute LOCATIONS of DEPARTMENT, while Dnumber—as foreign key—represents the primary key of the DEPARTMENT relation. The primary key of DEPT_LOCATIONS is the combination of {Dnumber, Dlocation}.



DEPT_LOCATIONS

Dnumber	Dlocation

Step 7: Mapping of N-ary Relationship Types. For each n-ary relationship type R, where n > 2, create a new relation S to represent R. Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types. Also include any simple attributes of the n-ary relationship type.

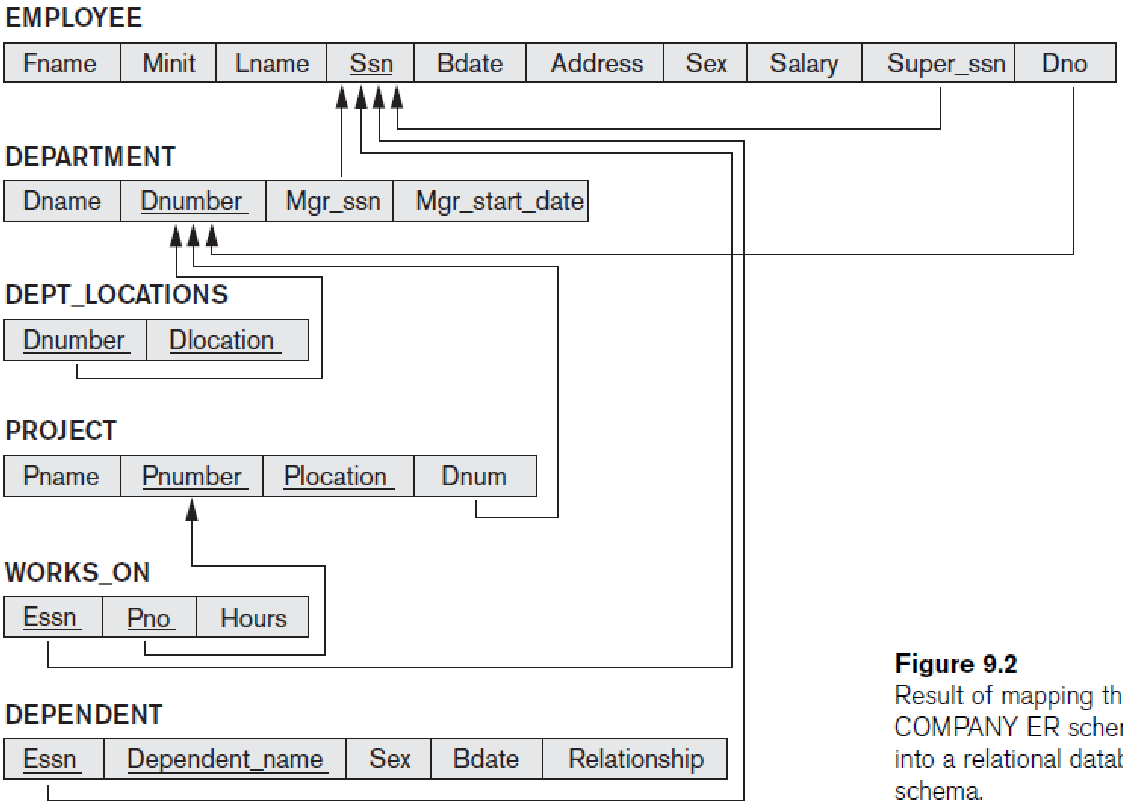


Figure 9.2
Result of mapping the COMPANY ER schema into a relational database schema.

Extended ER Model

The EER model includes all the modeling concepts of the ER model. In addition, it includes the concepts of subclass and superclass and the related concepts of specialization, generalization and aggregation.

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.

In terms of an E-R diagram, specialization is depicted by a triangle component labeled ISA, as Figure shows. The label ISA stands for “is a” and represents, for example, that a customer “is a” person. The ISA relationship may also be referred to as a superclass-subclass relationship. Higher- and lower-level entity sets are depicted as regular entity sets—that is, as rectangles containing the name of the entity set.

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

Generalization

We can think of a reverse process of abstraction in which we suppress the differences among several entity types, identify their common features, and generalize them into a single superclass of which the original entity types are special subclasses. The refinement from an initial entity set into successive levels of entity subgroupings represents a top-down design process in which distinctions are made explicit. The design process may also proceed in a bottom-up manner, in which multiple entity sets are synthesized into a higher-level entity set on the basis of common features. This commonality can be expressed by generalization, which is a containment relationship that exists between a higher-level entity set and one or more lower-level entity sets.

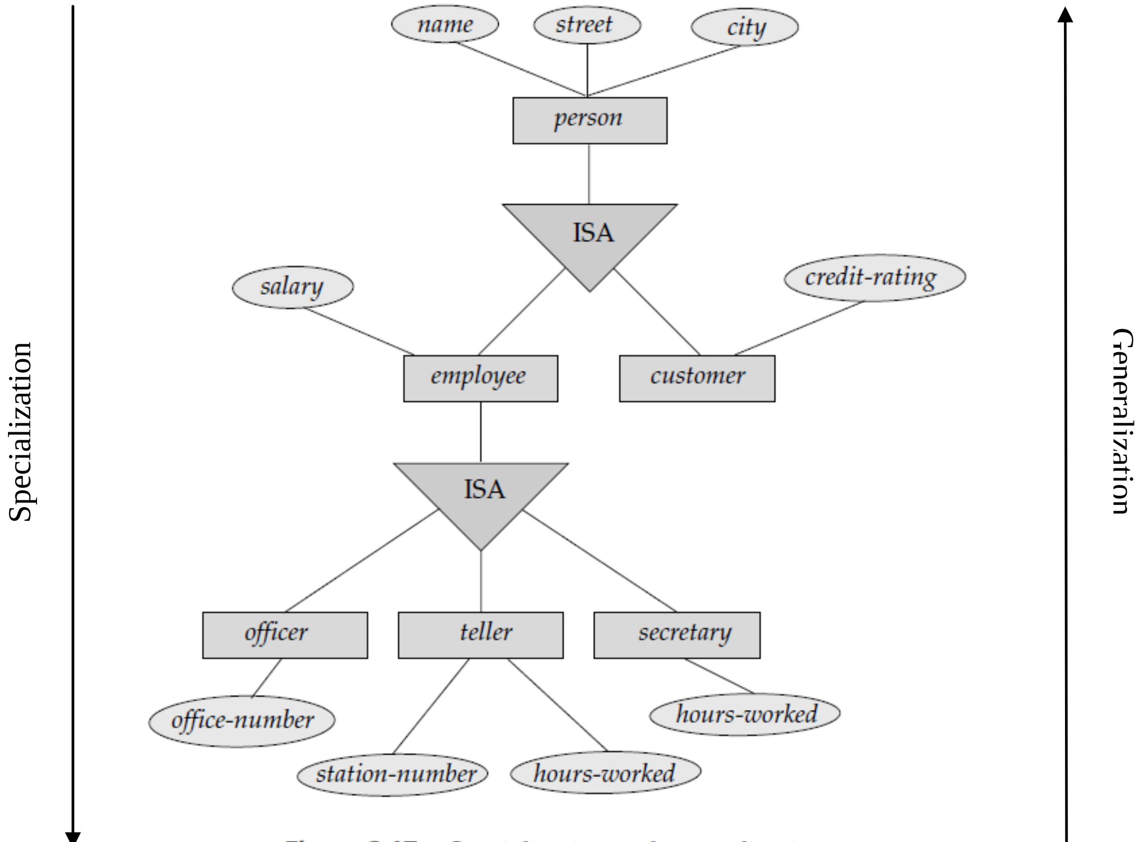
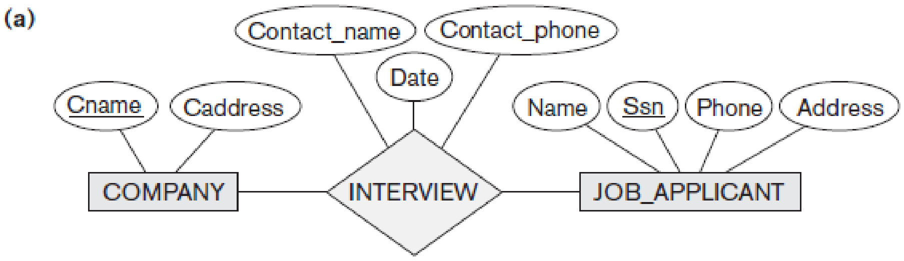


Figure 2.17 Specialization and generalization.

Aggregation

One limitation of ER model is that it cannot express relationships among relationships. Aggregation treats a relationship as a higher level entity. It allows modelling relationship between relationships. Aggregation is an abstraction in which relationship sets (along with their associated entity sets) are treated as higher level entity sets & can participate in relationships.

For Example:



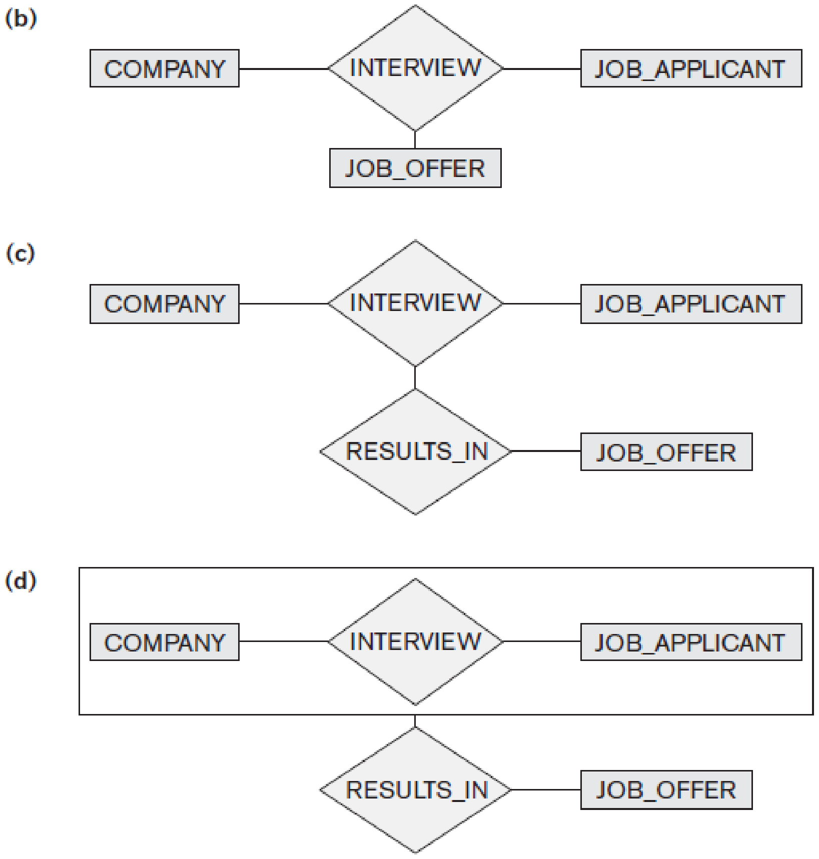


Figure 8.11
Aggregation. (a) The relationship type INTERVIEW. (b) Including JOB_OFFER in a ternary relationship type (incorrect). (c) Having the RESULTS_IN relationship participate in other relationships (not allowed in ER). (d) Using aggregation and a composite (molecular) object (generally not allowed in ER but allowed by some modeling tools). (e)

The ER schema shown in Figure 8.11(a), which stores information about interviews by job applicants to various companies. Suppose that some interviews result in job offers, whereas others do not. We would like to treat INTERVIEW as a class to associate it with JOB_OFFER. The schema shown in Figure 8.11(b) is incorrect because it requires each interview relationship instance to have a job offer. The schema shown in Figure 8.11(c) is not allowed because the ER model does not allow relationships among relationships. One way to represent this situation is to create a higher-level aggregate class composed of COMPANY, JOB_APPLICANT, and INTERVIEW and to relate this class to JOB_OFFER, as shown in Figure 8.11(d).