Unit I

Chapter 1: Introduction to Databases & Transaction

- What is database system
- Purpose of database system
- View of data
- Relational databases
- Database architecture
- Transaction Management

- What is data?

  Data is raw, unorganized facts that need to be processed. Data can be something simple and seemingly random and useless until it is organized

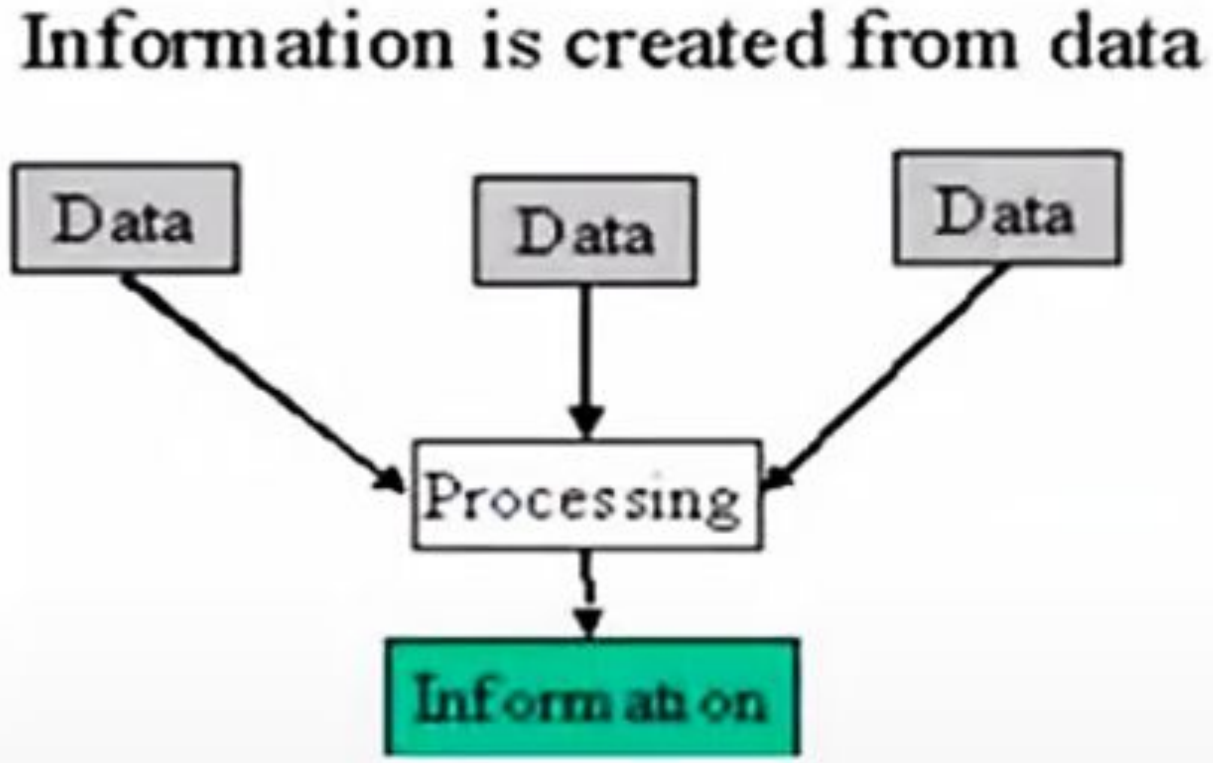
  Ex- Each students test score is one piece of data

- What is information?

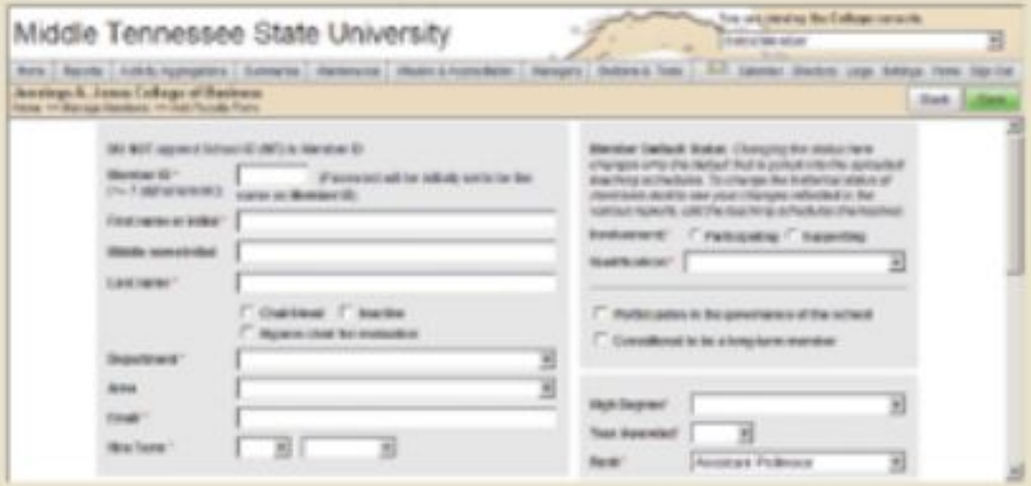
  When data is processed, organized, structured or presented in a given context so as to make it useful, is called information.

  Ex- The average score of a class or of the entire school is information that can be derived from the given data

Representation of data and information



a) Data entry screen

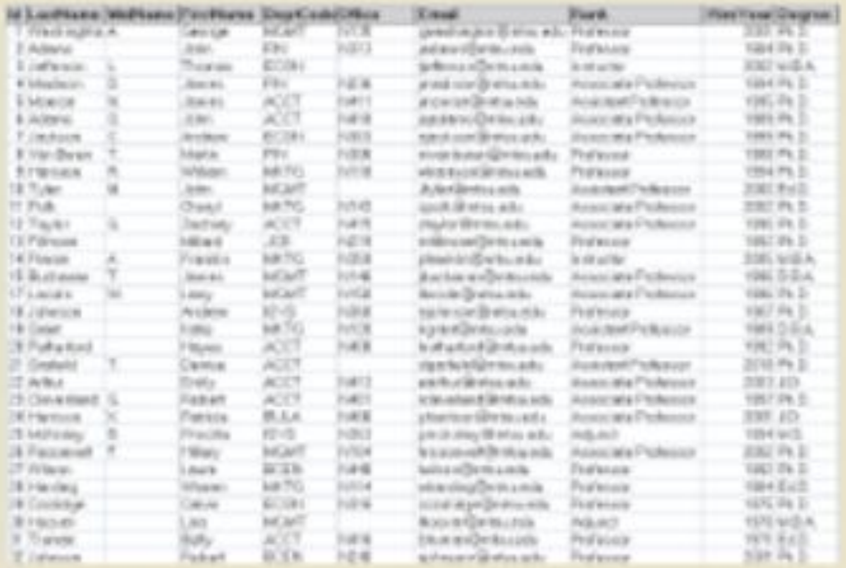


Middle Tennessee State University

Jennings A. Jones College of Business

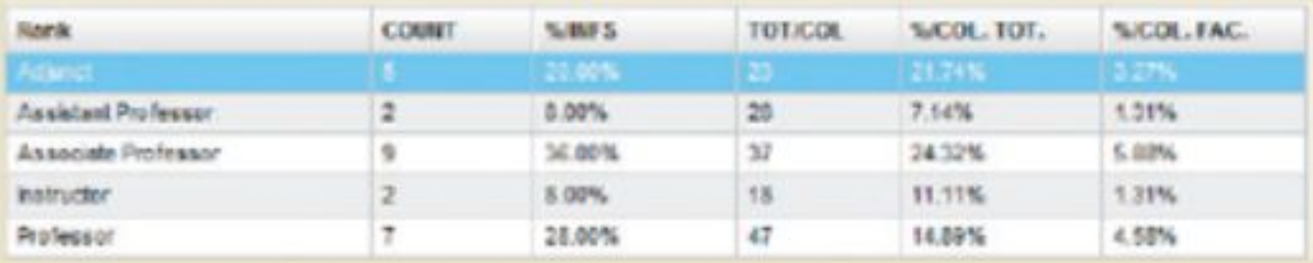
[illegible]

b) Raw data



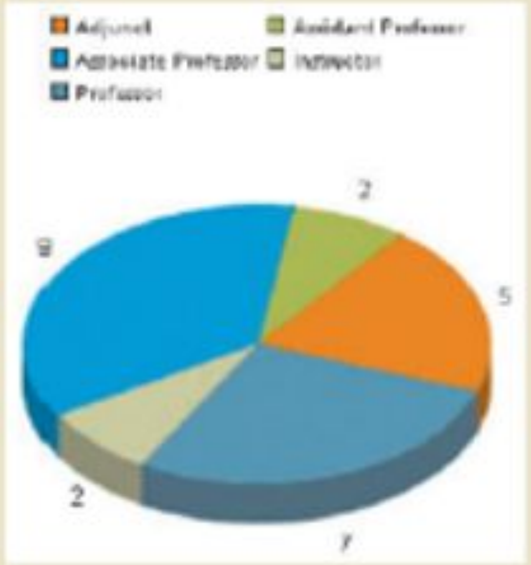
Id	LastName	MidName	FirstName	DeptCode	Office	Email	Rank	HireYear	Degree
1	Washington	A.	George	MGMT	[illegible]	[illegible]	Professor	[illegible]	Ph.D.
2	Adams		John	FIN	[illegible]	[illegible]	Professor	[illegible]	Ph.D.
3	Jefferson	L.	Thomas	ECON		[illegible]	Instructor	[illegible]	M.B.A.
4	Madison	D.	James	FIN	[illegible]	[illegible]	Associate Professor	[illegible]	Ph.D.
5	Monroe	K.	James	ACCT	[illegible]	[illegible]	Assistant Professor	[illegible]	Ph.D.
6	Adams	Q.	John	ACCT	[illegible]	[illegible]	Associate Professor	[illegible]	Ph.D.
7	Jackson	C.	Andrew	ECON	[illegible]	[illegible]	Associate Professor	[illegible]	Ph.D.
8	Van Buren	T.	Martin	FIN	[illegible]	[illegible]	Professor	[illegible]	Ph.D.
9	Harrison	R.	William	MKTG	[illegible]	[illegible]	Professor	[illegible]	Ph.D.
10	Tyler	M.	John	MGMT		[illegible]	Assistant Professor	[illegible]	Ed.D.
11	Polk		Cheryl	MKTG	[illegible]	[illegible]	Associate Professor	[illegible]	Ph.D.
12	Taylor	G.	Zachary	ACCT	[illegible]	[illegible]	Associate Professor	[illegible]	Ph.D.
13	Fillmore		Millard	[illegible]	[illegible]	[illegible]	Professor	[illegible]	Ph.D.
14	Pierce	A.	Franklin	MKTG	[illegible]	[illegible]	Instructor	[illegible]	M.B.A.
15	Buchanan	T.	James	MGMT	[illegible]	[illegible]	Associate Professor	[illegible]	D.B.A.
17	Lincoln	W.	Larry	MGMT	[illegible]	[illegible]	Associate Professor	[illegible]	Ph.D.
18	Johnson		Andrew	[illegible]	[illegible]	[illegible]	Professor	[illegible]	Ph.D.
19	Grant		Kathy	MKTG	[illegible]	[illegible]	Assistant Professor	[illegible]	D.B.A.
20	Rutherford		Hayes	ACCT	[illegible]	[illegible]	Professor	[illegible]	Ph.D.
21	Garfield	T.	[illegible]	ACCT		[illegible]	Assistant Professor	[illegible]	Ph.D.
22	Arthur		Emily	ACCT	[illegible]	[illegible]	Associate Professor	[illegible]	J.D.
23	Cleveland	G.	Robert	ACCT	[illegible]	[illegible]	Associate Professor	[illegible]	Ph.D.
24	Harrison	K.	Patricia	[illegible]	[illegible]	[illegible]	Associate Professor	[illegible]	J.D.
25	McKinley	B.	Priscilla	[illegible]	[illegible]	[illegible]	Adjunct	[illegible]	M.S.
26	Roosevelt	T.	Hillary	MGMT	[illegible]	[illegible]	Associate Professor	[illegible]	Ph.D.
27	Wilson		[illegible]	[illegible]	[illegible]	[illegible]	Professor	[illegible]	Ph.D.
28	Harding		Warren	MKTG	[illegible]	[illegible]	Professor	[illegible]	Ed.D.
29	Coolidge		Calvin	ECON	[illegible]	[illegible]	Professor	[illegible]	Ph.D.
30	Hoover		[illegible]	MGMT		[illegible]	Adjunct	[illegible]	M.B.A.
31	Truman		Betty	ACCT	[illegible]	[illegible]	Professor	[illegible]	Ed.D.
32	Johnson		Robert	[illegible]	[illegible]	[illegible]	Professor	[illegible]	Ph.D.

c) Information in summary format



Rank	COUNT	%/BFS	TOT/COL	%/COL. TOT.	%/COL. FAC.
Adjunct	5	20.00%	23	21.74%	3.27%
Assistant Professor	2	8.00%	28	7.14%	1.31%
Associate Professor	9	36.00%	37	24.32%	5.88%
Instructor	2	8.00%	18	11.11%	1.31%
Professor	7	28.00%	47	14.89%	4.58%

d) Information in graphical format



What is database?

- A database is a collection of information that is organized so that it can be easily accessed, managed and updated.
- Data is organized into rows and columns and tables and it is indexed to make it easier to find relevant information.
- Data gets updated, expanded and deleted as new information is added. Databases process workloads to create and update themselves, querying the data they contain and running applications against it.

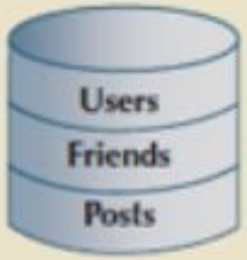
A Day In Susan's Life

See how many databases she interacts with each day

Before leaving for work, Susan checks her Facebook and Twitter accounts

Where is the data about the friends and groups stored?

Where are the "likes" stored and what would they be used for?

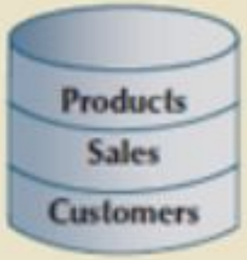


On her lunch break, she picks up her prescription at the pharmacy

Where is the pharmacy inventory data stored?

What data about each product will be in the inventory data?

What data is kept about each customer and where is it stored?

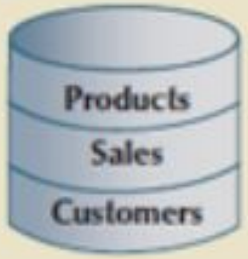


After work, Susan goes to the grocery store

Where is the product data stored?

Is the product quantity in stock updated at checkout?

Does she pay with a credit card?

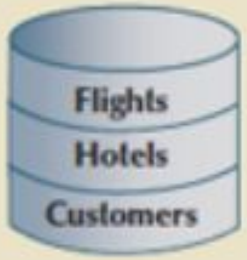


At night, she plans for a trip and buys airline tickets and hotel reservations online

Where does the online travel website get the airline and hotel data from?

What customer data would be kept by the website?

Where would the customer data be stored?

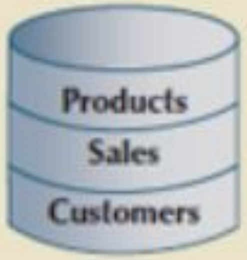


Then she makes a few online purchases

Where are the product and stock data stored?

Where does the system get the data to generate product "recommendations" to the customer?

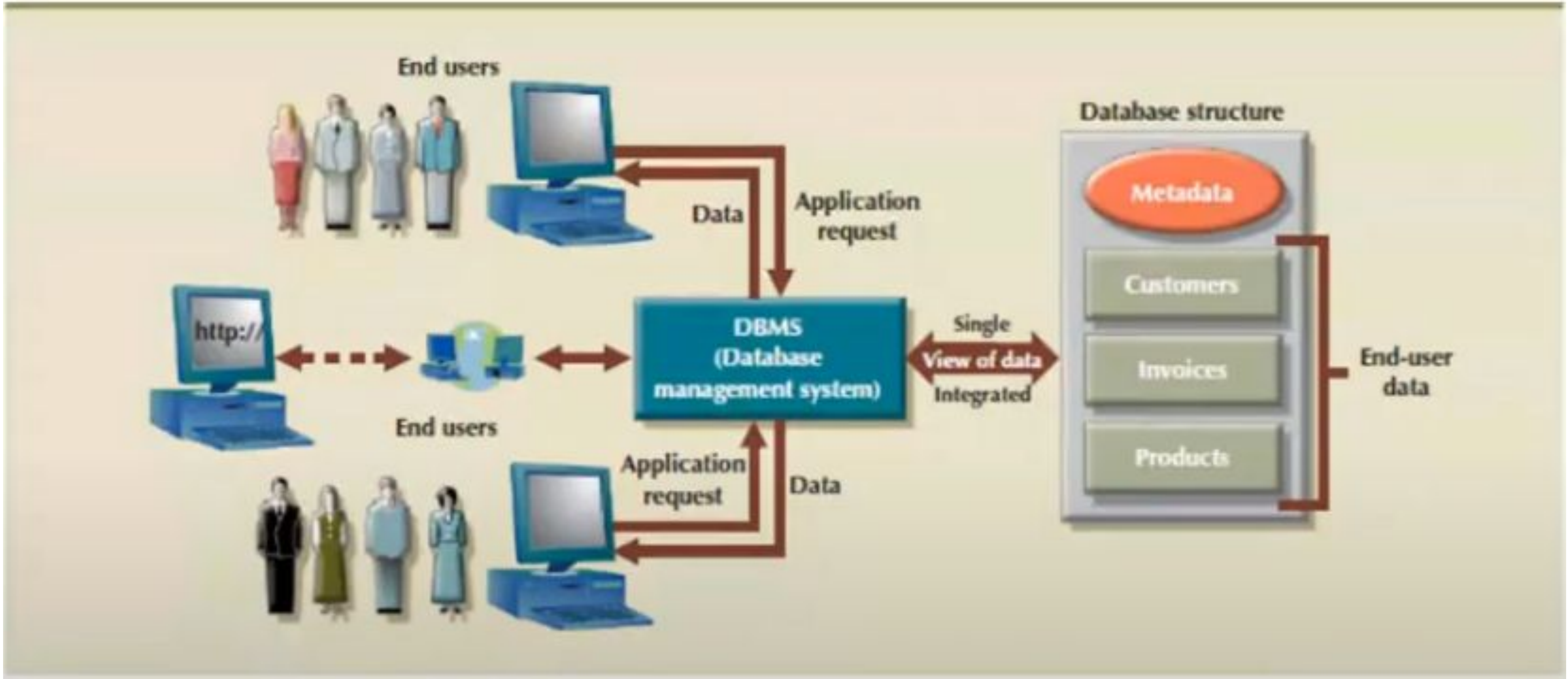
Where would credit card information be stored?



What is Database Management System?

- A database management system (DBMS) is a system software for creating and managing databases.
- The DBMS provides users and programmers with a systematic way to create, retrieve, update & manage data.
- A DBMS makes it possible for end users to create, read, update and delete data in a database.
- The DBMS essentially serves as an interface between the database and the end users or application programs, ensuring that data is consistently organized & remains easily accessible.

THE DBMS MANAGES THE INTERACTION BETWEEN THE END USER AND THE DATABASE



Purpose of Database Systems

- To overcome problems of File System
  - Data redundancy and inconsistency
  - Difficulty in accessing data
  - Data isolation
  - Security problems

Understanding File System

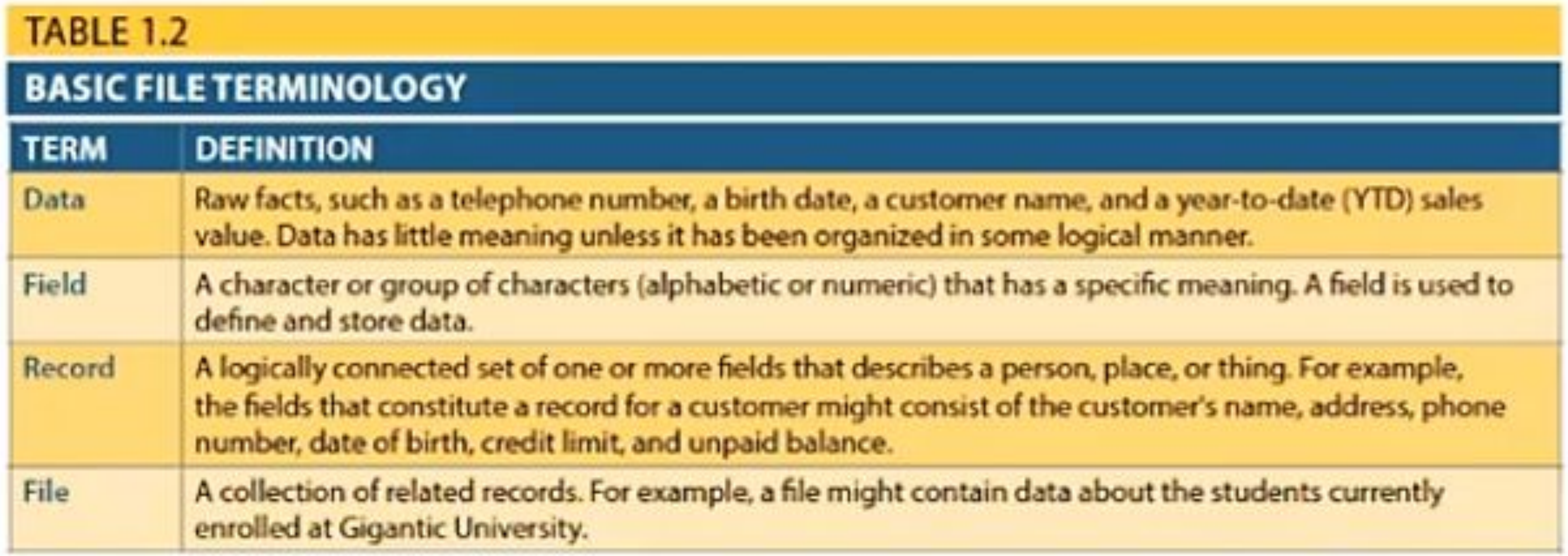
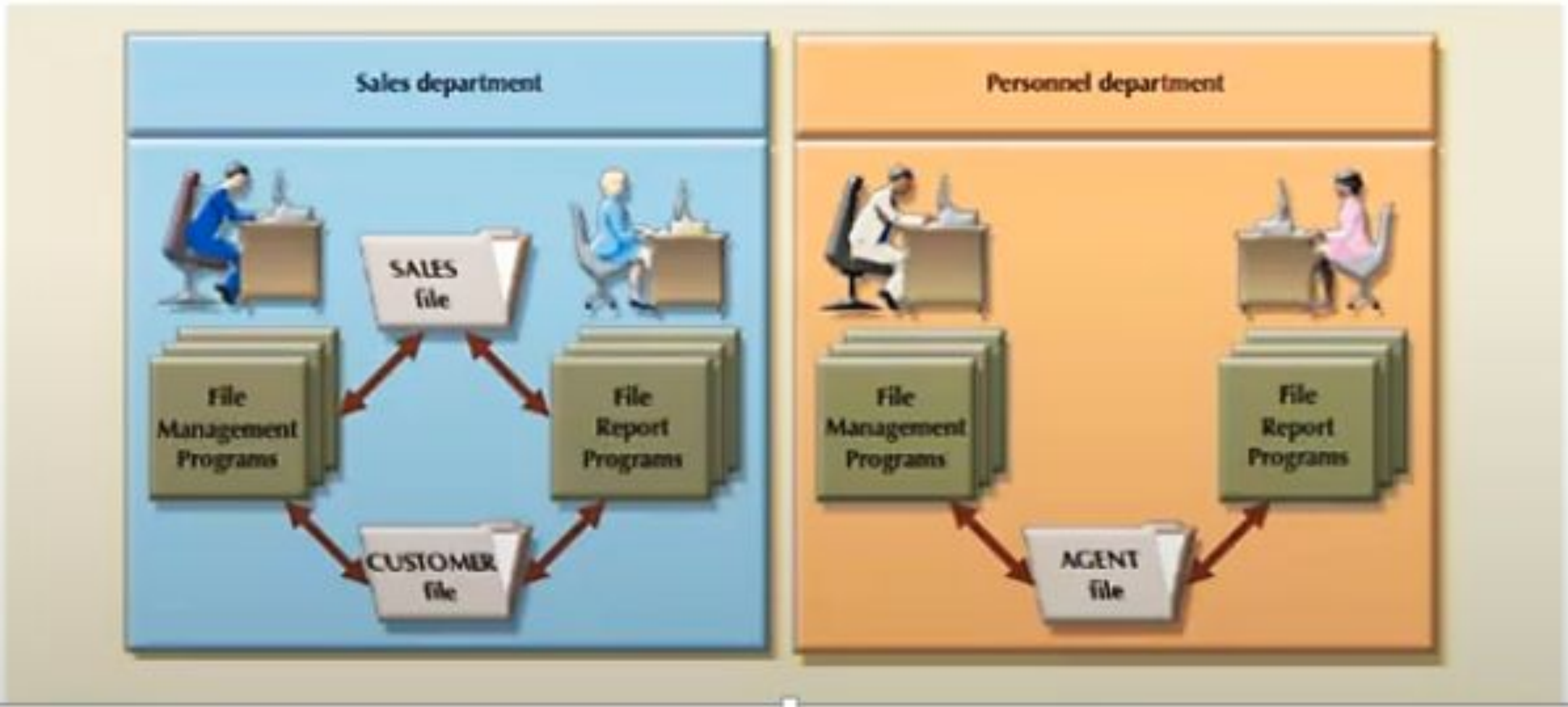


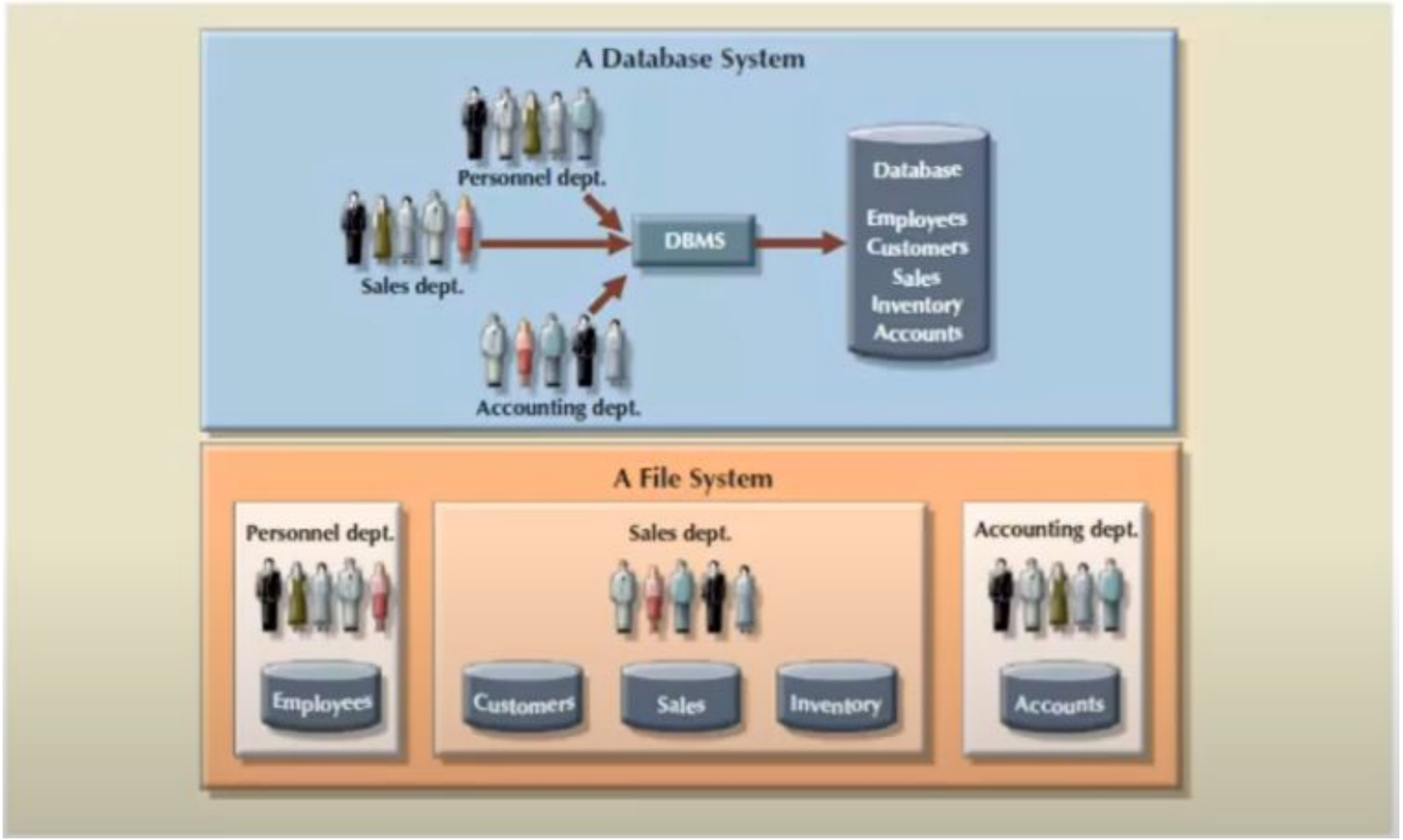
TABLE 1.2

BASIC FILE TERMINOLOGY

TERM	DEFINITION
Data	Raw facts, such as a telephone number, a birth date, a customer name, and a year-to-date (YTD) sales value. Data has little meaning unless it has been organized in some logical manner.
Field	A character or group of characters (alphabetic or numeric) that has a specific meaning. A field is used to define and store data.
Record	A logically connected set of one or more fields that describes a person, place, or thing. For example, the fields that constitute a record for a customer might consist of the customer's name, address, phone number, date of birth, credit limit, and unpaid balance.
File	A collection of related records. For example, a file might contain data about the students currently enrolled at Gigantic University.



Contrasting Database System and File System



Database System Applications

- ✔ Banking
- ✔ Airlines
- ✔ Universities
- ✔ Credit card and transactions
- ✔ Sales

View of Data

- A database system is a collection of interrelated data and a set of programs that allow users to access and modify these data

- A major purpose of a database systems to provide users an *abstract* view of the data

- That is, the system hides certain details of how the data are stored and maintained

Data Abstraction

- For the system to be usable, it must retrieve data efficiently.
- Since many database-system users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify user's interaction with system:
- **Physical level:** The lowest level of abstraction describes *how* the data are actually stored
- **Logical level:** The next-higher level of abstraction describes what data are stored in the database, and *what* relationships exist among those data; *physical data independence*
- **View level:** The highest level of abstraction describes only part of the entire database

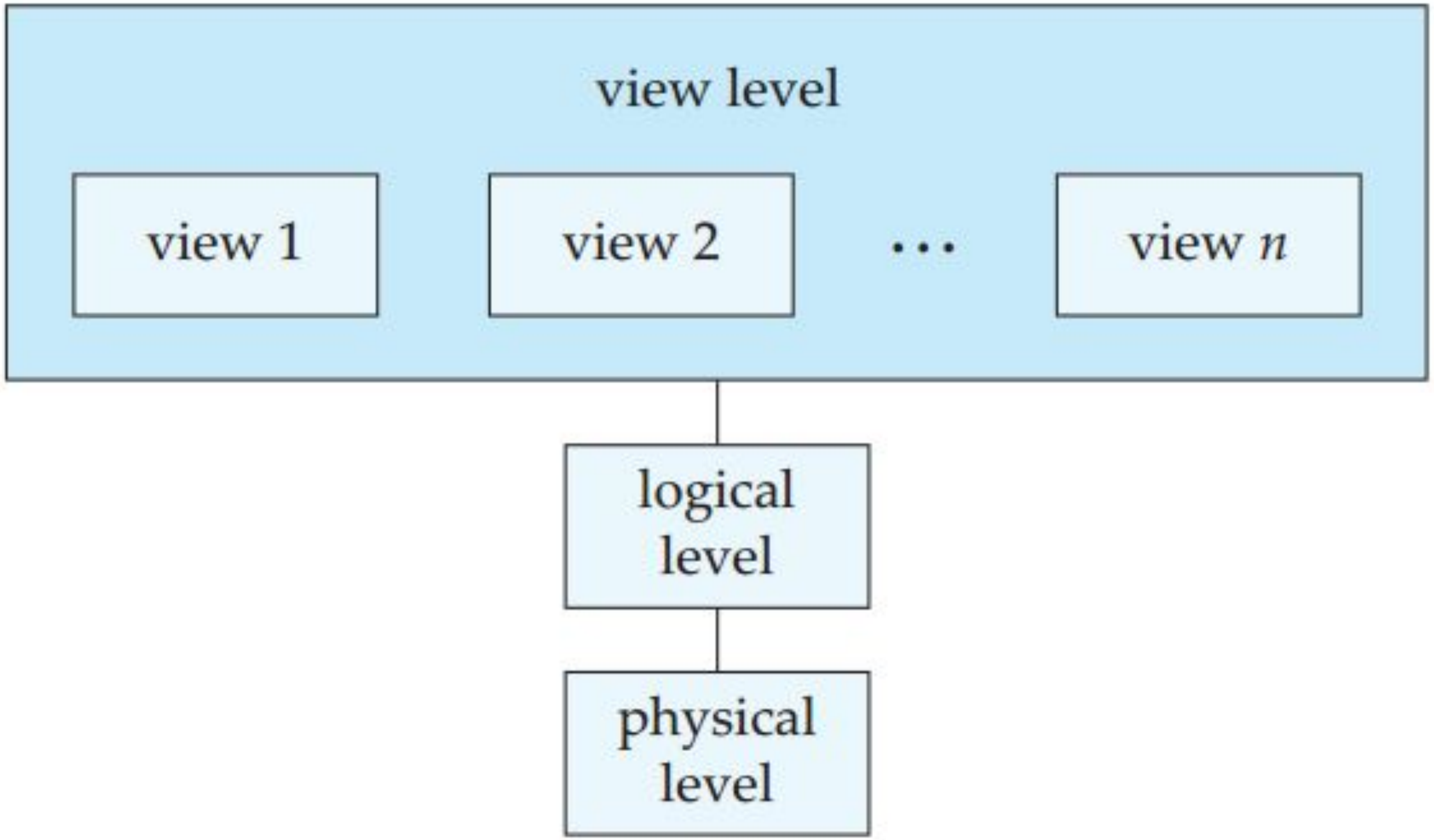


Figure 1.1 The three levels of data abstraction.

Instances and Schemas

- Databases change over time as information is inserted and deleted
- The collection of information stored in the database at a particular moment is called as an **instance** of the database
- The overall design of the database is called the database **schema**.
- The values of the variables in a program at a point in time correspond to an *instance* of a database schema.
- Database systems have several schemas, partitioned according to the levels of abstraction.

- The **physical schema** describes the database design at the physical level, while the **logical schema** describes the database design at the logical level.

- A database may also have several schemas at the view level, sometimes called subschemas, that describe different views of the database.

Data Models

- Underlying the structure of a database is the data model: a collection of conceptual tools for describing data, data relationships, data semantics and consistency constraints.
- A data model provides a way to describe the design of a database at physical, logical and view levels.
- Data models can be classified into four different categories:

1. Relational Model
2. Entity-Relationship Model
3. Object-Based Data Model
4. Semistructured Data Model

Relational Databases

- A relational database is based on relational model and uses a collection of tables to represent both data and the relationship among them.
- **Tables** : Each table has multiple column and each column has a unique name.
- The relational model is an example of record-based model. Record-based models are so named because the database is structured in fixed-format records of several types.
- Each table contains records of a particular type.

- Each record type defines a fixed number of fields, or attributes.
- The column of table corresponds to the attributes of the record type.
- Example: Refer Fig. 1.2 A Sample Relational Database

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table

Figure 1.2 A sample relational database.

Transaction Management

- Often, several operations on the database form a single logical unit of work.
- Atomicity- all-or-none requirement
- Consistency- correctness requirement
- Durability- persistence requirement
- A **transaction** is a collection of operations that performs a single logical function in a database application
- Each transaction is a unit of both atomicity and consistency.

- Thus, we require transactions do not violate any database consistency constraints.
- That is, if database was consistent when transaction started, the database must be consistent when the transaction successfully terminates.

Database Architecture

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

- Ensuring the atomicity and durability properties is the responsibility of the database system itself – specifically, of **recovery manager.**
- The database system must also perform **failure recovery,** that is detect system failures and restore the database to the state that existed prior to the occurrence of the failure.
- It is the responsibility of the **concurrency-control manager** to control the interaction among the concurrent transactions, to ensure the consistency of the database.
- The transaction manager consists of the concurrency-control manager and the recovery manager

- Database applications are usually partitioned into two or three parts.
- In **two-tier architecture**, the application resides at the client machine, where it 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 server.
- In contrast, in **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.

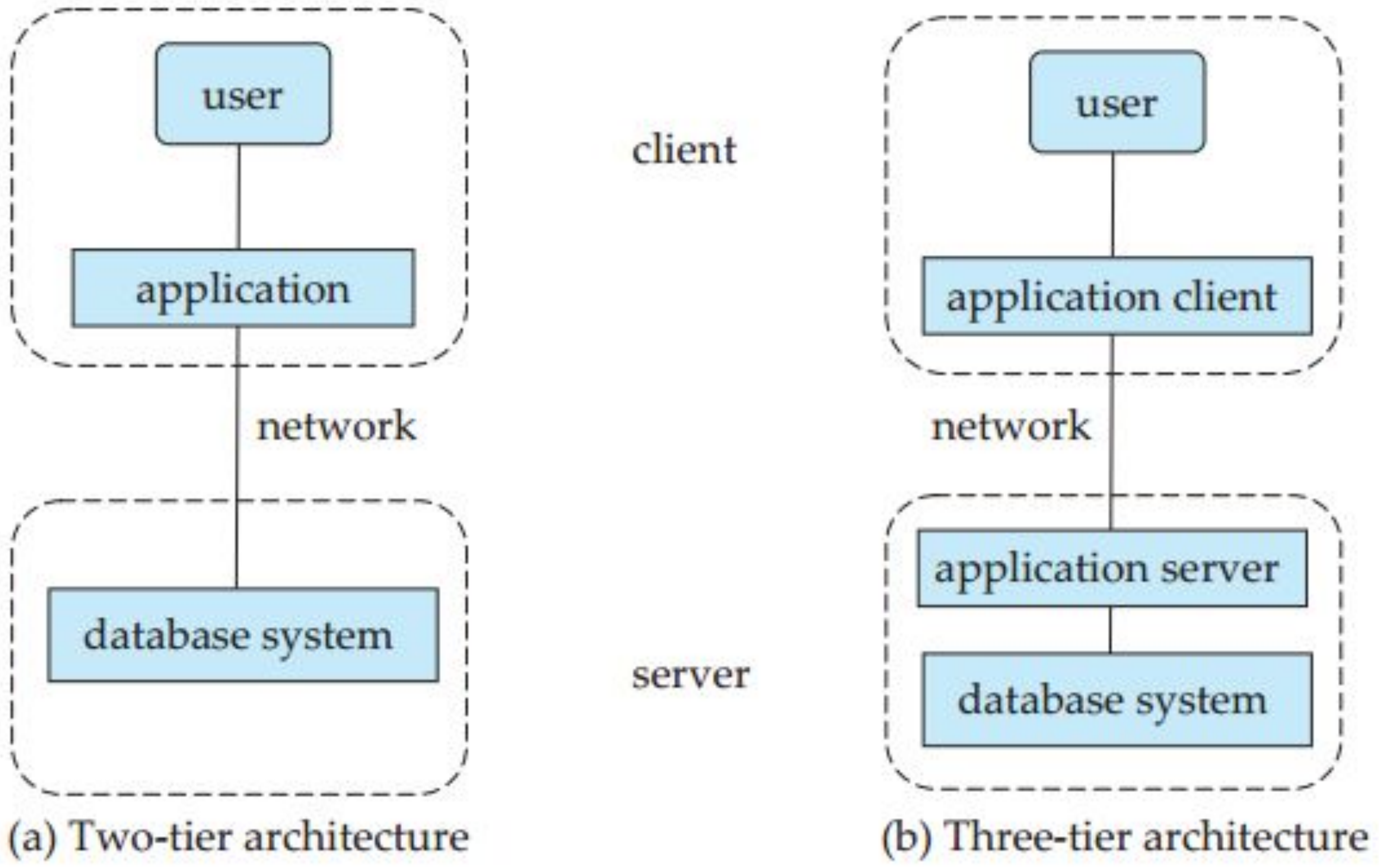


Figure 1.6 Two-tier and three-tier architectures.

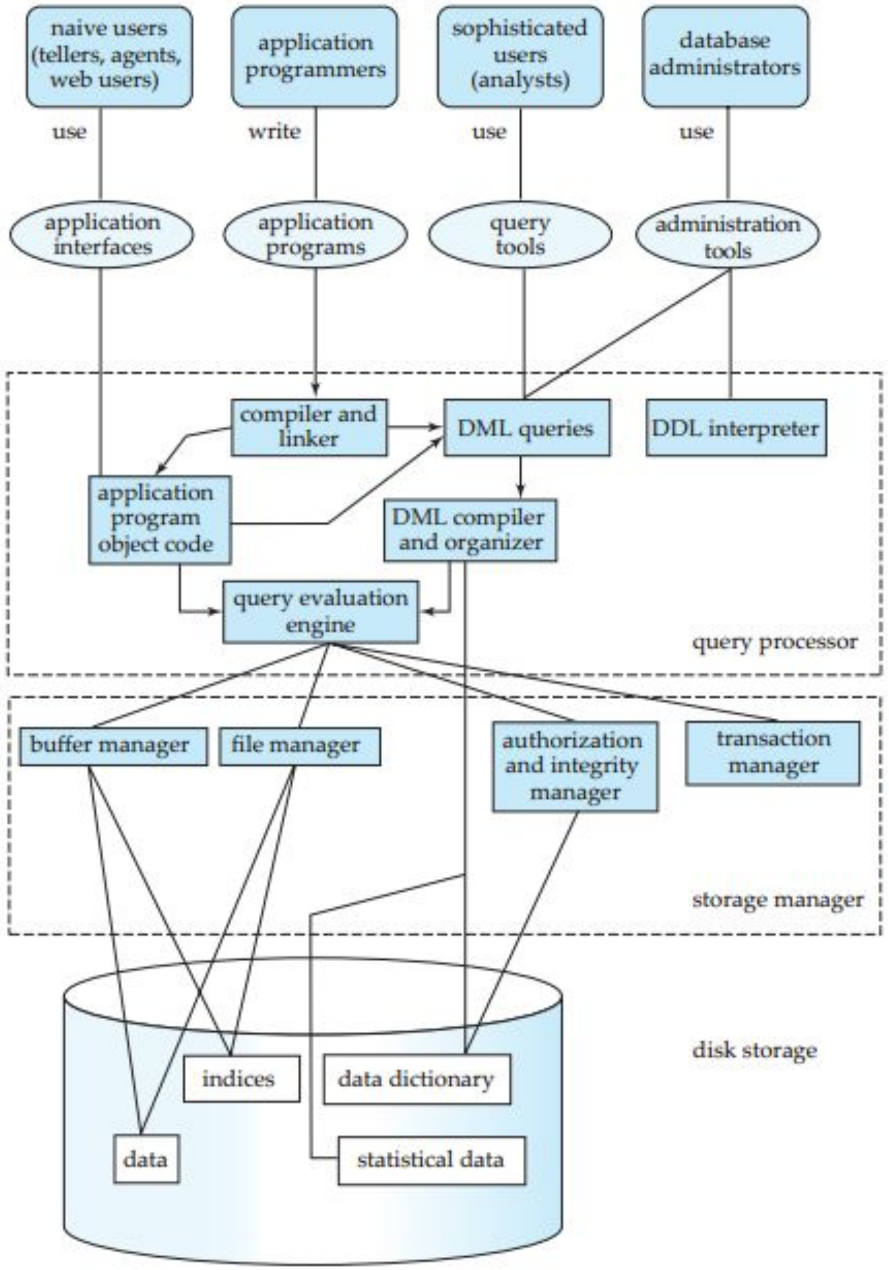
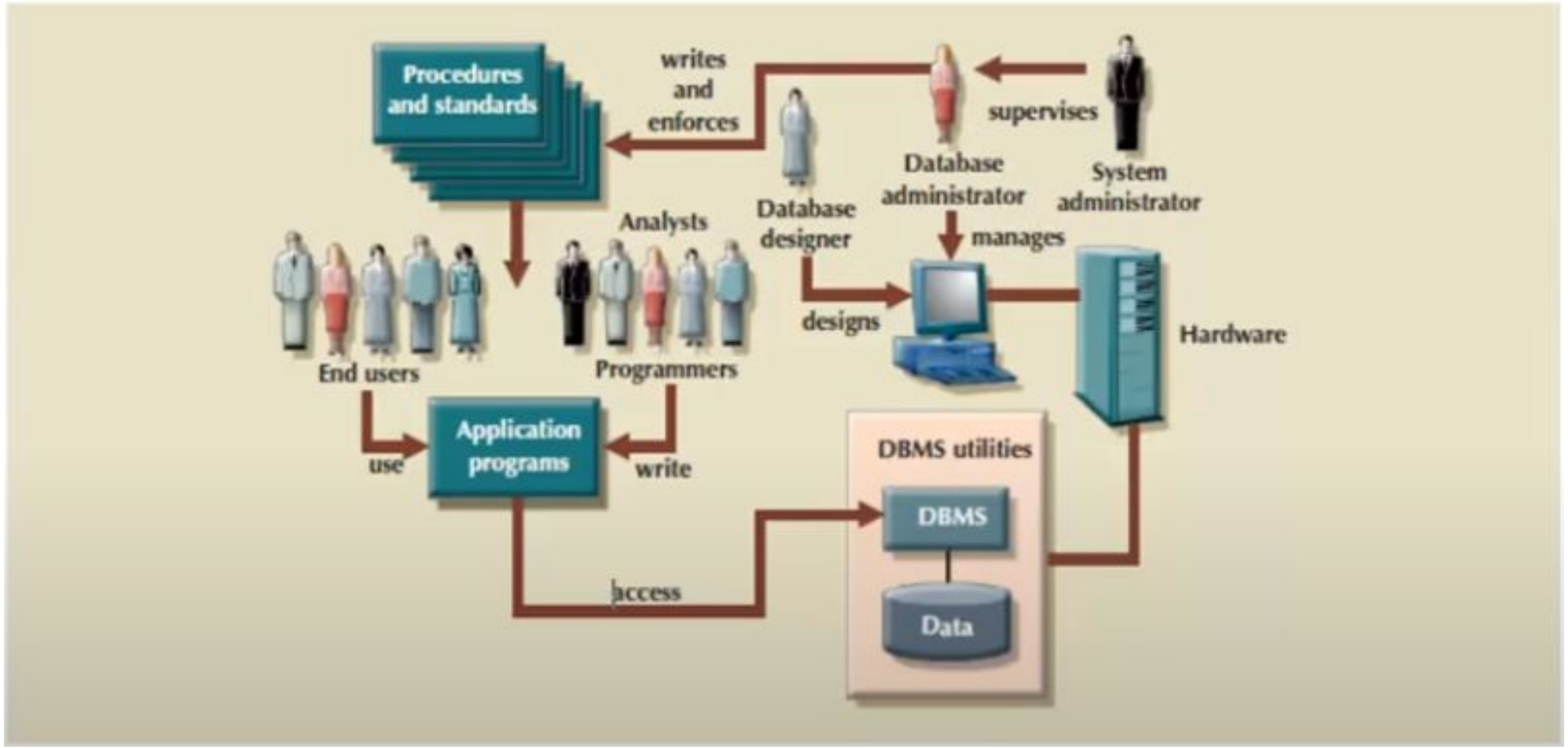


Figure 1.5 System structure.

THE DATABASE SYSTEM ENVIRONMENT



Unit I

Chapter 2: Data Models

- The importance of data models
- Basic building blocks
- Business rules
- The evolution of data models
- Degrees of data abstraction

Introduction

- Database design focuses on how the database structure will be used to store and manage end-user data.
- Data-modelling, the first step in designing a database, refers to the process of creating a specific data model for a determined problem domain.
- A **data model** is a relatively simple representation, usually graphical, of more complex real-world data structures. In general terms, a *model* is an abstraction of a more complex real-world object or event.

- Within the database environment, a data model represents data structures and their characteristics, relations, constraints, transformations and other constructs with the purpose of supporting a specific problem domain.
- Data modelling is an iterative, progressive process.
- Start with a simple understanding of the problem domain, and as understanding of the problem domain increases, so does the level of detail of the data model.
- Done properly, the final data model is in effect a “blueprint”. This blueprint is narrative and graphical in nature, i.e. it contains both text descriptions in plain, unambiguous language & clear, useful diagrams depicting main data elements

Importance of Data Models

- Data Models can facilitate interaction among the designer, the applications programmer, and the end users; *data models are a communication tool.*
- Applications are created to manage data and to help transform data into information. But data are viewed in different way by different people.
- Application programmers have another view of data, more concerned with data location, formatting and specific reporting requirements; *output, query and report screens are created.*

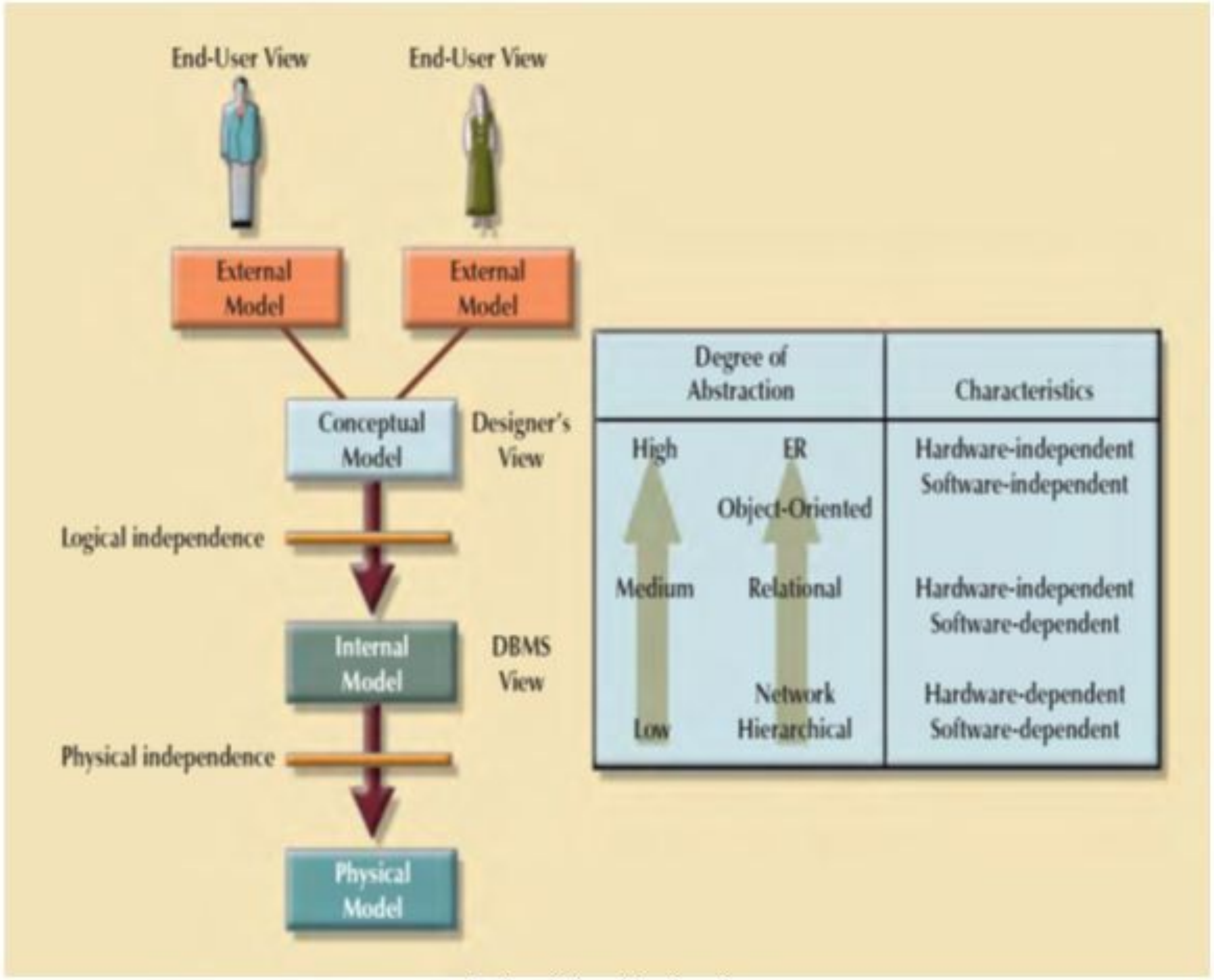
- A house is not a random collection of rooms; if someone is going to build a house, he or she should first have the overall view that is provided by blueprints.
- Likewise, a sound data environment requires an overall database blueprint based on an appropriate data model.
- A house blueprint is an abstraction; you cannot live in the blueprint. Similarly, the data model is an abstraction; you cannot draw the required data out of the data model.
- Just as you are not likely to build a good house without a blueprint, you are equally unlikely to create a good database without first creating an appropriate data model

Degrees of Data Abstraction

- If you ask 10 database designers what a data model is, you will end up with 10 different answers – depending on the degree of data abstraction.
- Designing a usable database follows the basic process, i.e. a database designer starts with an abstract view of the overall data environment and adds details as the design comes closer to implementation.
- Using levels of abstraction can also be very helpful in integrating multiple (and sometimes conflicting) views of data as seen at different levels of an organization

- In the early 1970’s, the **American National Standards Institute** (**ANSI)** Standards Planning and Requirements Committee (SPARC) defined a framework for data modelling based on degrees of data abstraction.

- The ANSI/SPARC architecture defines three levels of data abstraction: *external, conceptual and internal* which has been expanded with the addition of a *physical* model to explicitly address physical-level implementation details of the internal model



Degree of Abstraction		Characteristics
High	ER Object-Oriented	Hardware-independent Software-independent
Medium	Relational	Hardware-independent Software-dependent
Low	Network Hierarchical	Hardware-dependent Software-dependent

Data abstraction levels

Basic Building Blocks

- The basic building model of all data models are entities, attributes, relationships and constraints.
- An ***entity*** is anything (a person, a place, a thing, or an event) about which data are to be collected and stored. *An entity represents a particular type of object in real-world;* each entity occurrence is unique and distinct.
- For example: a CUSTOMER entity would have many distinguishable customer occurrences, such as John Smith, Peter Dinamita, Tom Strickland, etc.

- *An **attribute** is a characteristics of an entity. For example, a* CUSTOMER entity would be described by attributes such as customer last name, customer first name, customer phone number, customer address and customer credit limit.
- *A **relationship** describes an association among entities.* For example, a relationship exists between customers and agents that can be described as follows: an agent can serve many customers, and each customer may be served by one agent. Data models use three types of relationships: one-to-one, one-to-many, many-to-many. Database designers usually use shorthand notations 1:M or 1..*, M:N or *..*, and 1:1 or 1..1, respectively.

- **One-to-many (1:M or 1..*) relationship.** A painter paint many different paintings, but each one of them is painted by only one painter.
- Similarly, a customer (the “one”) may generate many invoices, but each invoice (the “many”) is generated by only a single customer.
- **Many-to-many (M:N or *..*) relationship.** An employee may learn many job skills, and each job skills may be learned by many employees.
- Similarly, a student can take many classes and each class can be taken by many students

- **One-to-one (1:1 or 1..1) relationship.** A retail company's management structure may require that each of its store's be managed by a single employee. In turn, each store manager, who is an employee, manages only a single store.

- *A **constraint** is a restriction placed on the data.* Constraints are important because they *help to ensure data integrity.*
- Constraints are normally expressed in the form of rules.
- For Ex
  - An employee's salary must have values that are between 6,000 and 350,000.
  - A student's GPA must be between 0.00 and 4.00
  - Each class must have one and only one teacher

Business Rules

- **What are Business Rules?**
- Business rules are created to affect the way our business works.
- 3 rules- involve the users, what they can do and what they cannot do!
- **Need of Business Rules:**
- Business rules define entities, attributes, relationships and constraints.
- Though used for organization that has policy, procedure or principle, the data can be considered significant only if business rules are defined, without them its just records!

- Business rules help employees focus on and implement the actions within the organization’s environment.

- **Identifying Business Rules:**
- They are communication tool.
- Sources of business rules are managers, policy makers, written documentation, procedures, standards, operation manuals and interviews with end users.

The Evolution of data models

- Many of the “new” database concepts and structures bear a remarkable resemblance to some of the “old” data model concepts and structures.
- **Hierarchical and Network Models**
- The Hierarchical model was developed in the 1960s to manage large amounts of data for complex manufacturing projects such as the Apollo rocket that landed on the moon in 1969. Its basic logical structure is represented by an upside-down tree.

- The hierarchical structure contains levels or segments. A **segment** is the equivalent of a file system's record type.
- The **network model** was created to represent complex data relationships more effectively than the hierarchical model, to improve database performance, and to impose database standard.
- While the network database model is generally not used today, definitions of standard database concepts that emerged with the network model are still used by modern data models

- Some important concepts that were defined at this time are:
- The **schema,**
- The **sub-schema,**
- A **data management language (DML)**
- A schema **data definition language (DDL)**
- Because of the disadvantages of the hierarchical and network models, they were largely replaced by the relational data model in the 1980s

The Relational model

- The **relational model** was introduced in 1970 by E. F. Codd (of IBM)
- The relational model foundation is a mathematical concept known as relation. To avoid the complexity of abstract mathematical theory, you can think of **relation** (sometimes called a **table**) as a matrix composed of intersecting rows and columns.
- Each row in a table is called **tuple**. Each column represents an **attribute**

- For most relational database software, the query language is Structured Query Language (SQL) which allows user to specify what must be done without specifying how it must be done.
- The RDBMS uses SQL to translate user queries into instructions for retrieving the requested data. SQL makes it easier to retrieve data with far less effort than any other database or file environment.
- From an end-user perspective, any SQ-based relational database application involves three parts: a user interface, a set of tables stored in database and the SQL “engine.”

The Entity Relationship Model

- The entity relationship (ER) model, or ERM, Peter Chen first introduced the ER data model in 1976; it was the graphical representation of entities and their relationships in a database structure

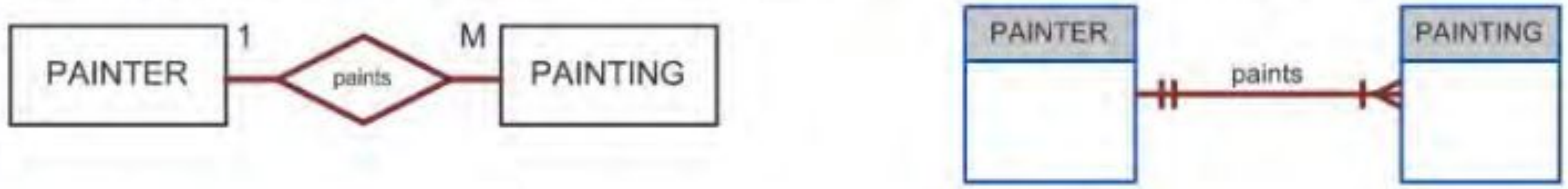
- ER models are normally represented in an entity relationship diagram (ERD), which uses graphical representations to model database components.

- The ER model is based on the following components:
- Entity. Row of relational model- entity instance or entity occurrence
- Relationships. Connectivity

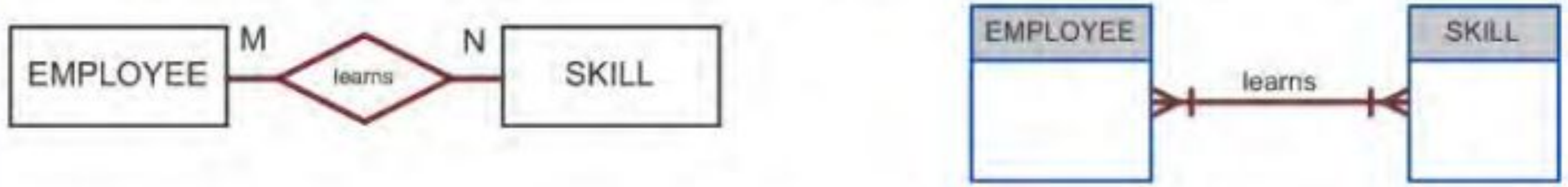
FIGURE 2.3 The Chen and Crow's Foot notations

Chen Notation *Crow's Foot Notation*

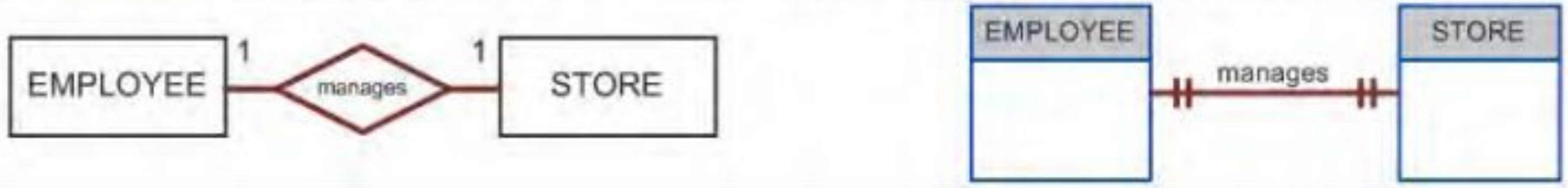
A One-to-Many (1:M) Relationship: a PAINTER can paint many PAINTINGs; each PAINTING is painted by one PAINTER.



A Many-to-Many (M:N) Relationship: an EMPLOYEE can learn many SKILLs; each SKILL can be learned by many EMPLOYEEs.



A One-to-One (1:1) Relationship: an EMPLOYEE manages one STORE; each STORE is managed by one EMPLOYEE.



The Object Oriented(OO) model

- In the object-oriented data model (OODM), both data *and their relationships* are contained in a single structure known as an object.
- In turn, the OODM is the basis for the object-oriented database management system (OODBMS).
- The OODM is said to be a semantic data model because *semantic* indicates meaning.
- The OO data model is based on the following components:

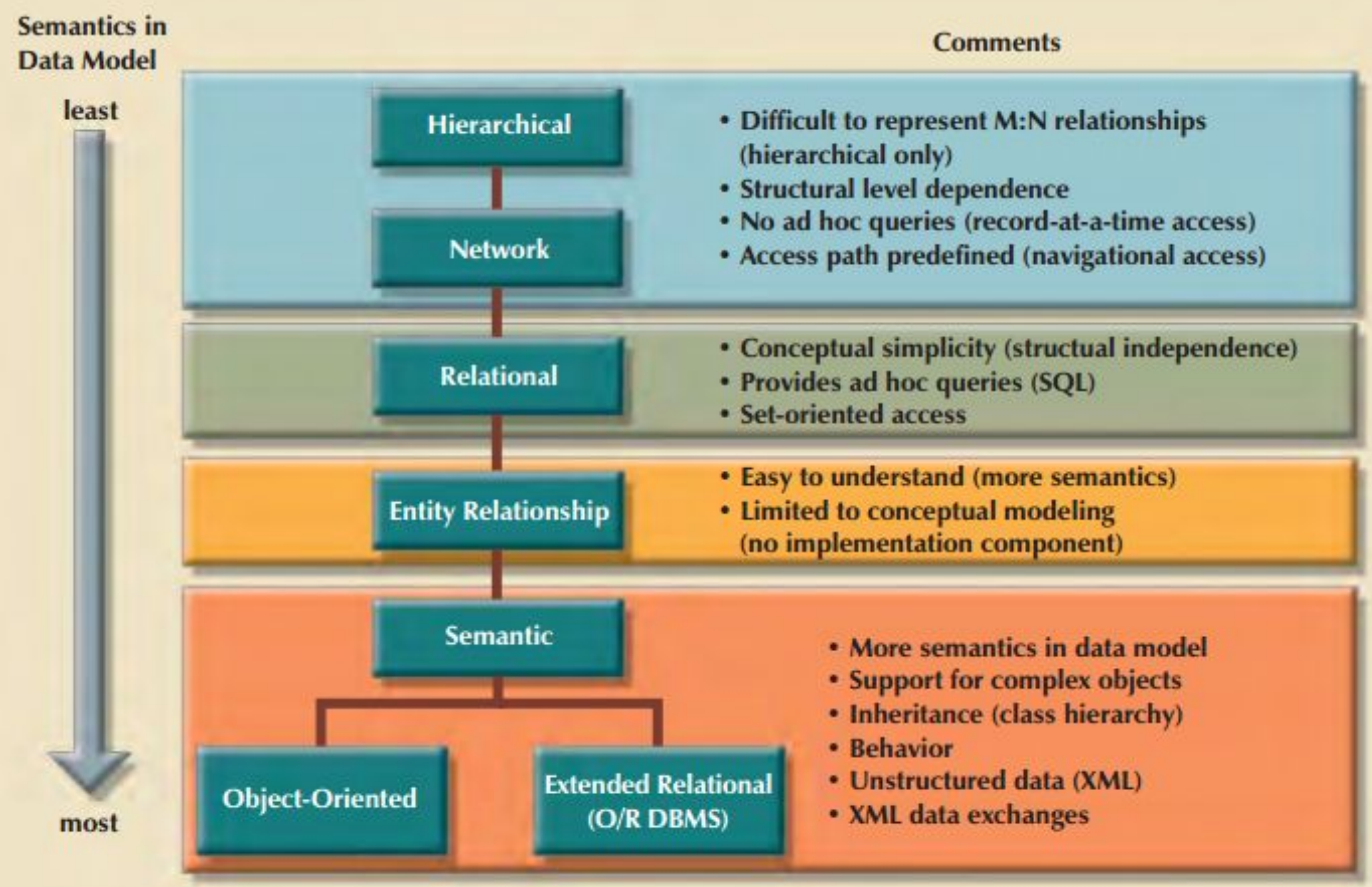
- An **object** is an abstraction of a real-world entity. In general terms, an object may be considered equivalent to an ER model’s entity. More precisely, an object represents only one occurrence of an entity.
- **Attributes** describe the properties of an object.
- Objects that share similar characteristics are grouped in classes. A **class** is a collection of similar objects with shared structure (attributes) and behavior (methods).
- A class’s **method** represents a real-world action i.e. it contains a set of *procedures;* methods are the equivalent of *procedures* in traditional programming languages. In OO terms, methods define an object’s *behavior*.

- Classes are organized in a *class hierarchy*. The **class hierarchy** resembles an upside-down tree in which each class has only one parent.
- **Inheritance** is the ability of an object within the class hierarchy to inherit the attributes and methods of the classes above it.
- Object-oriented data models are typically depicted using Unified Modeling Language (UML) class diagrams.
- **Unified Modeling Language (UML)** is a language based on OO concepts that describes a set of diagrams and symbols that can be used to graphically model a system.

TABLE 2.1 **Evolution of Major Data Models**

GENERATION	TIME	DATA MODEL	EXAMPLES	COMMENTS
First	1960s–1970s	File system	VMS/VSAM	Used mainly on IBM mainframe systems Managed records, not relationships
Second	1970s	Hierarchical and network	IMS ADABAS IDS-II	Early database systems Navigational access
Third	Mid-1970s to present	Relational	DB2 Oracle MS SQL-Server MySQL	Conceptual simplicity Entity relationship (ER) modeling and support for relational data modeling
Fourth	Mid-1980s to present	Object-oriented Object/relational (O/R)	Versant Objectivity/DB DB/2 UDB Oracle 11g	Object/relational supports object data types Star Schema support for data warehousing Web databases become common
Next generation	Present to future	XML Hybrid DBMS	dbXML Tamino DB2 UDB Oracle 11g MS SQL Server	Unstructured data support O/R model supports XML documents Hybrid DBMS adds an object front end to relational databases

FIGURE 2.5 The evolution of data models


Semantics in Data Model
least
most
Comments
Hierarchical
Network
Relational
Entity Relationship
Semantic
Object-Oriented
Extended Relational (O/R DBMS)
• Difficult to represent M:N relationships (hierarchical only)
• Structural level dependence
• No ad hoc queries (record-at-a-time access)
• Access path predefined (navigational access)
• Conceptual simplicity (structual independence)
• Provides ad hoc queries (SQL)
• Set-oriented access
• Easy to understand (more semantics)
• Limited to conceptual modeling (no implementation component)
• More semantics in data model
• Support for complex objects
• Inheritance (class hierarchy)
• Behavior
• Unstructured data (XML)
• XML data exchanges

Unit I

Chapter 3: Data Design, ER Diagram and Unified Modelling Language

- Database Design and ER model: overview, ER model, Constraints.
- ER Diagrams
- ERD issues
- Weak entity sets
- Codd’s rules
- Relational schemas
- Introduction to UML

Overview of Design Process

- The task of creating a database application is a complex one, involving design of database schema, design of the programs that access and update the data, and design of security scheme to control access to data
- The need of the users play a central role in design process.
- **Design Phases**
- For a small application, it may be feasible for a database designer who understands the application requirement to decide directly on the relations to be created, their attributes and constraints on the relations.

- However, such a direct design process is difficult for real-world applications, since they are often highly complex.
- Often no one person understands the complete data needs of an application
- The database designer must

✔ interact with users of the application to understand the needs of the application,

✔ Represent them in a high-level fashion that can be understood by the users,

✔ And then translate the requirements into lower levels of the design.

- The initial phase of database design is to characterize fully the data needs of the database users.
- Next, the designer chooses a data model and, by applying the concepts of the chosen data model, translates these requirements into a conceptual schema of the database. The schema developed at this conceptual-design phase provides a detailed overview of the enterprise.
- A fully developed conceptual schema also indicates the functional requirements of the enterprise.

- In a **specification of functional requirements**, users describe the kinds of operations (or transactions) that will be performed on the data.
- The process of moving from an abstract data model to the implementation of the database proceeds in two final design phases.
- **Logical-design phase,**
- **Physical-design phase**

- **Design Alternatives**
- A major part of the database design process is deciding how to represent in the design the various types of “things” such as people, places, products, and the like.
- We use the term *entity* to refer to any such distinctly identifiable item.
- In designing a database schema, we must ensure that we avoid two major pitfalls:

    1. Redundancy   2. Incompleteness

- Avoiding bad design is not enough. There may be large number of good designs from which we must choose.

Entity Relationship Model

- The **entity-relationship (E-R)** data model was developed to facilitate database design by allowing specification of an *enterprise schema* that 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 concepts: entity set, relationship sets and attributes.

Entity Sets

- An entity is a “thing” or “object” in the real world that is distinguishable from all other objects.
- For example, each person in a university is an entity.
- An entity has a set of properties, and the values for some set of properties may uniquely identify an entity.
- An **entity set** is a set of entities of the same type that share the same properties, or attributes.
- The set of all people who are instructors at a given university, can be defined as the entity set *instructor.*
- Similarly, the entity set *student* might represent the set of all students in the university .

- An entity is represented by a set of **attributes.**
- Attributes are descriptive properties possessed by each member of an entity set.
- Each entity has a **value** for each of its attributes.
- ***Relationship Sets***
- A relationship is an association among several entities.
- A **relationship set** is a set of relationships of the same type.
- A relationship may also have attributes called **descriptive attributes.**

- The number of entity sets that participate in a relationship set is the degree of the relationship set. A binary relationship set is of degree 2; a ternary relationship set is of degree 3.

- ***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.
- **Single-valued** and **multivalued** attributes
- **Derived** attribute

- An attribute takes a **null** value when an entity does not have a value for it.
- The *null* value may indicate “not applicable” –that is, that the value does not exist for the entity.
- For example, one may have no middle name.
- *Null* can also designate that an attribute value is unknown.
- An unknown value may be either
- *Missing* (the value does exist, but we do not have that information) or
- *Not known* (we do not know whether or not the value actually exists).

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

- ***One-to-many:*** A 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.

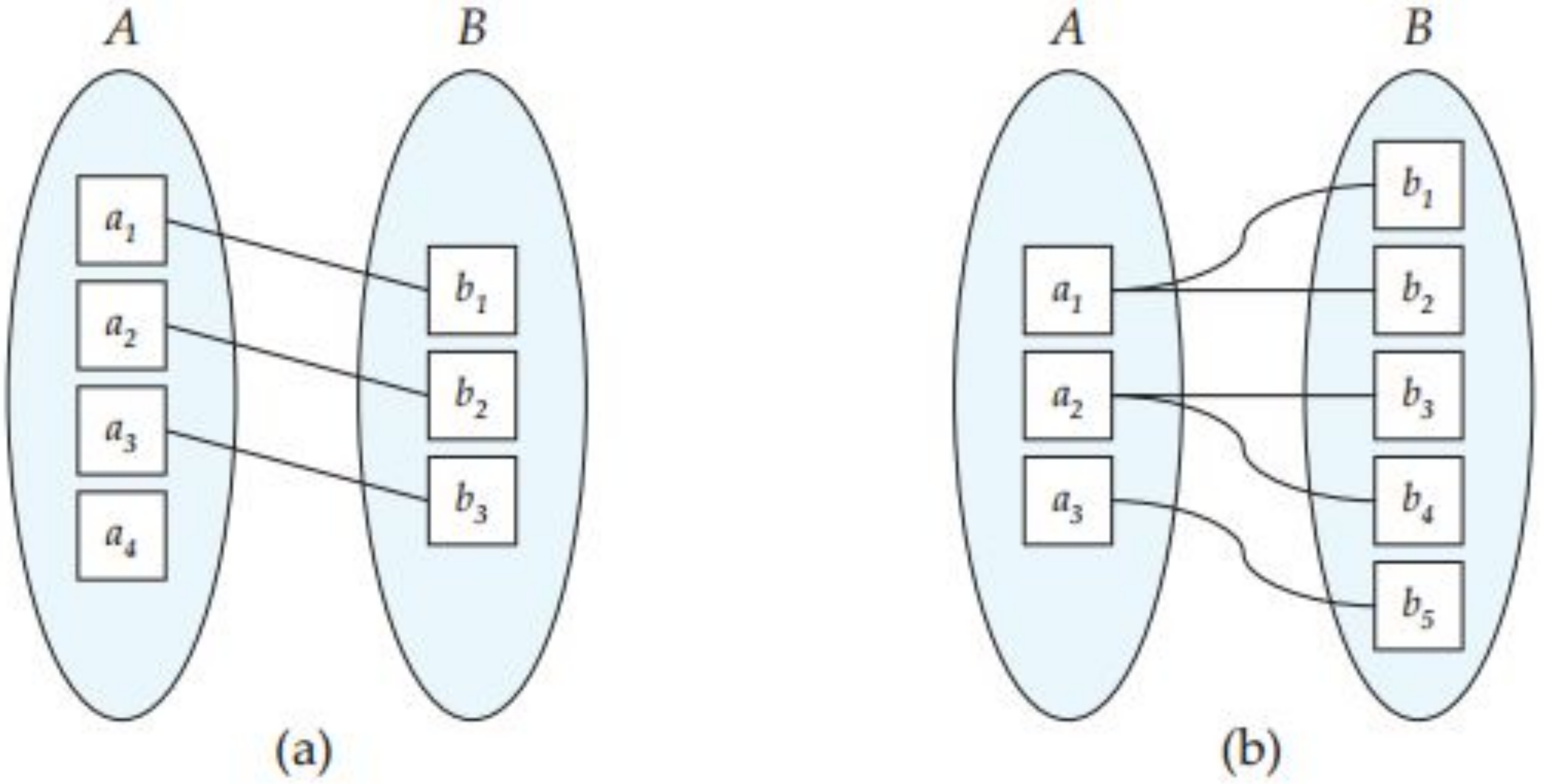


Figure 7.5 Mapping cardinalities. (a) One-to-one. (b) One-to-many.

- **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*.
- **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*.

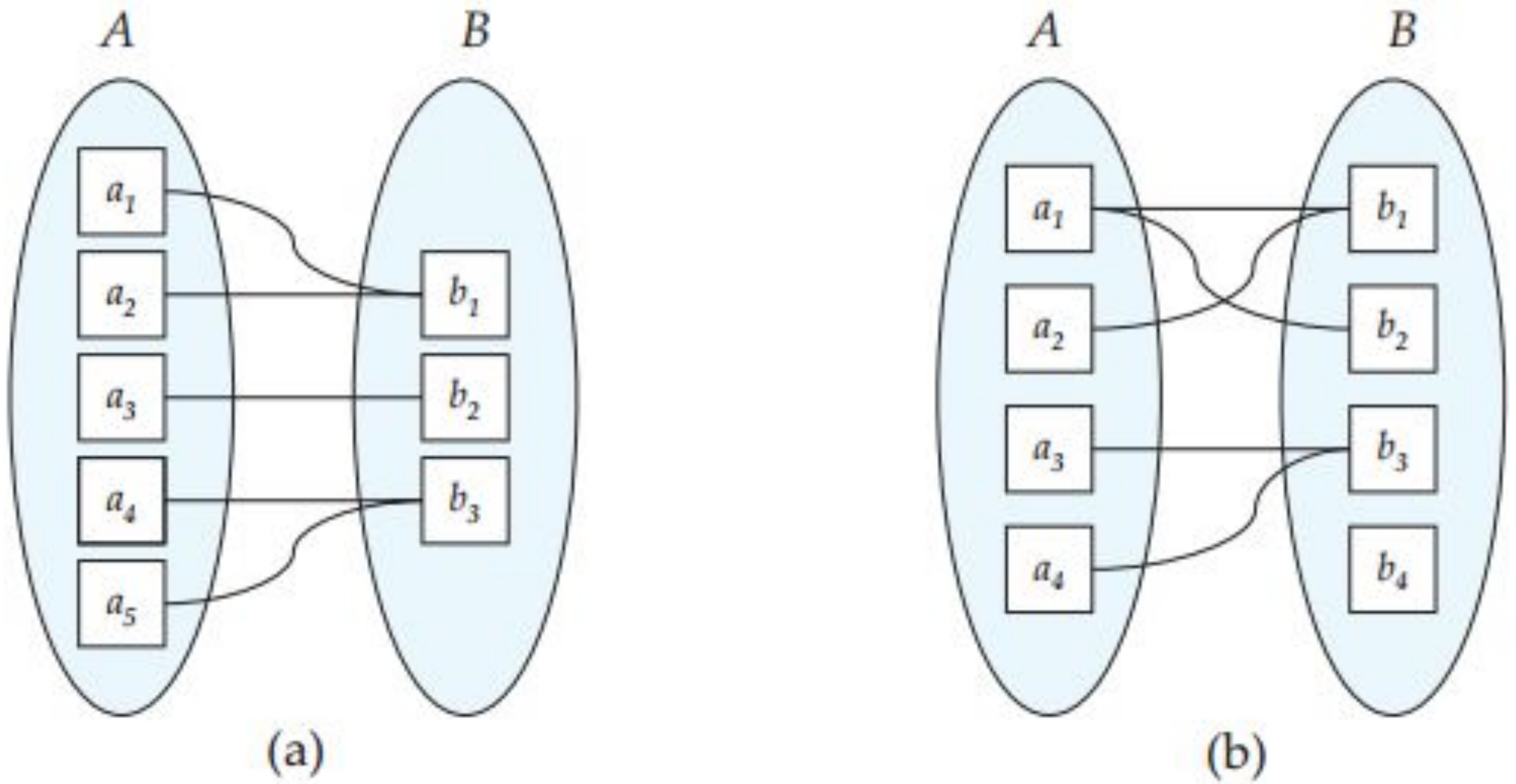


Figure 7.6 Mapping cardinalities. (a) Many-to-one. (b) Many-to-many.

- **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**.

- **Keys**
- We must have a way to specify how entities within a given entity set are distinguished.
- The differences among them must be expressed in terms of their attributes.
- Therefore, the values of the attribute values of an entity must be such that they can *uniquely identify* the entity.
- In other words, no two entities in an entity set are allowed to have exactly the same value for all attributes.
- A key for an entity is a set of attributes that suffice to distinguish entities from each other.

- Entity Relationship Diagrams
- **E-R diagram** can express the overall logical structure of a database graphically.

- **Basic Structure**
- An E-R diagram consists of the following major components:
- **Rectangles** which represent entity
- **Ellipse** which represent attributes
- **Diamonds** which represent relationship sets
- **Lines** link entity sets to relationship sets, attributes to entity set.

- **Double lines** indicates total participation of an entity in a relationship set
- **Double rectangles** which represent weak entity set.
- **Double ellipses** which represent multivalued attribute.
- **Dashed ellipses** which denotes derived attributes.

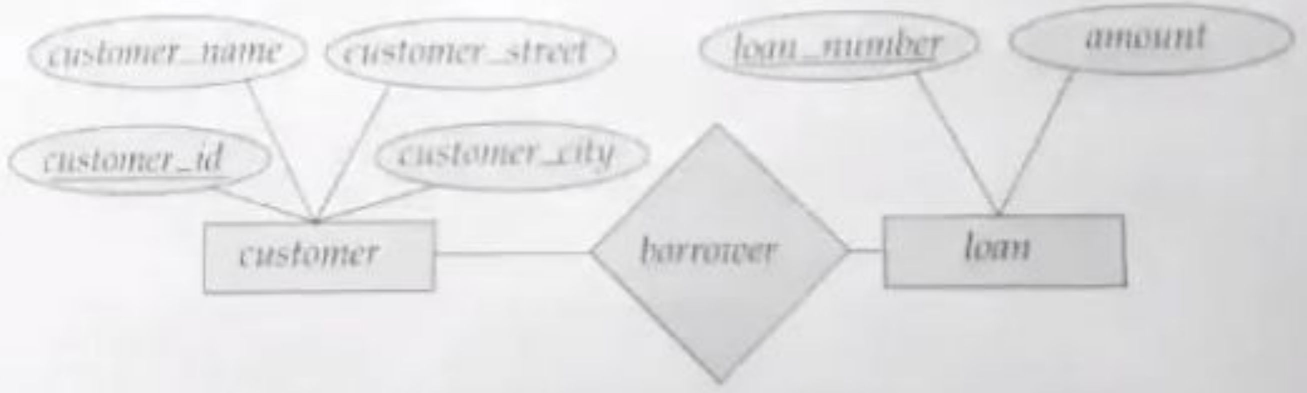


Figure 6.7 E-R diagram corresponding to customers and loans.

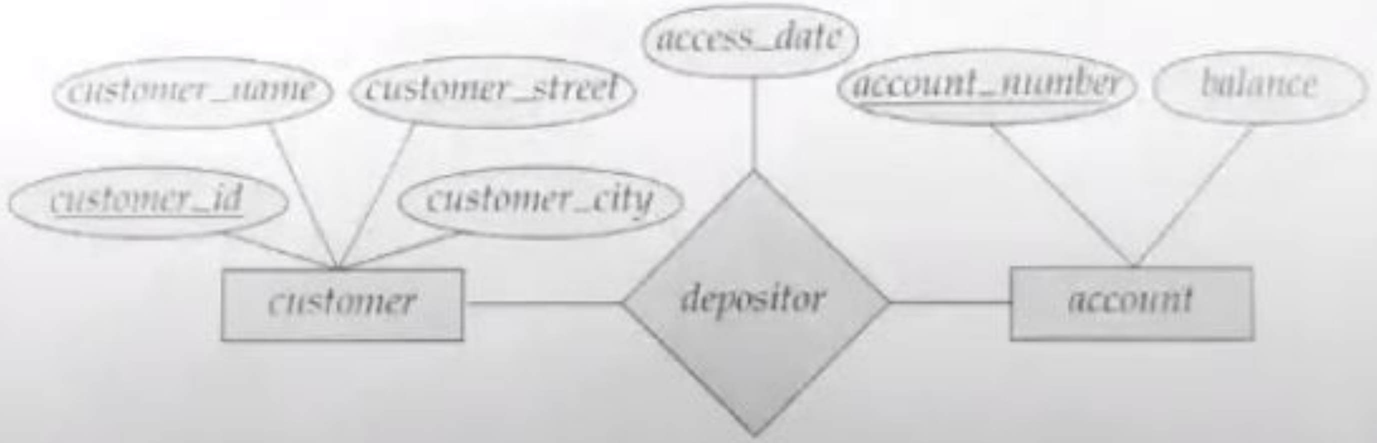


Figure 6.9 E-R diagram with an attribute attached to a relationship set.

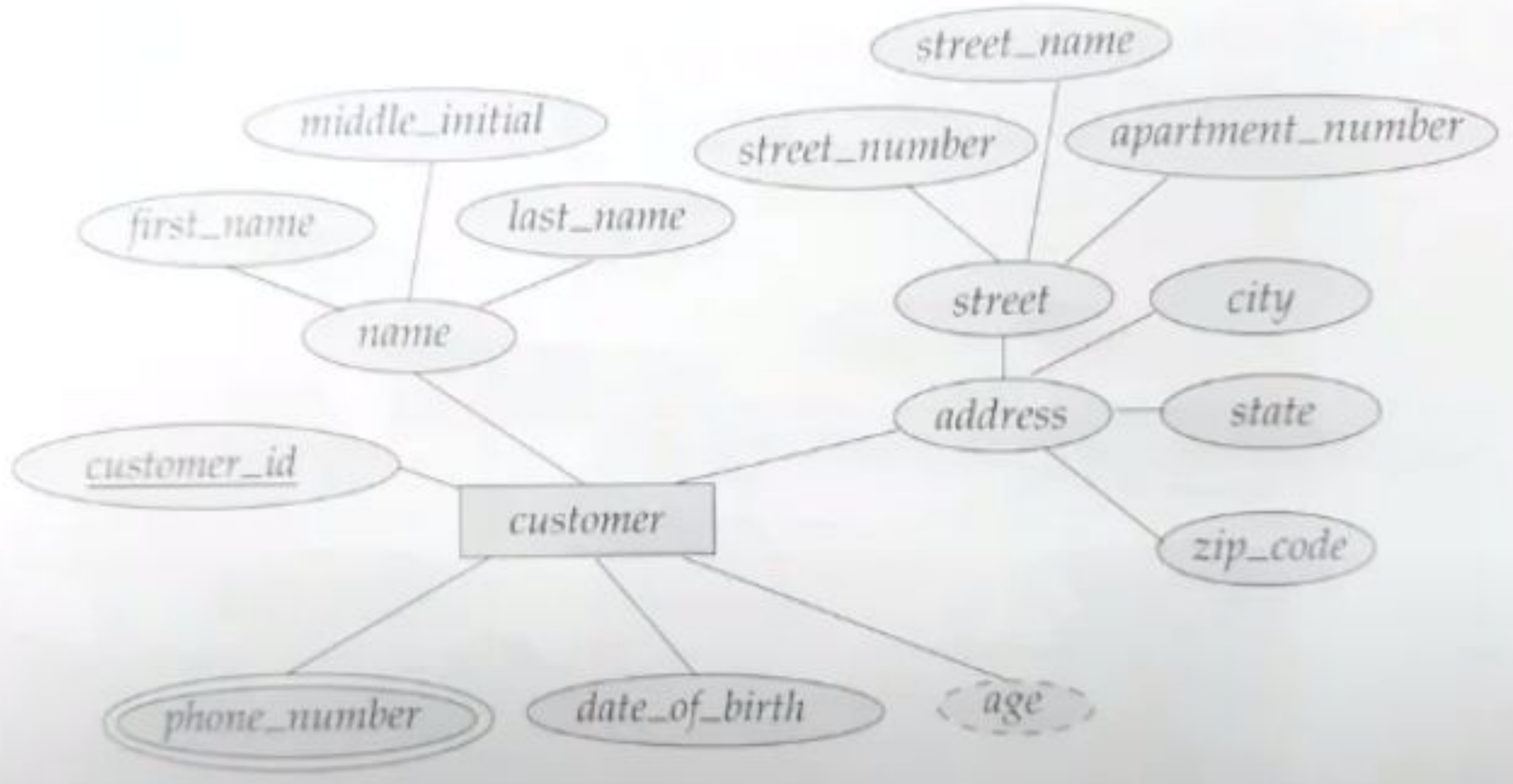


Figure 6.10 E-R diagram with composite, multivalued, and derived attributes.

Weak Entity Sets

- An entity set that does 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 a weak entity set to be meaningful, it must be associated with another entity set, called the **Identifying** or **Owner entity set**.
- The relationship associating the weak entity set with the identifying entity set is called the **identifying relationship**.

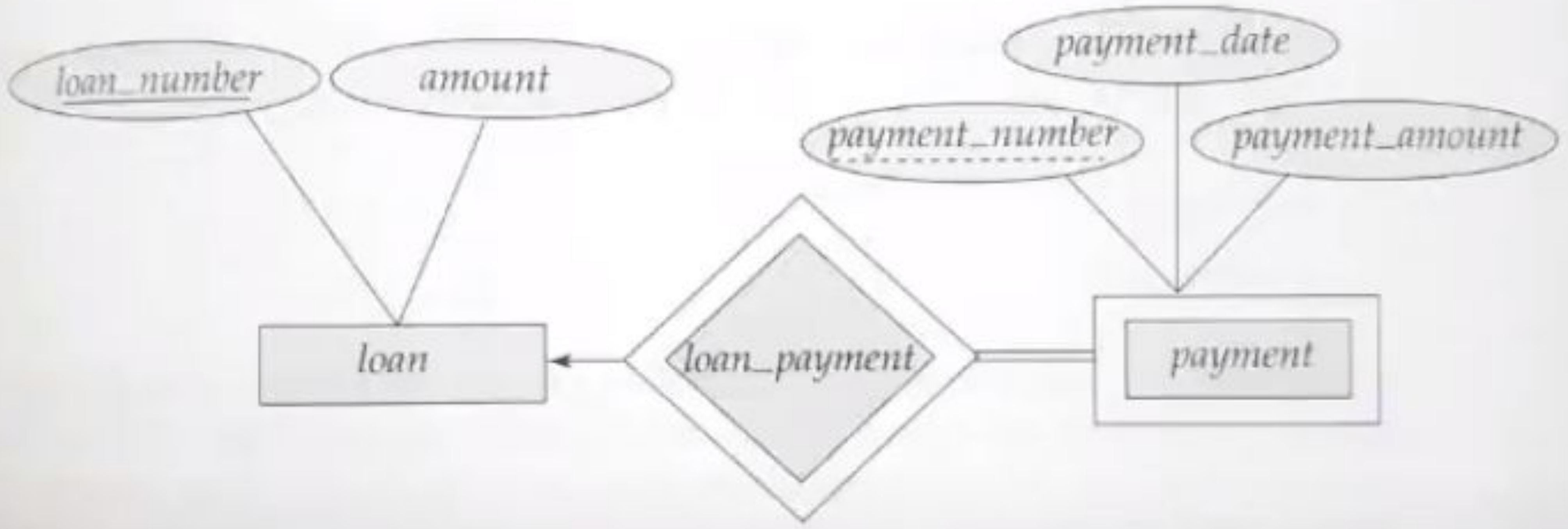
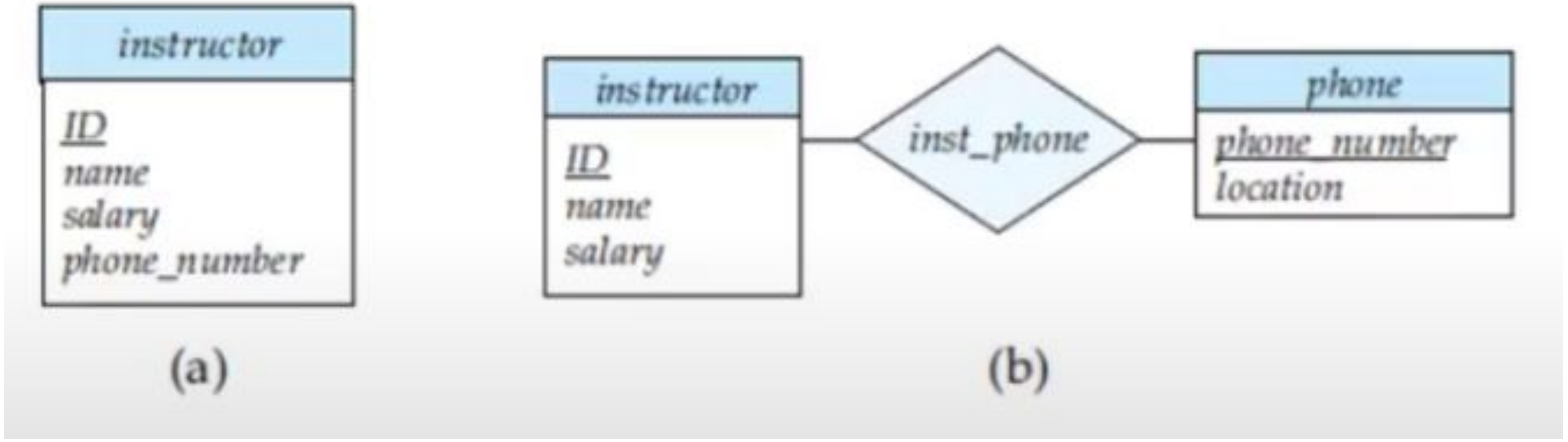


Figure 6.19 E-R diagram with a weak entity set.

Entity Relationship Design Issues

- *Use of Entity Set vs Attributes*



- Use of Entity Sets versus Relationship Sets

- It is not always clear weather an object is best expressed by an entity set or a relationship set.
- **Student** takes **Section**
- **Student** does **Registration** *for a* **Section**

12 Codd’s rules

- Dr Edgar F. Codd, came up with twelve rules of his own, which according to him, a database must obey in order to be regarded as a true relational database.
- These rules can be applied on any database system that manages stored data using only its relational capabilities. This is a foundation rule, which acts as a base for all the other rules.
- **Rule 1: Information Rule**
- The data stored in database, may it be user data or metadata, must be a value of some table cell. Everything in a database must be stored in a table format.

- **Rule 2: Guaranteed Access Rule**
- Every single data element *value* is guaranteed to be accessible logically with a combination of table-name, primary-key *rowvalue,* and attribute-name *columnvalue.*
- **Rule 3: Systematic Treatment of NULL Values**
- The NULL values in a database must be given a systematic and uniform treatment. This is very important rule because a NULL can be interpreted as one of the following – data is missing, data is not known, or data is not applicable.

- **Rule 4: Active Online Catalog**
- The structure description of the entire database must be stored in an online catalog, known as **data dictionary,** which can be accessed by authorized users. Users can use the same query language to access the catalog which they use to access the database itself.
- **Rule 5: Comprehensive Data Sub-Language Rule**
- A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations.

- This language can be used directly or by means of some application. If the database allows access to data without any help of this language, then it is considered as a violation.
- **Rule 6: View Updating Rule**
- All the views of a database, which can theoretically be updated, must also be updatable by the system.
- **Rule 7: High-Level Insert, Update and Delete Rule**
- A database must support high-level insertion, updation and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

- **Rule 8: Physical Data Independence**
- The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.
- **Rule 9: Logical Data Independence**
- The logical data in a database must be independent of its user’s view *application*. Any change in logical data must not affect the applications using it. For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application. This is one of the most difficult rule to apply.

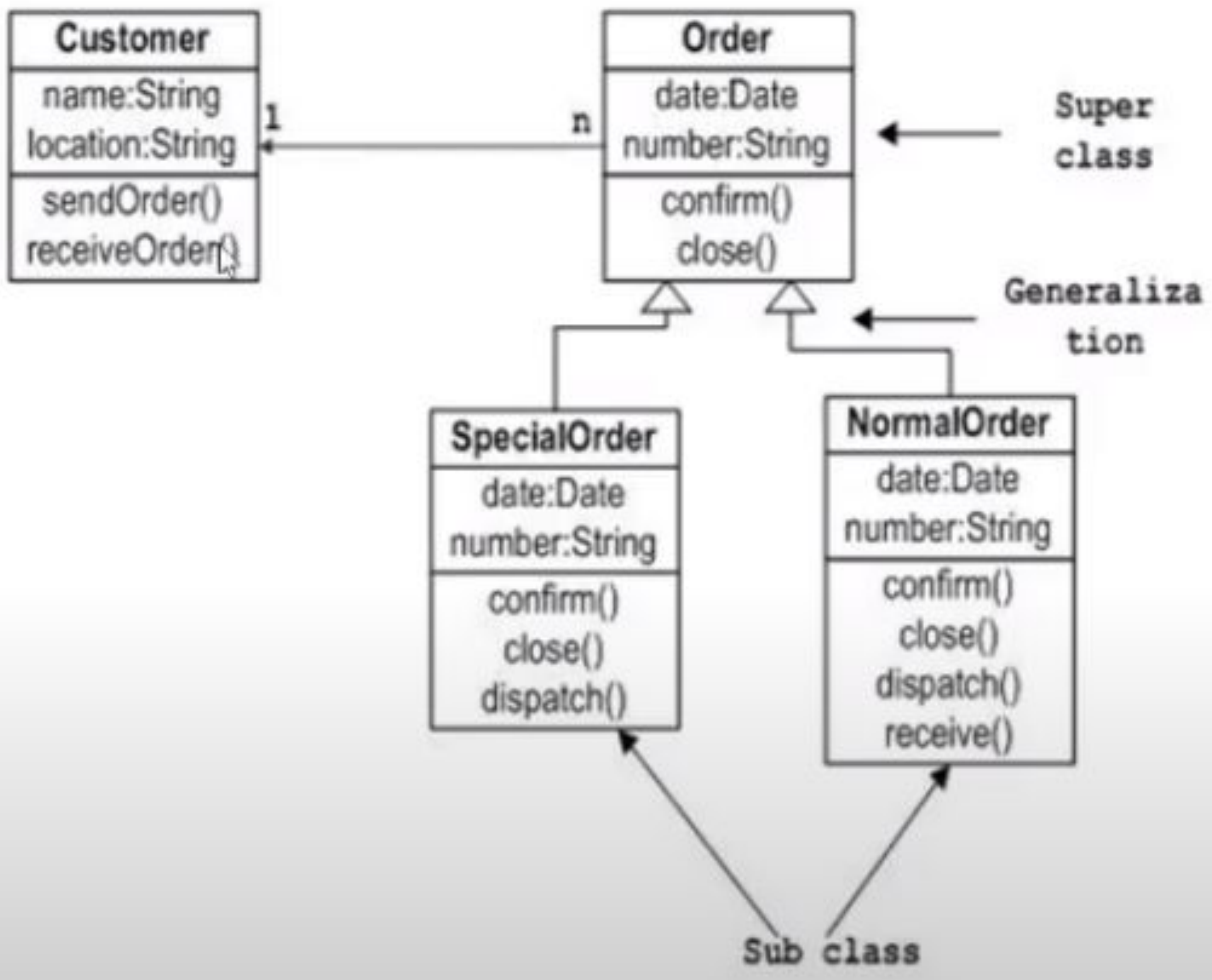
- **Rule 10: Integrity Independence**
- A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.
- **Rule 11: Distribution Independence**
- The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. This rule has been regarded as the foundation of distributed database systems.

- **Rule 12: Non-Subversion Rule**
- if a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.

- **The Unified Modelling Language UML**
- Entity-relationship diagram help model the data representation component of a software system. Other components include models of user interactions with the system, specification of foundational modules of the system and their interaction, etc.
- The **Unified Modeling Language (UML)** is a standard developed under the auspices of the Object Management Group (OMG) for creating specifications of various components of a software system

- Some of the parts of UML are:
- **Class diagram**. A class diagram is similar to an E-R diagram.
- **Use case diagram**. Use case diagrams show the interaction between users and the system, in particular the steps of tasks that users perform (such as withdrawing money or registering for a course).
- **Activity diagram**. Activity diagrams depict the flow of tasks between various components of a system.

Sample Class Diagram



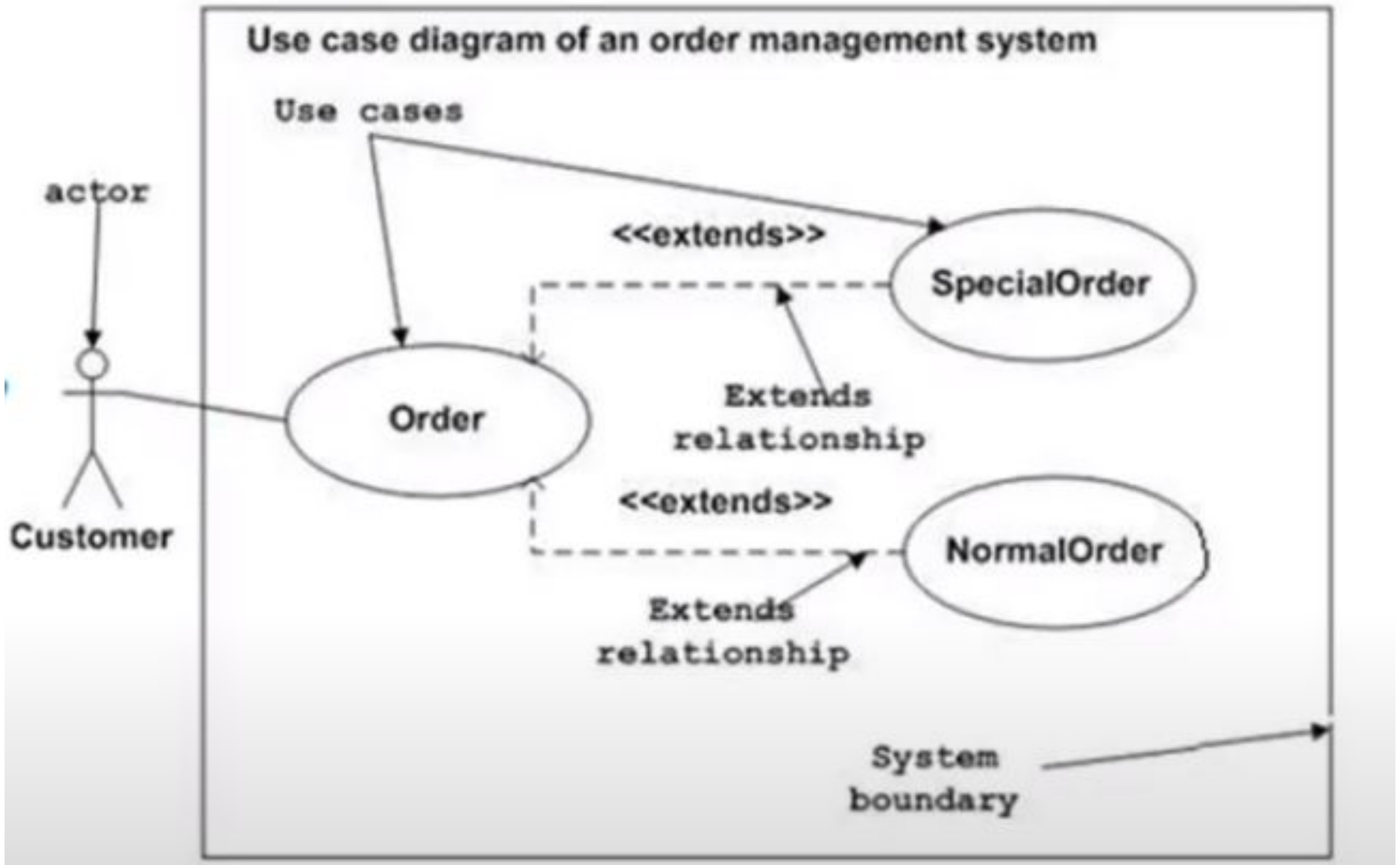
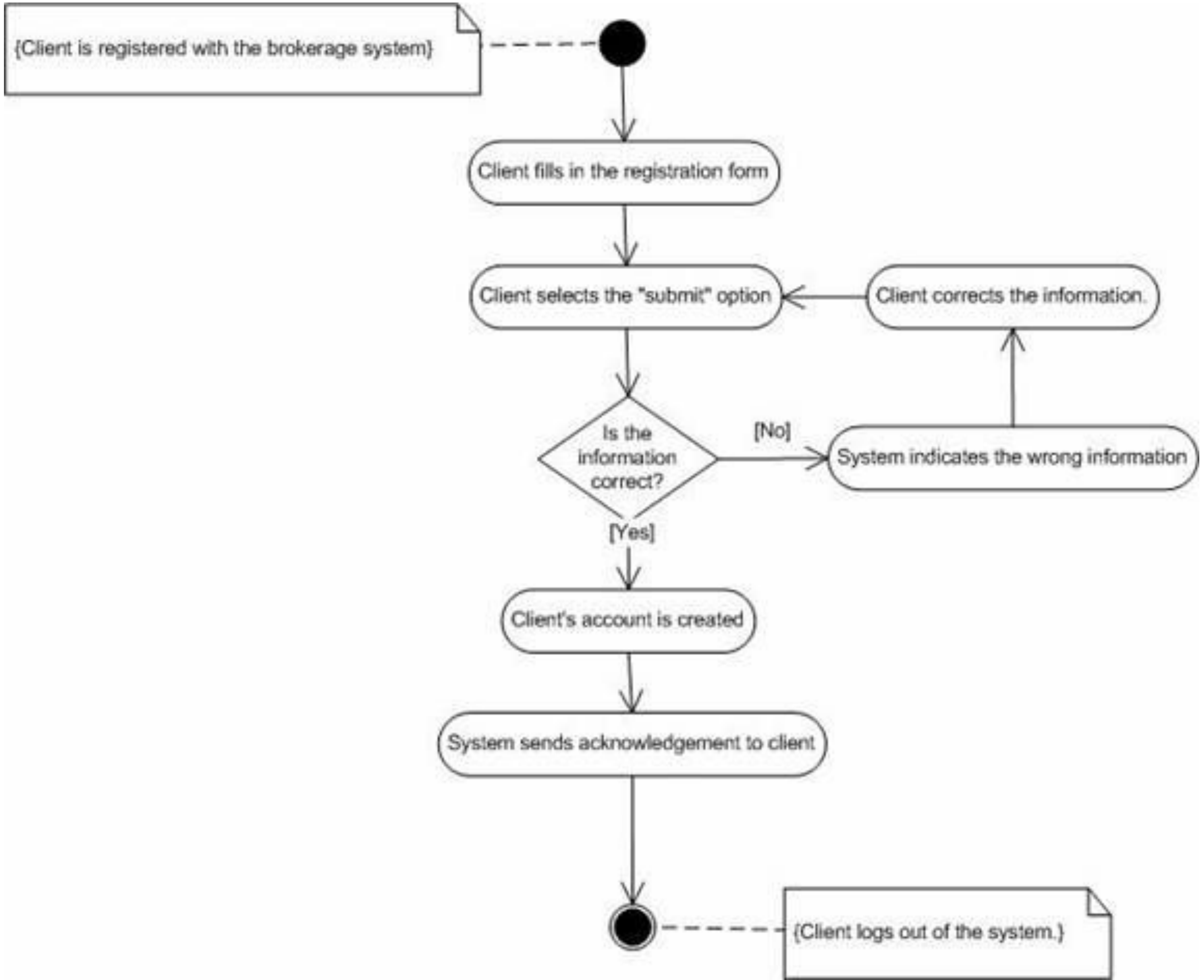
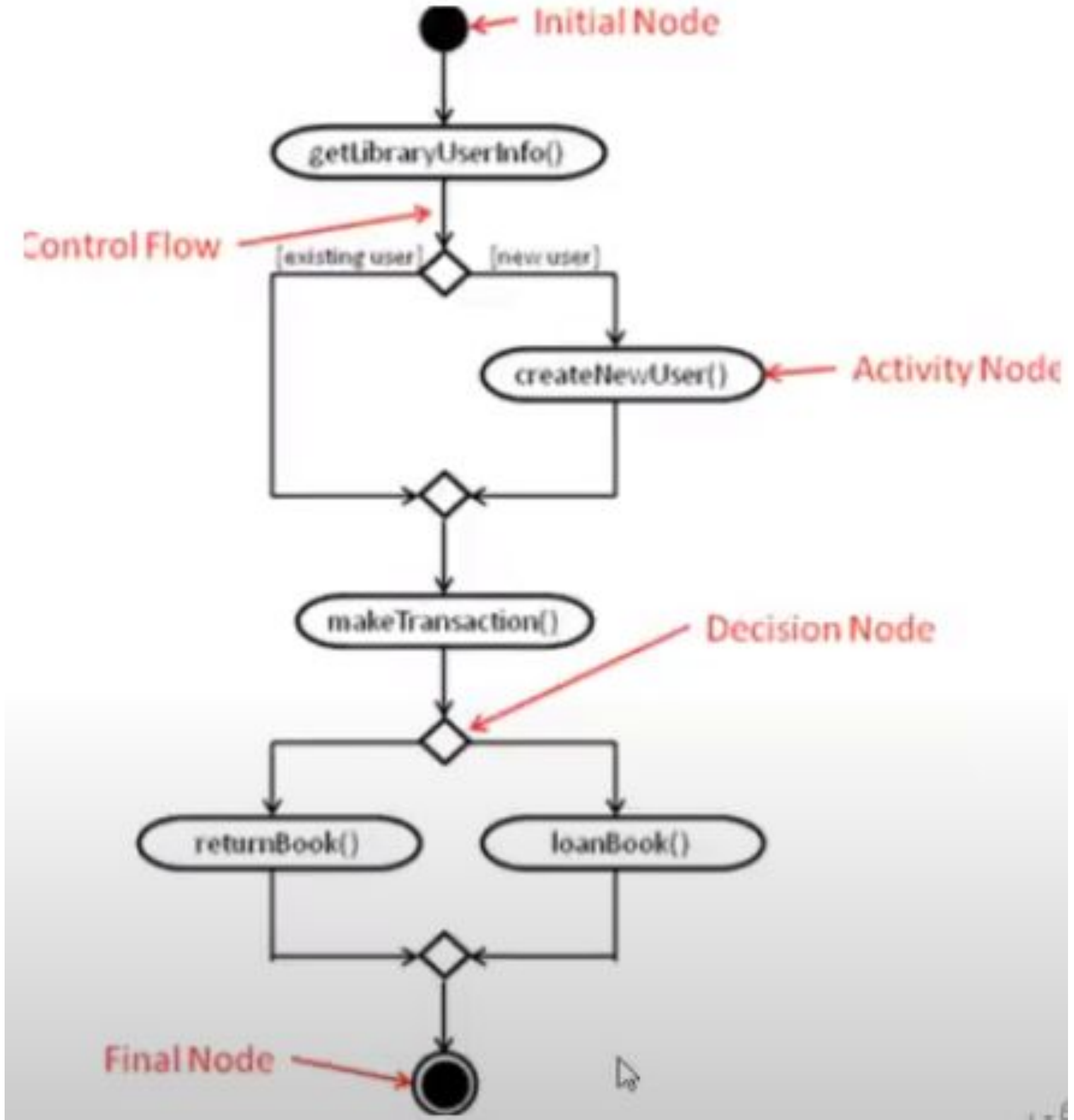


Figure: Sample Use Case diagram

Activity Diagram


{Client is registered with the brokerage system}
Client fills in the registration form
Client selects the "submit" option
Client corrects the information.
Is the information correct?
[No]
System indicates the wrong information
[Yes]
Client's account is created
System sends acknowledgement to client
{Client logs out of the system.}

Activity Diagram


Initial Node
getLibraryUserInfo()
Control Flow
[existing user]
[new user]
createNewUser()
Activity Node
makeTransaction()
Decision Node
returnBook()
loanBook()
Final Node