ADBMS

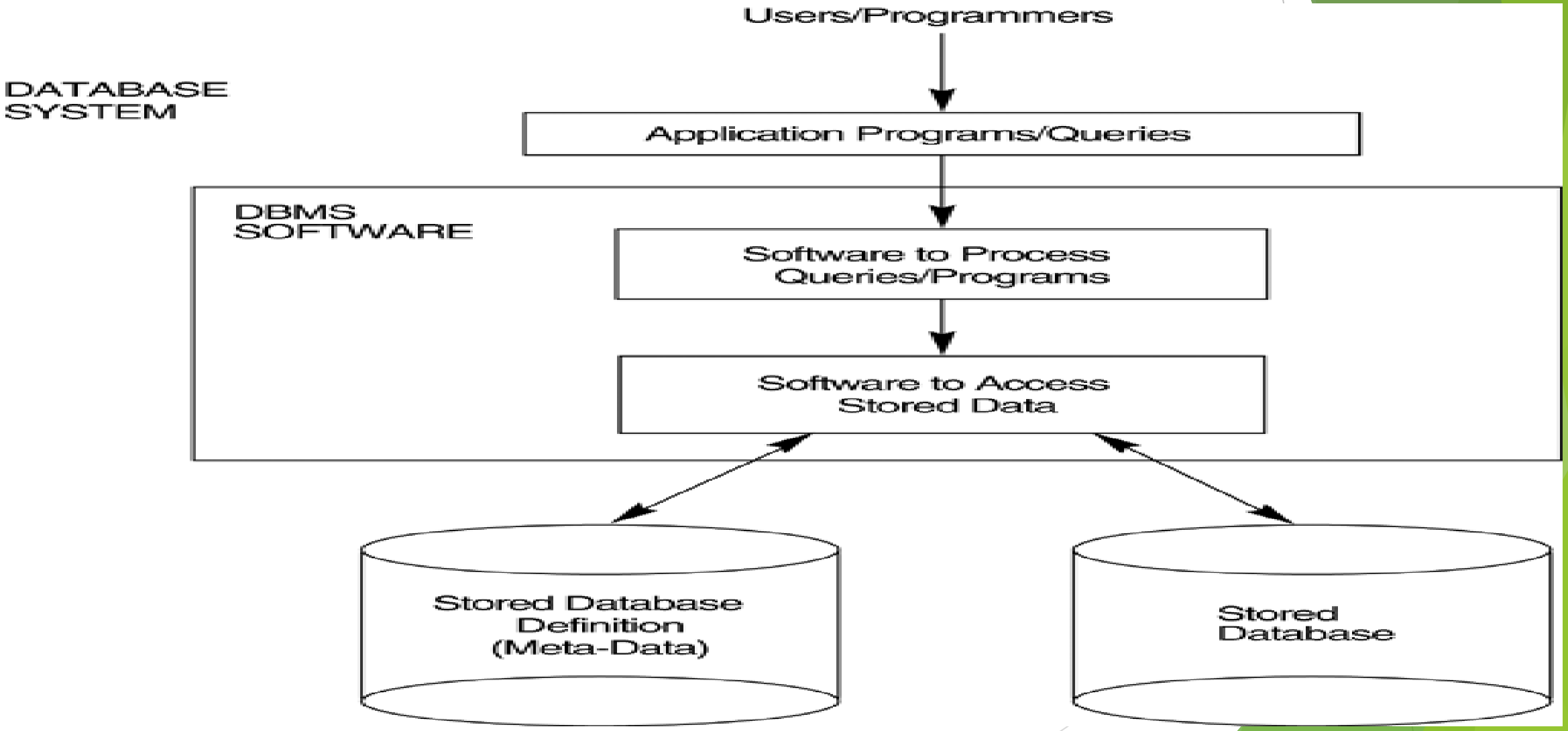
Database& DBMS

- Collection of related data that represents real world entities
- Entities:Employee,Student,Teacher,Colleges,Bank Account
- A **database-management system (DBMS) is a collection of interrelated data and** a set of programs to access those data
- DBMS is a collection of programs that enable users to create and maintain a database
- The DBMS is hence a general purpose software system that facilitate the process of defining,constructing , manipulating and sharing databases among the various users and applications

STUDENT	Name	StudentNumber	Class	Major
	Smith	17	1	CS
	Brown	8	2	CS

COURSE	CourseName	CourseNumber	CreditHours	Department
	Intro to Computer Science	CS1310	4	CS
	Data Structures	CS3320	4	CS
	Discrete Mathematics	MATH2410	3	MATH
	Database	CS3380	3	CS

Database System Enviornment



Database-System Applications

1 Enterprise Information

- *Sales: For customer, product, and purchase information.*
- *Accounting: For payments, receipts, account balances, assets and other* accounting information.
- *Human resources: For information about employees, salaries, payroll taxes,* and benefits, and for generation of paychecks.
- *Manufacturing: For management of the supply chain and for tracking production* of items in factories, inventories of items in warehouses and stores, and orders for items.
- *Online retailers: For sales data noted above plus online order tracking,*generation of recommendation lists, and maintenance of online product evaluations

Database-System Applications

2 Banking and Finance

- *Banking: For customer information, accounts, loans, and banking transactions.*
- *Credit card transactions: For purchases on credit cards and generation of* monthly statements.
- *Finance: For storing information about holdings, sales, and purchases of* financial instruments such as stocks and bonds; also for storing real-time market data to enable online trading by customers and automated trading by the firm.

3 Universities: For student information, course registrations, and grades (in addition to standard enterprise information such as human resources and accounting).

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

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

Purpose of Database-Systems

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 over a long period, the various files are likely to have different structures and the programs may be written in several programming languages

  Moreover, the same information may be duplicated in several places (files)

- For example, if a student has a double major (say, music and mathematics) the address and telephone number of that student may appear in a file that consists of student records of students in the Music department and in a file that consists of student records of students in the Mathematics department.

- This redundancy leads to higher storage and access cost and it may lead to **data inconsistency**

Purpose of Database-Systems

- For example, a changed student address may be reflected in the Music department records but not elsewhere in the system.
- **Difficulty in accessing data:** Conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner. More responsive data-retrieval systems are required for general use.
- **Data isolation. Because data 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. Suppose the university maintains an** account for each department, and records the balance amount in each account. Suppose also that the university requires that the account balance of a department may never fall below zero. Developers enforce these constraints in the system by adding appropriate code in the various application programs.
- However, when new constraints are added, it is difficult to change the programs to enforce them. The problem is compounded when constraints involve several data items from different files.

Purpose of Database-Systems

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

Purpose of Database-Systems

- **Concurrent-access anomalies:** Many systems allow multiple users to update the data simultaneously
- In such an environment, interaction of concurrent updates is possible and may result in inconsistent data.
- Suppose a registration program maintains a count of students registered for a course, in order to enforce limits on the number of students registered.
- When a student registers, the program reads the current count for the courses, verifies that the count is not already at the limit, adds one to the count, and stores the count back in the database
- Suppose two students register concurrently, with the count at (say) 39. The two program executions may both read the value 39, and both would then write back 40, leading to an incorrect increase of only 1, even though two students successfully registered for the course and the count should be 41.
- Furthermore, suppose the course registration limit was 40; in the above case both students would be able to register, leading to a violation of the limit of 40 students

Purpose of Database-Systems

- **Security problems. Not every user of the database system should be able** to access all the data.
- For example, in a university, payroll personnel need to see only that part of the database that has financial information. They do not need access to information about academic records. But, since application programs are added to the file-processing system in an ad hoc manner, enforcing such security constraints is difficult

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 system is to provide users with an *abstract view of the data. That is, the system* hides certain details of how the data are stored and maintained.

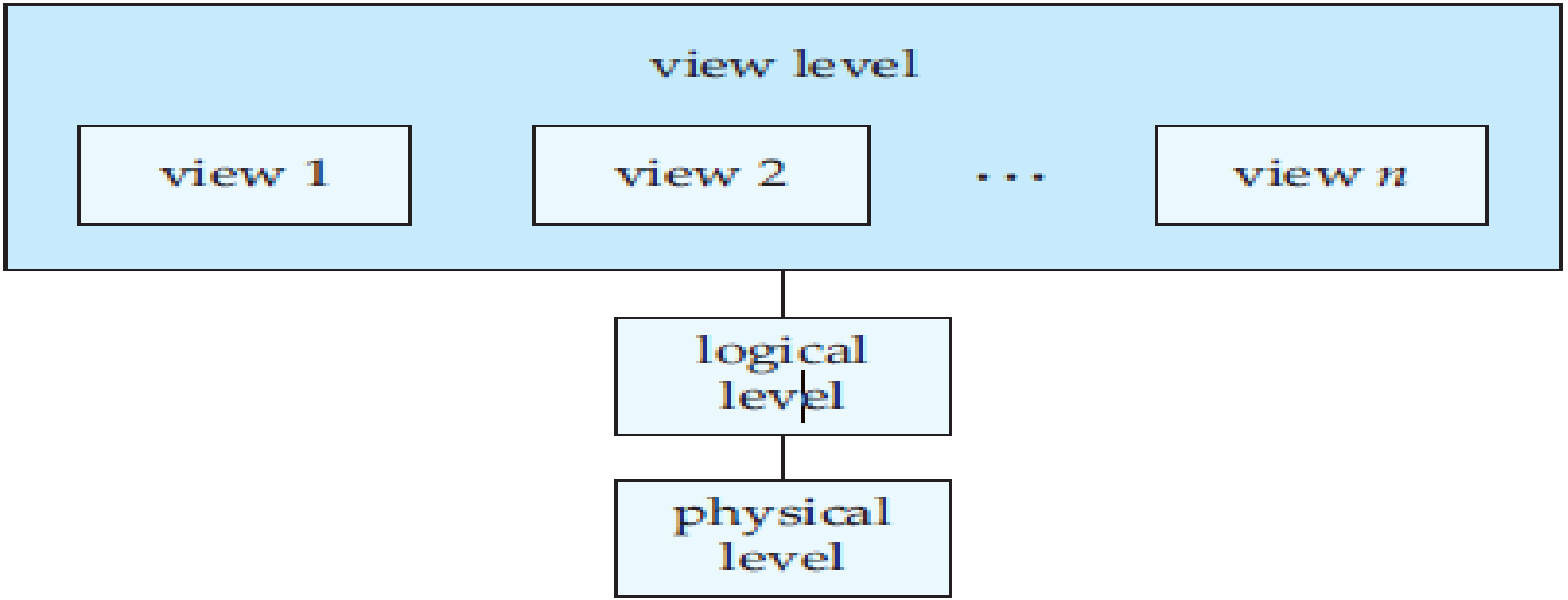
Data Abstraction

- 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:

View of Data

- **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.
- This view describes the part of database that a particular user group is interested in and hides the rest of the database from the user group

View of Data



View of Data

For example, Consider a record

type *instructor = record*

ID : ***char (5);***

name : ***char (20);***

dept name : ***char (20);***

salary : ***numeric (8,2);***

end;

- This code defines a new record type called *instructor with four fields. Each field* has a name and a type associated with it

View of Data

- At the physical level, an *instructor record can be described* as a block of consecutive storage locations.
- The compiler hides this level of detail from programmers. Similarly, the database system hides many of the lowest-level storage details from database programmers
- At the logical level, each such record is described by a type definition, and the interrelationship of these record types is defined as well.
- Programmers using a programming language work at this level of abstraction
- Finally, at the view level, computer users see a set of application programs that hide details of the data types.
- At the view level, several views of the database are defined, and a database user sees some of these views.
- The view level also provide a security mechanism to prevent users from accessing certain parts of the database

Database Instances and Schemas

- The collection of information stored in the database at a particular moment is called an **instance** of the database.
- The overall design of the database is called the database **schema.**
- A database schema corresponds to the variable declarations (along with associated type definitions) in a program.
- Each variable has a particular value at a given instant. The values of the variables in a program at a point in time correspond to an *instance of a database schema*
- 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 sub schemas, that describe different views of the database.

Data Independence

- Applications insulated from how data is structured and stored
- Capacity to change schema at one level of database without having to change schema at next higher level

Logical Data Independence

- The capacity to change the logical schema without having to change external schemas or application program
- We can change the logical schema to expand the database ,to change the constraints or to reduce database

Physical Data Independence

- The capacity to change the physical schema without having to change the logical 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 Models

- **Data Model:** 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 the physical, logical, and view levels.
- **1) Relational Data Model:** This type of model designs the data in the form of rows and columns within a table. Thus, a relational model uses tables for representing data and in-between relationships. Tables are also called relations. This model was initially described by Edgar F. Cod, in 1969. The relational data model is the widely used model which is primarily used by commercial data processing applications.
- **2) Entity-Relationship Data Model:** An ER model is the logical representation of data as objects and relationships among them. These objects are known as entities, and relationship is an association among these entities. It was widely used in database designing. A set of attributes describe the entities. For example, student_name, student_id describes the 'student' entity. A set of the same type of entities is known as an 'Entity set', and the set of the same type of relationships is known as 'relationship set'.

Data Models

- **Data Model:** 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 the physical, logical, and view levels.
- **1) Relational Data Model:** This type of model designs the data in the form of rows and columns within a table. Thus, a relational model uses tables for representing data and in-between relationships. Tables are also called relations. This model was initially described by Edgar F. Cod, in 1969. The relational data model is the widely used model which is primarily used by commercial data processing applications.
- **2) Entity-Relationship Data Model:** An ER model is the logical representation of data as objects and relationships among them. These objects are known as entities, and relationship is an association among these entities. It was widely used in database designing. A set of attributes describe the entities. For example, student_name, student_id describes the 'student' entity. A set of the same type of entities is known as an 'Entity set', and the set of the same type of relationships is known as 'relationship set'.

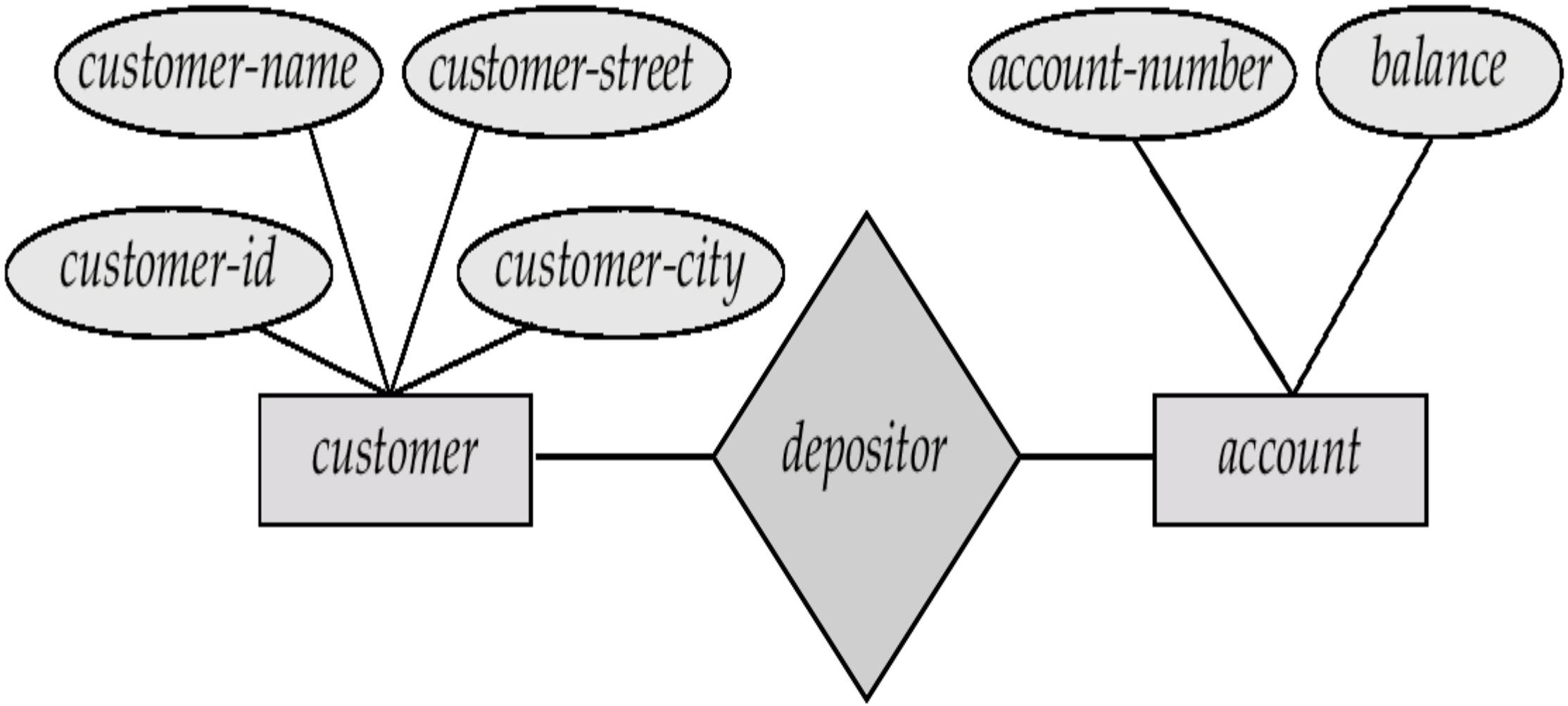
customer-id	*customer-name*	*customer-street*	*customer-city*
192-83-7465	Johnson	12 Alma St.	Palo Alto
019-28-3746	Smith	4 North St.	Rye
677-89-9011	Hayes	3 Main St.	Harrison
182-73-6091	Turner	123 Putnam Ave.	Stamford
321-12-3123	Jones	100 Main St.	Harrison
336-66-9999	Lindsay	175 Park Ave.	Pittsfield
019-28-3746	Smith	72 North St.	Rye

(a) The *customer* table

account-number	*balance*
A-101	500
A-215	700
A-102	400
A-305	350
A-201	900
A-217	750
A-222	700

(b) The *account* table

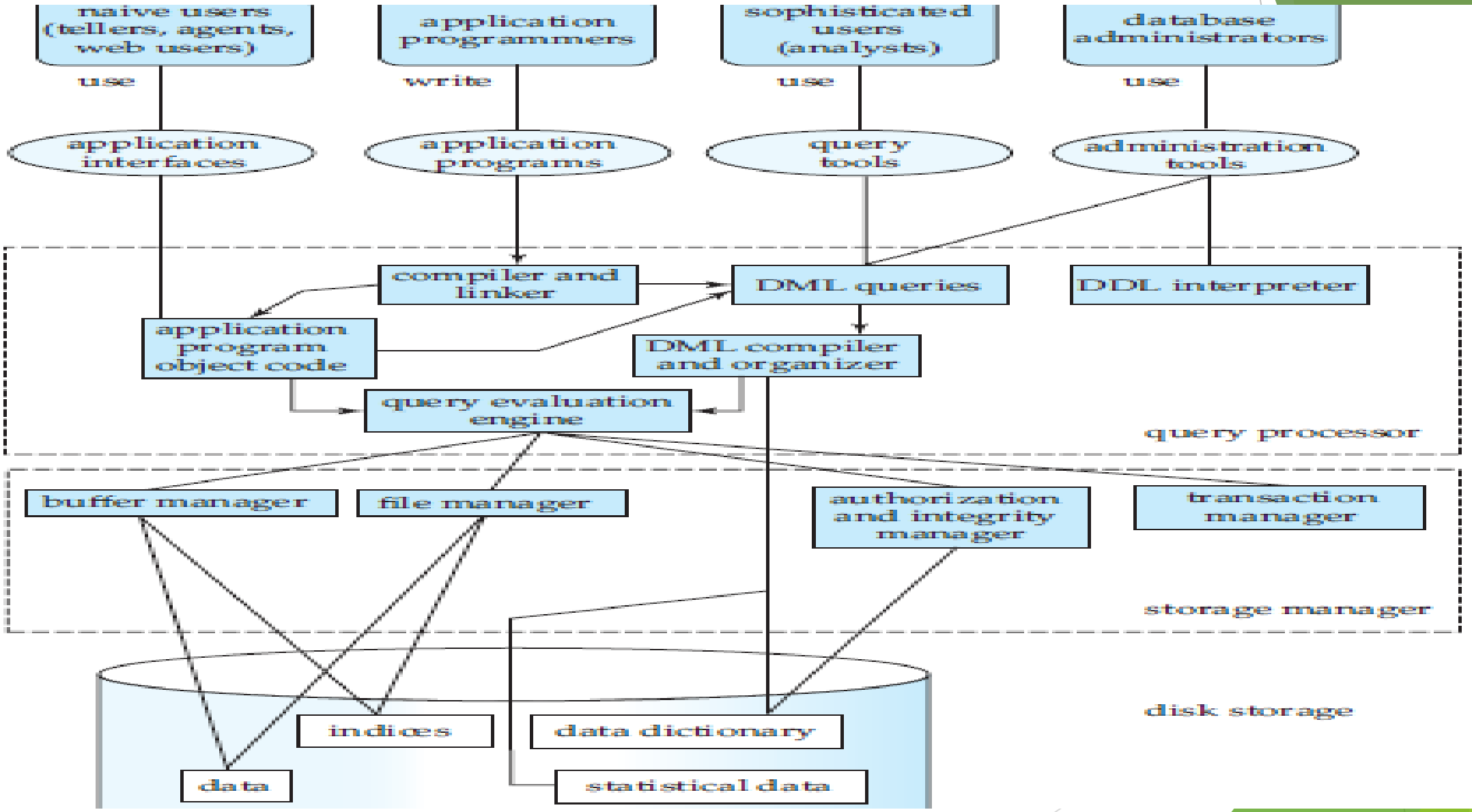
customer-id	*account-number*
192-83-7465	A-101
192-83-7465	A-201
019-28-3746	A-215
677-89-9011	A-102
182-73-6091	A-305
321-12-3123	A-217
336-66-9999	A-222
019-28-3746	A-201


customer-name
customer-street
account-number
balance
customer-id
customer-city
customer
depositor
account

Data Models

- **3) Object-based Data Model:** An extension of the ER model with notions of functions, encapsulation, and object identity, as well. This model supports a rich type system that includes structured and collection types. Thus, in 1980s, various database systems following the object-oriented approach were developed. Here, the objects are nothing but the data carrying its properties.
- **4) Semistructured Data Model:** This type of data model is different from the other three data models (explained above). The semistructured data model allows the data specifications at places where the individual data items of the same type may have different attributes sets. The Extensible Markup Language, also known as XML, is widely used for representing the semistructured data. Although XML was initially designed for including the markup information to the text document, it gains importance because of its application in the exchange of data.

Database Architecture



Database Architecture

- **Storage Manager**
- The *storage manager is the component of a database system 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 raw data are stored on the disk using the file system provided by the operating system
- 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

Database Architecture

Storage Manager

- 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 buffer manager enables the database to handle data sizes that are much larger than the size of main memory.

Database Architecture

Storage Manager

- 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 can provide fast access to data items.**

Database Architecture

- **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;**
- **Query evaluation engine, which executes low-level instructions generated** by the DML compiler

Database Architecture

Transaction Manager

- A **transaction** is a collection of operations that performs a single logical function in a database application
- **Transaction-management component** ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.
- **Concurrency-control manager** controls the interaction among the concurrent transactions, to ensure the consistency
- The **transaction manager consists of the concurrency-control** manager and the recovery manager
- It is the responsibility of the **concurrency-control manager to control** the interaction among the concurrent transactions, to ensure the consistency of the database.
- Ensuring the atomicity and durability properties is the responsibility of the **recovery manager**

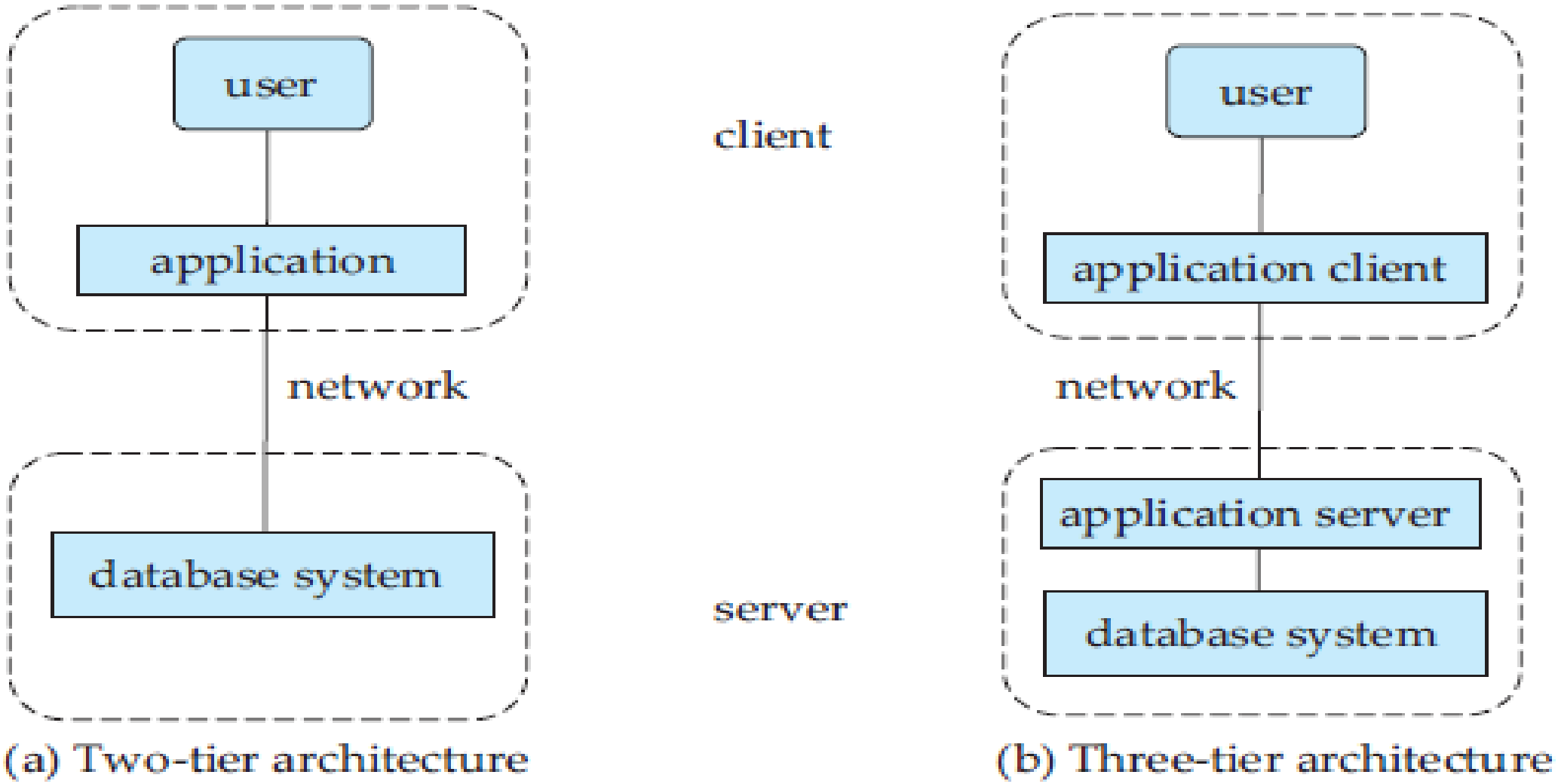


Figure 1.6 Two-tier and three-tier architectures.

Two Tier and Three Tier Architecture

- In a **two-tier architecture, the application resides at the client machine,** where it invokes database system functionality at the server machine through query language statements.
- Application program interface standards like ODBC and JDBC are used for interaction between the client and the server
- In a **three-tier architecture, the client machine acts as 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

Database Users and User Interfaces

There are four different types of database-system users

1. **Native users** are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously.

- For example, a clerk in the university who needs to add a new instructor to department A invokes a program called new hire. This program asks the clerk for the name of the new instructor, new ID, the name of the department (that is, A), and the salary
- The typical user interface for native users is a forms interface, where the user can fill in appropriate fields of the form.
- Native users may also simply read reports generated from the database.

2. **Application programmers** are computer professionals who write application programs.

- Application programmers can choose from many tools to develop user interfaces.
- **Rapid application development (RAD)** tools are tools that enable an application programmer to construct forms and reports with minimal programming effort.

Database Users and User Interfaces

3. Sophisticated users interact with the system without writing programs.

- Instead, they form their requests either using a database query language or by using tools such as data analysis software.
- Analysts who submit queries to explore data in the database fall in this category

4. Specialized users are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework.

- Among these applications are computer-aided design systems, knowledgebase and expert systems, systems that store data with complex data types (for example, graphics data and audio data), and environment-modeling systems.

Database Administrator

- One of the main reasons for using DBMSs is to have central control of both the data and the programs that access those data.
- A person who has such central control over the system is called a **database administrator (DBA).**
- The functions of a DBA include:
- **Schema definition.** The DBA creates the original database schema by executing a set of data definition statements in the DDL.
- **Storage structure and access-method definition.**
- **Schema and physical-organization modification.** The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical organization to improve performance.

Database Administrator

- **Granting of authorization for data access. By granting different types of** authorization, the database administrator can regulate which parts of the database various users can access.
- The authorization information is kept in a special system structure that the database system consults whenever someone attempts to access the data in the system.
- **Routine maintenance. Examples of the database administrator's routine** maintenance activities are:

◦ Periodically backing up the database, either onto tapes or onto remote servers, to prevent loss of data in case of disasters such as flooding.

◦ Ensuring that enough free disk space is available for normal operations, and upgrading disk space as required.

◦ Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.

Structure of Relational Databases

- A relational database consists of a collection of **tables, each of which is assigned a** unique name
- Consider the instructor table , which stores information about instructors. The table has four column headers: ID, name, dept name, and salary.
- Each row of this table records information about an instructor, consisting of the instructor's ID, name, dept name, and salary
- In the relational model the term relation is used to refer to a table, while the term tuple is used to refer to a row.
- Similarly, the term attribute refers to a column of a table.

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

Figure 2.1 The *instructor* relation.

Structure of Relational Databases

- For each attribute of a relation, there is a set of permitted values, called the **domain** of that attribute.
- Thus, the domain of the salary attribute of the instructor relation is the set of all possible salary values, while the domain of the name attribute is the set of all possible instructor names
- A domain is **atomic** if elements of the domain are considered to be indivisible units.
- The **null** value is a special value that signifies that the value is unknown or does not exist

Keys in DBMS

- Key is either single column or attribute or a group of columns that uniquely identify rows or tuples in a table
- Keys ensure that there are no rows with duplicate information
- Keys also help in establishing a relationship between multiple tables in the database
- Following types of keys are in DBMS
- Super Key
- Candidate Key
- Primary Key
- Foreign Key

Keys

- A **super key is a set of one or more attributes that, taken collectively, allow us** to identify uniquely a tuple in the relation
- The given table has following super keys. All of the following sets of super key are able to uniquely identify a row of the employee table.
- {Id,Name}
- {Id,Email}
- {Id,Name,Email}
- {Id}

Id	Name	Gender	City	Mail
1	ABC	M	EKM	
2	XYZ	M	EKM	
3	EFG	F	TVM	

Keys

- Super key for which no proper subset is a super key .
- Such minimal super key with no redundant attributes are called candidate key.
- It is possible that several distinct sets of attributes could serve as a candidate key
- The following two set of super keys are chosen from the above sets as there are no redundant attributes in these sets
- {Id}
- {Mail}

Id	Name	Gender	City	Mail
1	ABC	M	EKM	
2	XYZ	M	EKM	
3	EFG	F	TVM	

Keys

- Primary key is the candidate key selected by the database designer to uniquely identify tuples in a table
- Out of all the candidate keys that can be possible for a table, there can be only one key that will be used to retrieve unique tuples from the table
- This candidate key is called the Primary Key

Id	Name	Gender	City	Mail
1	ABC	M	EKM	
2	XYZ	M	EKM	
3	EFG	F	TVM	

Keys

- **Foreign Key** is an attribute which is a primary key in its parent table,but it is included as attribute in another table
- A relation schema r1 may include attribute that is the primary key of another relation schema r2
- This attribute is called r1 referencing r2
- r1 is called referencing relation and r2 is called referenced relation

Eid	Name	Gender	City	Did
1	ABC	M	EKM	1
2	XYZ	M	EKM	2

Did	Name	Location
1	Marketing	Delhi
2	Finance	Bhopal

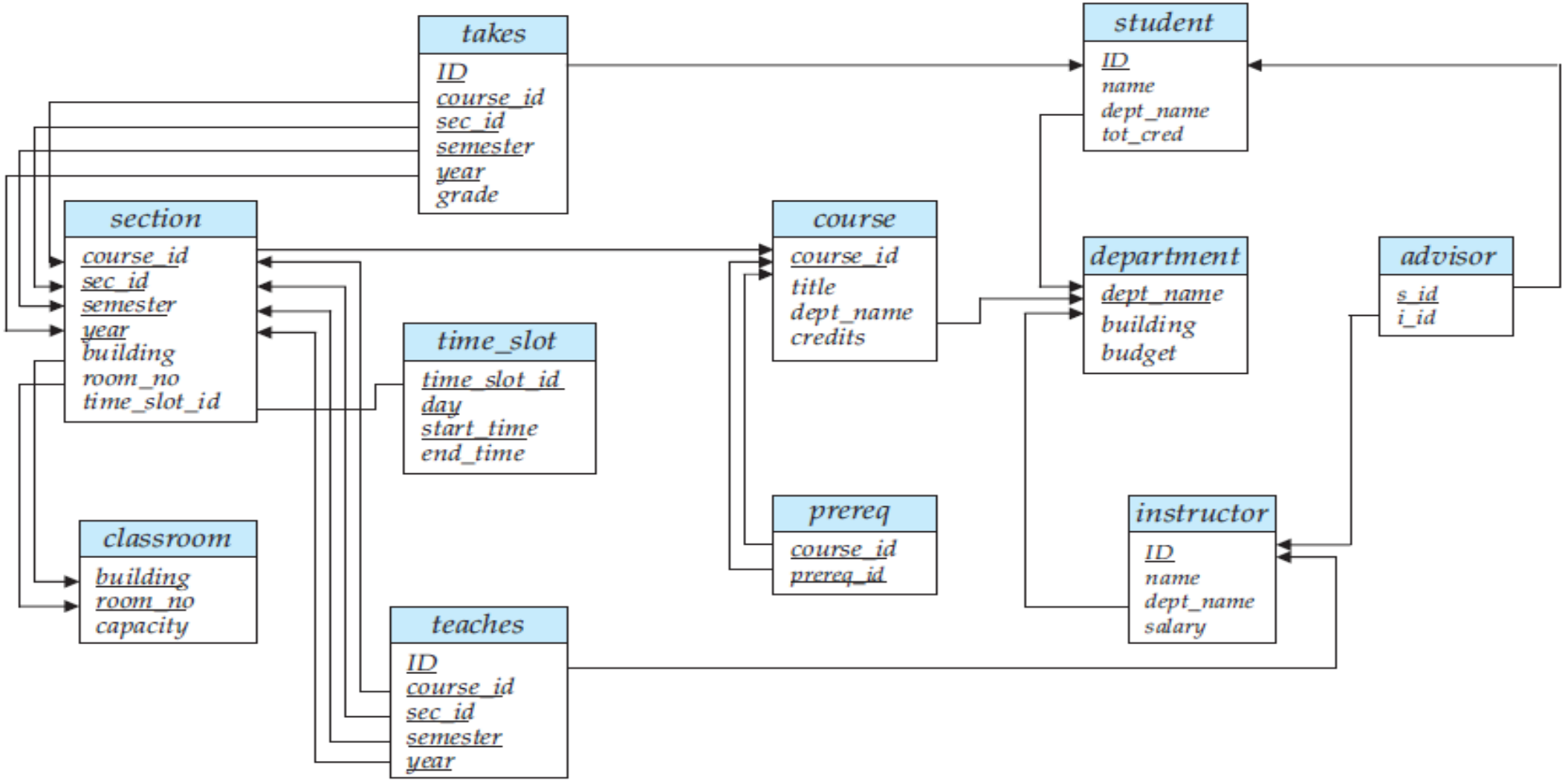
Database Schema

- Database schema is the logical design of the database
- Relation schema consists of a list of attributes and their corresponding domains.
- Schema for relation student can be defined as

Student (ID, name, dept name, tot_cred)

- A database schema, along with primary key and foreign key dependencies, can be depicted by **schema diagrams**
- Each relation appears as a box, with the relation name at the top and the attributes listed inside the box.
- Primary key attributes are shown underlined. Foreign key dependencies appear as arrows from the foreign key attributes of the referencing relation to the primary key of the referenced relation.

Database Schema Diagram



Relational Query Languages

- A query language is a language in which a user requests information from the database
- Query languages can be categorized as either procedural or nonprocedural.
- In a procedural language, the user instructs the system to perform a sequence of operations on the database to compute the desired result.

Example:Relational Algebra

- In a nonprocedural language, the user describes the desired information without giving a specific procedure for obtaining that information

Example:Tuple Relational Calculus(t|p(t)) and Domain Relational Calculus

- The relational algebra consists of a set of operations that take one or two relations as input and produce a new relation as their result.
- The relational calculus uses predicate logic to define the result desired without giving any specific algebraic procedure for obtaining that result

- $\{t \mid t \in \text{instructor} \wedge t[\text{salary}] > 80000\}$

Relational Algebra

- The relational algebra is a *procedural query language*.
- *It consists of a set of operations* that take one or two relations as input and produce a new relation as their result.
- The fundamental operations in the relational algebra are *select*, *project*, *union*, *set difference*, *Cartesian product*, *and rename*

Fundamental Operations

- The select, project, and rename operations are called *unary operations*, *because* they operate on one relation.
- The other three operations operate on pairs of relations and are, therefore, called *binary operations*.

Select Operation(σ)

- The select operation selects tuples that satisfy a given predicate
- The predicate appears as a subscript to σ
- The argument relation is in parentheses after the σ
- In order to select those tuples of the *instructor relation where the instructor is in the "Physics"* department, we write

$\sigma_{\text{dept-name}}$="Physics" (*instructor)*

In order to find all instructors with salary greater than $90,000 by writing

σ_{salary}>9000(instructor)

ID	*name*	*dept_name*	*sala*
10101	Srinivasan	Comp. Sci.	650
12121	Wu	Finance	900
15151	Mozart	Music	400
22222	Einstein	Physics	950
32343	El Said	History	600
33456	Gold	Physics	870
45565	Katz	Comp. Sci.	750
58583	Califieri	History	620
76543	Singh	Finance	800
76766	Crick	Biology	720
83821	Brandt	Comp. Sci.	920
98345	Kim	Elec. Eng.	800

Figure 6.1 The *instructor* relation.

Project Operation(π)

- The project operation is a unary operation that returns its argument relation, with certain attributes left out
- Since a relation is a set, any duplicate rows are eliminated.
- The argument relation follows in parentheses

ID	*name*	*salary*
10101	Srinivasan	65000
12121	Wu	90000
15151	Mozart	40000
22222	Einstein	95000
32343	El Said	60000
33456	Gold	87000
45565	Katz	75000
58583	Califieri	62000
76543	Singh	80000
76766	Crick	72000
83821	Brandt	92000
98345	Kim	80000

Figure 6.3 Result of $\Pi_{ID,\ name,\ salary}(instructor)$.

Composition of Relational Operations

- Since the result of a relational-algebra operation is of the same type(relation) as its inputs, relational-algebra operations can be composed together into a **relational-algebra expression**
- **For Example,consider the query**
- "Find the name of all instructors in the Physics department.

$$\pi_{name}\,(\sigma_{dept\ name} = \text{“}Physics\text{”}\ (instructor))$$

name
Einstein
Gold

The Set-Difference Operation

- The **set-difference operation, denoted by −, allows us to find tuples that are** in one relation but are not in another.
- The expression *r − s produces a relation* containing those tuples in *r but not in s.*

DEPOSITOR RELATION

CUSTOMER_NAME	ACCOUNT_NO
Johnson	A-101
Smith	A-121
Mayes	A-321
Turner	A-176
Johnson	A-273
Jones	A-472
Lindsay	A-284

BORROW RELATION

CUSTOMER_NAME	LOAN_NO
Jones	L-17
Smith	L-23
Hayes	L-15
Jackson	L-14
Curry	L-93
Smith	L-11
Williams	L-17

The Set-Difference Operation

$\pi_{CUSTOMER\_NAME}(Borrow) - \pi_{CUSTOMER\_NAME}(Depositor)$

Output:

CUSTOMER_NAME
Jackson
Hayes
Willians
Curry

The Union Operation

- Suppose there are two tuples R and S. The union operation contains all the tuples that are either in R or S or both in R & S.
- It eliminates the duplicate tuples. It is denoted by ∪.

A union operation must hold the following condition:

- R and S must have the attribute of the same number.
- Duplicate tuples are eliminated automatically.
- $\pi_{CUSTOMER\_NAME}(Borrow) U \pi_{CUSTOMER\_NAME}(Depositor)$

CUSTOMER_NAME
Johnson
Smith
Hayes
Turner
Jones
Lindsay
Jackson
Curry
Williams
Mayes

The Cartesian Product Operation

- The Cartesian-product operation, denoted by a cross (×), allows us to combine information from any two relations
- The Cartesian product is used to combine each row in one table with each row in the other table. It is also known as a cross product.

The Cartesian Product Operation

EMPLOYEE

EMP_ID	EMP_NAME	EMP_DEPT
1	Smith	A
2	Harry	C
3	John	B

DEPARTMENT

DEPT_NO	DEPT_NAME
A	Marketing
B	Sales
C	Legal

EMP_ID	EMP_NAME	EMP_DEPT	DEPT_NO	DEPT_NAME
1	Smith	A	A	Marketing
1	Smith	A	B	Sales
1	Smith	A	C	Legal
2	Harry	C	A	Marketing
2	Harry	C	B	Sales
2	Harry	C	C	Legal
3	John	B	A	Marketing
3	John	B	B	Sales

The Rename Operation(ρ)

- The results of relational algebra are also relations but without any name
- The rename operation provides database designers to rename the output relation

$\rho_x(E)$

returns the result of expression *E under the name x*.

- Lets say we have a table customer, we are fetching customer names and we are renaming the resulted relation to CUST_NAMES

Table: CUSTOMER

Customer_Id	Customer_Name	Customer_City
C10100	Steve	Agra
C10111	Raghu	Agra
C10115	Chaitanya	Noida
C10117	Ajeet	Delhi
C10118	Carl	Delhi

Query:

```
ρ(CUST_NAMES, Π(Customer_Name)(CUSTOMER))
```

Output:

```
CUST_NAMES
----------
Steve
Raghu
Chaitanya
Ajeet
Carl
```

Additional Relational-Algebra Operations

- Additional relational algebra operations are
- The Set-Intersection Operation
- The Natural-Join Operation
- The Assignment Operation
- Outer join Operations

Intersection Operator (∩)

- Intersection operator is denoted by ∩ symbol and it is used to select common rows (tuples) from two tables (relations).
- Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in both R & S.
- Lets say we have two relations R1 and R2 both have same columns and we want to select all those tuples(rows) that are present in both the relations, then in that case we can apply intersection operation on these two relations R1 ∩ R2.

Intersection Operator (∩)

Table 1: COURSE

Course_Id	Student_Name	Student_Id
C101	Aditya	S901
C104	Aditya	S901
C106	Steve	S911
C109	Paul	S921
C115	Lucy	S931

Table 2: STUDENT

Student_Id	Student_Name	Student_Age
S901	Aditya	19
S911	Steve	18
S921	Paul	19
S931	Lucy	17
S941	Carl	16
S951	Rick	18

Query:

```
∏ Student_Name (COURSE) ∩ ∏ Student_Name (STUDENT)
```

Output:

```
Student_Name
------------
Aditya
Steve
Paul
Lucy
```

Natural Join Operation

- The *natural join is a binary operation that allows us to combine certain selections* and a Cartesian product into one operation. It is denoted by the **join symbol**
- The natural-join operation forms a Cartesian product of its two arguments, performs a selection forcing equality on those attributes that appear in both relation schemas, and finally removes duplicate attributes

Natural Join Operation ⋈

Loanno	amount
L-11	900
L-14	1500
L-15	1500
L-16	1300
L-17	1000
L-23	2000

Loan Relation

Customer_ name	Loanno	Amount
Adams	L-16	1300
Hayes	L-15	1500
Jackson	L-14	1500
Jones	L-17	1000
Smith	L-11	900
Smith	L-23	2000

Borrower Relation

Customer_name	Loanno-
Adams	L-16
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23

Natural Join example

empname	street	city
Coyote	Toon	Hollywood
Rabbit	Tunnel	Carrotville
Smith	Revolver	Death Valley
Williams	Seaview	Seattle

Employee

empname	branchname	Salary
Coyotte	Mesa	1500
Rabbit	Mesa	1300
Gates	Redmond	5300
Williams	Redmond	1500

ft-works

Employee ⋈ ft-works

empname	street	city	branchname	Salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500

Outer Join Operation

- The **outer-join operation is an extension of the join operation to deal with missing** information
- The **outer join operation** preserves those tuples that would be lost in an join by creating tuples in the result containing null values
- There are actually three forms of Outer Join Operation:

1.Left outer join

2.Rightt outer join

3.Full outer join

Left Outer Join: ⟕

- When applying join on two relations R and S, some tuples of R or S does not appear in result set which does not satisfy join conditions
- Left Outer Joins give all tuples of R in the result set
- The tuples of R which does not satisfy join condition will have values as NULL for attribute of S

Left Outer Join

empname	street	city
Coyote	Toon	Hollywood
Rabbit	Tunnel	Carrotville
Smith	Revolver	Death Valley
Williams	Seaview	Seattle

Employee

empname	branchname	Salary
Coyotte	Mesa	1500
Rabbit	Mesa	1300
Gates	Redmond	5300
Williams	Redmond	1500

ft-works

Employee ⟕ ft-works

empname	street	city	branchname	Salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500
Smith	Revolver	Death Valley	null	null

Right Outer Join(⟖)

- When applying join on two relations R and S, some tuples of R or S does not appear in result set which does not satisfy join conditions
- Right Outer Join give all tuples of R in the result set
- The tuples of S which does not satisfy join condition will have values as NULL for attribute of R

Right Outer Join

empname	street	city
Coyote	Toon	Hollywood
Rabbit	Tunnel	Carrotville
Smith	Revolver	Death Valley
Williams	Seaview	Seattle

Employee

empname	branchname	Salary
Coyotte	Mesa	1500
Rabbit	Mesa	1300
Gates	Redmond	5300
Williams	Redmond	1500

ft-works

empname	street	city	branchname	Salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500
Gates	null	null	Redmond	5300

Employee ⋈ ft-works

Fully Outer Join(⋈)

- When applying join on two relations R and S, some tuples of R or S does not appear in result set which does not satisfy join conditions
- Fully Outer Join give all tuples of S and all tuples of R in the result set
- The tuples of S which does not satisfy join condition will have values as NULL for attribute of R and vice versa

Full Outer Join

empname	street	city
Coyote	Toon	Hollywood
Rabbit	Tunnel	Carrotville
Smith	Revolver	Death Valley
Williams	Seaview	Seattle

Employee

empname	branchname	Salary
Coyotte	Mesa	1500
Rabbit	Mesa	1300
Gates	Redmond	5300
Williams	Redmond	1500

ft-works

Employee ⟗ ft-works

empname	street	city	branchname	Salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500
Smith	Revolver	Death Valley	null	null
Gates	null	null	Redmond	5300

Assignment Operation

- It is convenient at times to write a relational-algebra expression by assigning parts of it to temporary relation variables.
- The **assignment operation, denoted by ←,** works like assignment in a programming language
- We could write R ⋈ *S as:*
- $temp1 \leftarrow R \times S$
- $temp2 \leftarrow r.A1 = s.A1 \wedge r.A2 = s.A2 \wedge \ldots \wedge r.An = s.An\ (temp1)$
- $result = R \cup S\ (temp2)$

ER Model

- ER Model describes structure of a database with the help of a diagram called ER Diagram
- This is used to define data elements and relationship for a specified system
- The basic object in ER model is entity

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
- For instance, a person may have a *person id property whose* value uniquely identifies that person.

ER Model

Entity Sets

- 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, for example, 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.**
- **For instance, a particular** instructor entity may have the value 12121 for ID, the value Wu for name, the value Finance for dept name, and the value 90000 for salary.

ER Model

Entity Sets

76766	Crick
45565	Katz
10101	Srinivasan
98345	Kim
76543	Singh
22222	Einstein

instructor

98988	Tanaka
12345	Shankar
00128	Zhang
76543	Brown
76653	Aoi
23121	Chavez
44553	Peltier

student

ER Model

Relationship Sets

- A **relationship is an association among several entities.**
- **For example, we can** define a relationship advisor that associates instructor Katz with student Shankar.
- This relationship specifies that Katz is an advisor to student Shankar.
- A **relationship set is a set of relationships of the same type.**
- **Formally, it is a** mathematical relation on $n \geq 2$ entity sets.
- If E1, E2, . . . , En are entity sets, then a relationship set R is a subset of {(e1, e2, . . . , en) | e1 ∈ E1, e2 ∈ E2, . . . , en ∈ En} where (e1, e2, . . . , en) is a relationship.

ER Model

Relationship Sets

- The association between entity sets is referred to as participation; that is, the entity sets *E1*, *E2*, . . . , *En* ***participate in relationship set R.***
- **A relationship instance** in an E-R schema represents an association between the named entities in the real-world enterprise that is being modeled

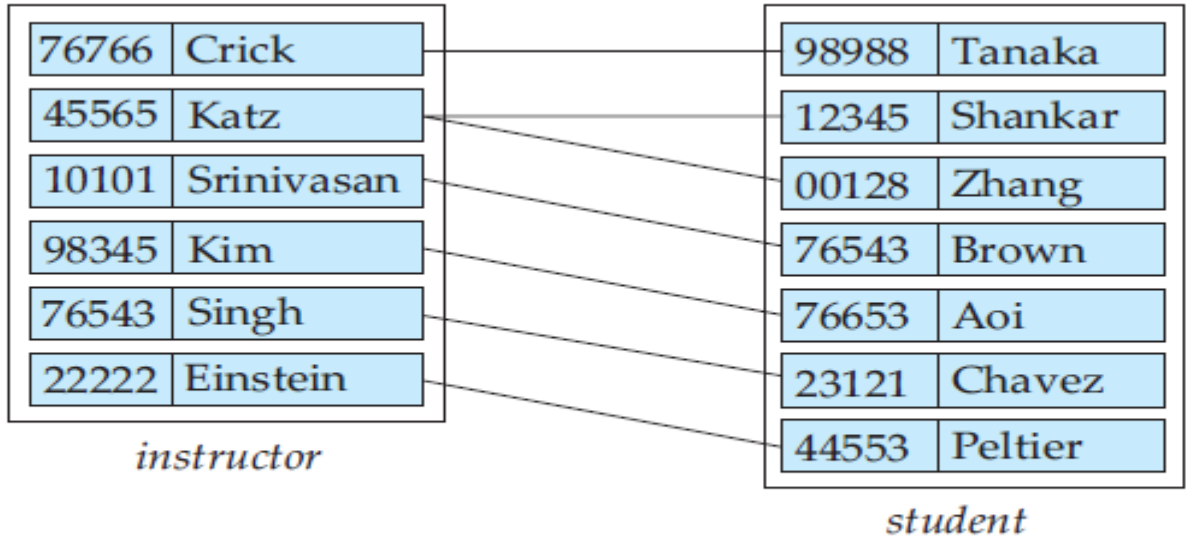


Figure 7.2 Relationship set *advisor*.

ER Model

Relationship Sets

- The function that an entity plays in a relationship is called that entity's **role.**
- Since entity sets participating in a relationship set are generally distinct, roles are implicit and are not usually specified
- If the same entity participates more than once in a relationship it is known as a recursive relationship. In the below example an employee can be a supervisor and be supervised, so there is a recursive relationship

Example of a recursive relationship in ER diagrams

ER Model

Relationship Sets

- A relationship may also have attributes called descriptive attributes
- For Example,employee works for department.Here ‘works for’ is the relation between employee and department entities.The relation ‘works for’ can have attribute DATE_OF_JOIN which is descriptive attribute

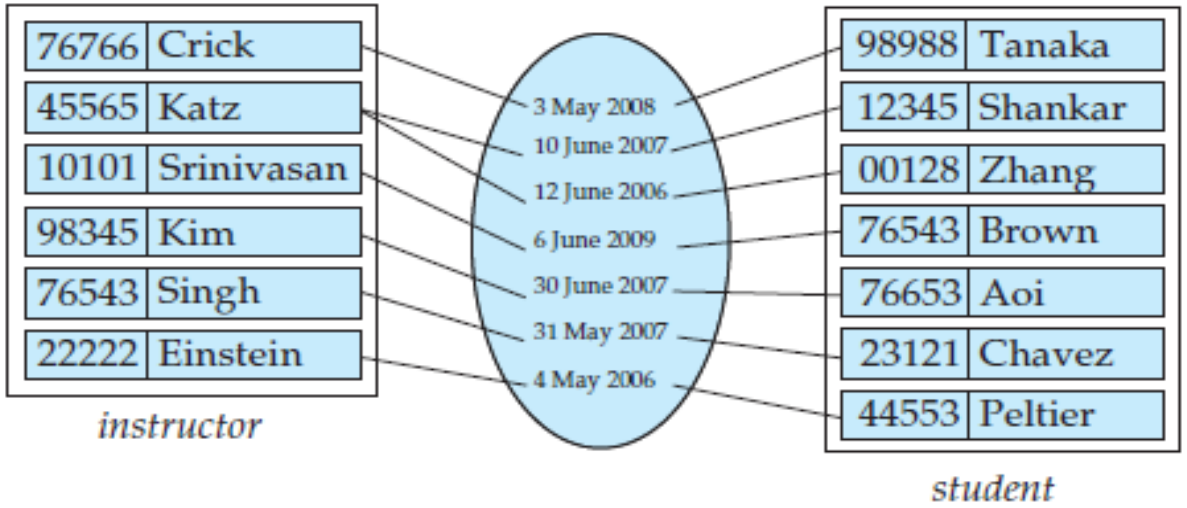


Figure 7.3 *date* as attribute of the *advisor* relationship set.

ER Model

Relationship Sets

- 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.**
- **The domain of attribute course id might be the set of all text** strings of a certain length.
- Similarly, the domain of attribute semester might be strings from the set

{Fall,Winter, Spring, Summer}.

ER Model

Attributes

- An attribute, as used in the E-R model, can be characterized by the following attribute types

1 **Simple and composite attributes:** The attributes that are not divided into subparts are called simple attribute. **Composite** attributes, on the other hand, can be divided into subparts. For example, an attribute name could be structured as a composite attribute consisting of first name, middle initial, and last name

2 **Single-valued and multi valued attributes. The attributes that** have a single value for a particular entity are called single valued attribute.(Eg:Studentid). There may be instances where an attribute has a set of values for a specific entity are called multi valued attribute(Eg:{Phone number})

ER Model

3 **Derived attribute. The value for this type of attribute can be derived from the** values of other related attributes or entities

- Suppose that the instructor entity set has an attribute age that indicates the instructor's age. If the instructor entity set also has an attribute date of birth, we can calculate age from date of birth and the current date.
- Thus, age is a derived attribute. In this case, date of birth may be referred to as a base attribute, or a stored attribute. The value of a derived attribute is not stored but is computed when required.

4 Null Values

- 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

ER Model

Constraints

- An E-R enterprise schema may define certain constraints to which the contents of a database must conform
- This includes Mapping cardinalities, Key Constraints, Participation Constraints

a)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, although they can contribute to the description of relationship sets that involve more than two entity sets

ER Model

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.
- 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
- 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.

ER Model

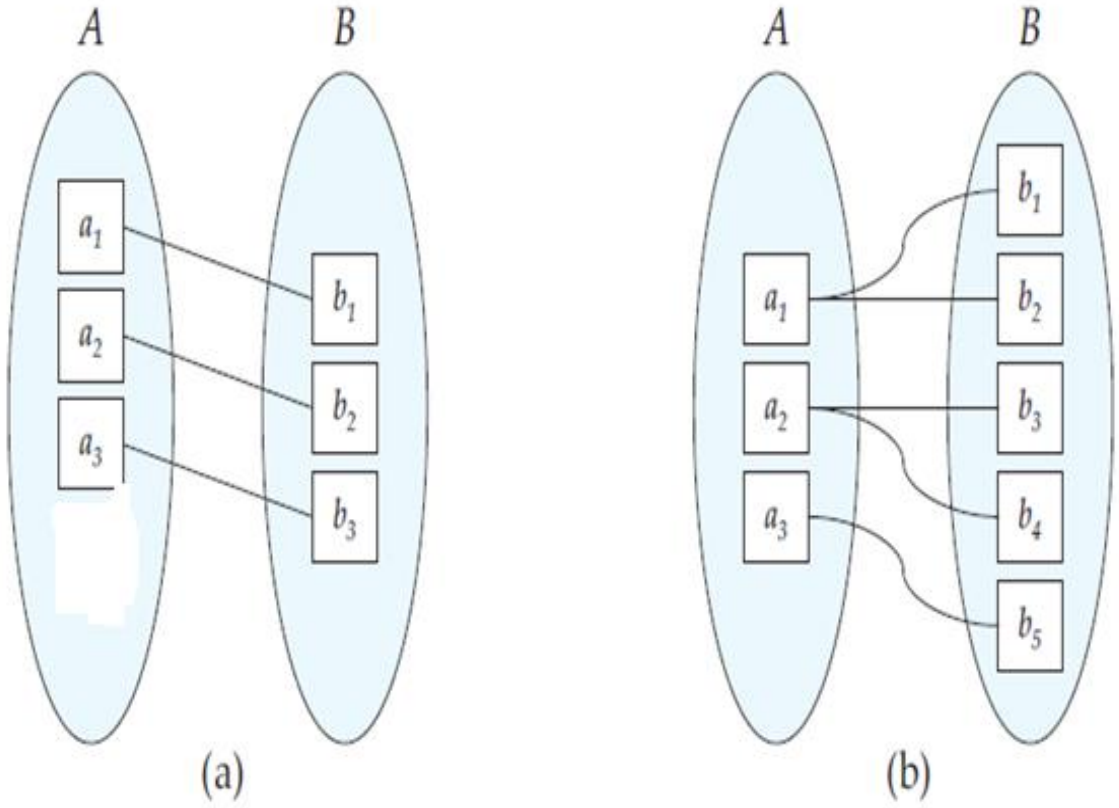


Figure 7.5 Mapping cardinalities. (a) One-to-one. (b) One-to-many.

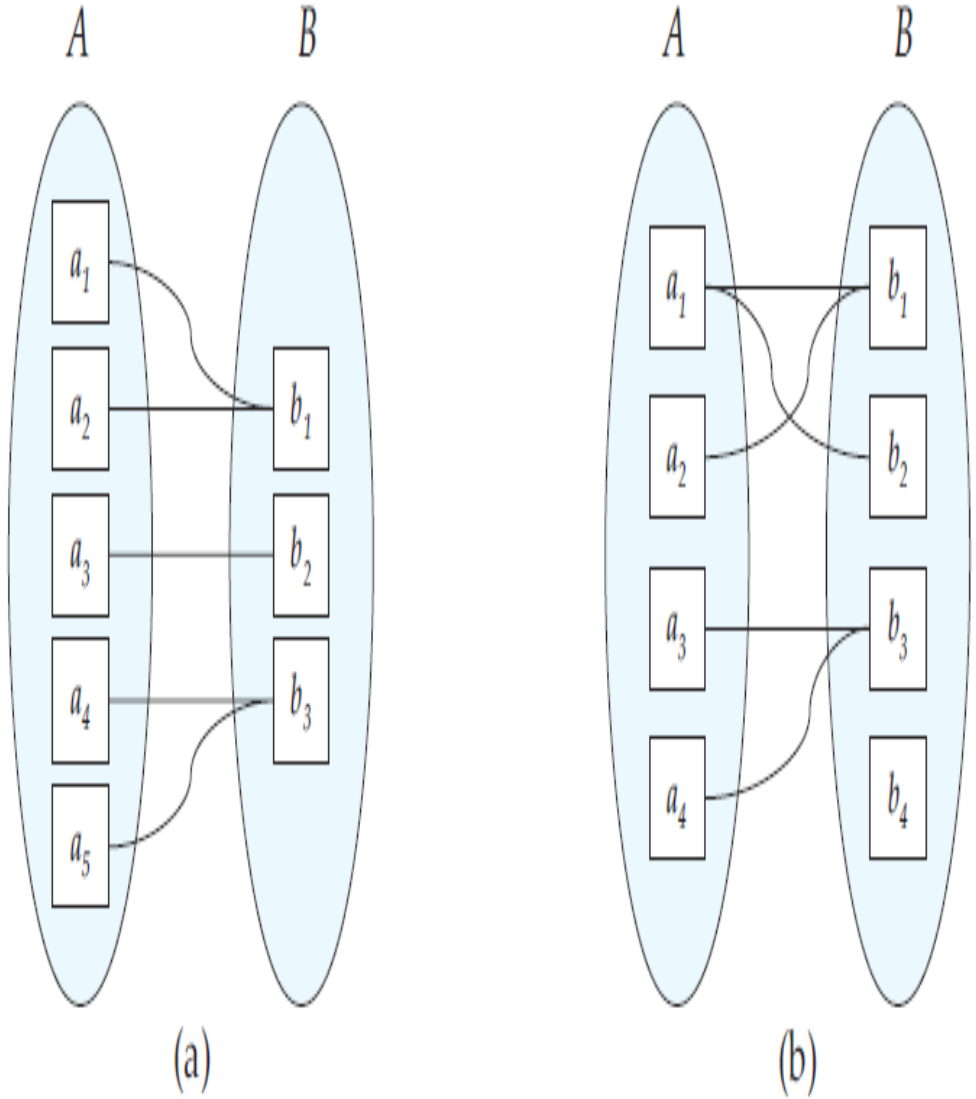


Figure 7.6 Mapping cardinalities. (a) Many-to-one. (b) Many-to-many.

ER Model

b) **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.**
- For Example, the participation of Employee in WORKS_FOR is called total participation ,meaning that every entity in "the total set" of employee entities must be related to department entity via WORKS_FOR

ER Model

C) Key Constraints

- Key for an entity is a set of attributes that is sufficient to distinguish entities from each other.
- Keys also help to identify relationships uniquely, and thus distinguish relationships from each other.
- The primary key of an entity set allows us to distinguish among the various entities of the set
- Let *R be a relationship set involving entity sets E1, E2, . . . , En.*
- Let primarykey(Ei) denote the set of attributes that forms the primary key for entity set Ei .
- Assume for now that the attribute names of all primary keys are unique. The composition of the primary key for a relationship set depends on the set of attributes associated with the relationship set *R.*

ER Model

Basic Structure

- An E-R diagram consists of the following major components:
- Rectangles divided into two parts represent entity sets. The first part, contains the name of the entity set. The second part contains the names of all the attributes of the entity set.
- Diamonds represent relationship sets.
- Undivided rectangles represent the attributes of a relationship set.Attributes that are part of the primary key are underlined.
- Lines link entity sets to relationship sets.
- Dashed lines link attributes of a relationship set to the relationship set.
- Double lines indicate total participation of an entity in a relationship set.
- Double diamonds represent identifying relationship sets linked to weak entity sets

ER Model

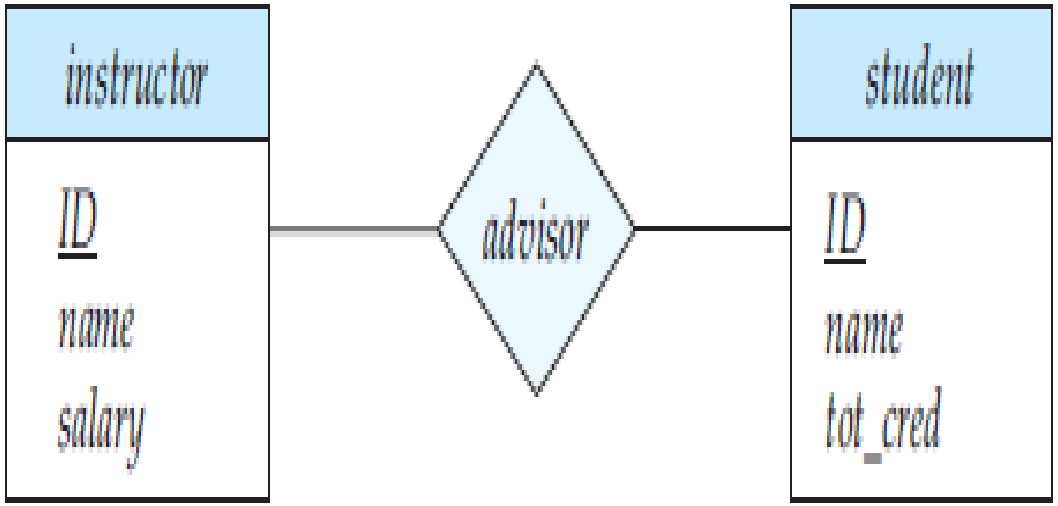


Figure 7.7 E-R diagram corresponding to instructors and students.

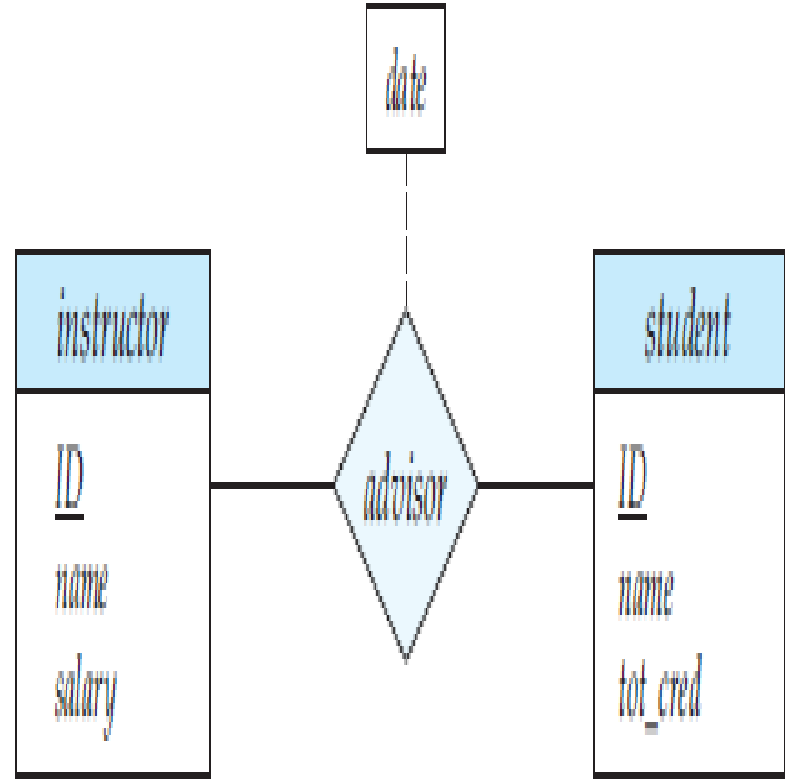
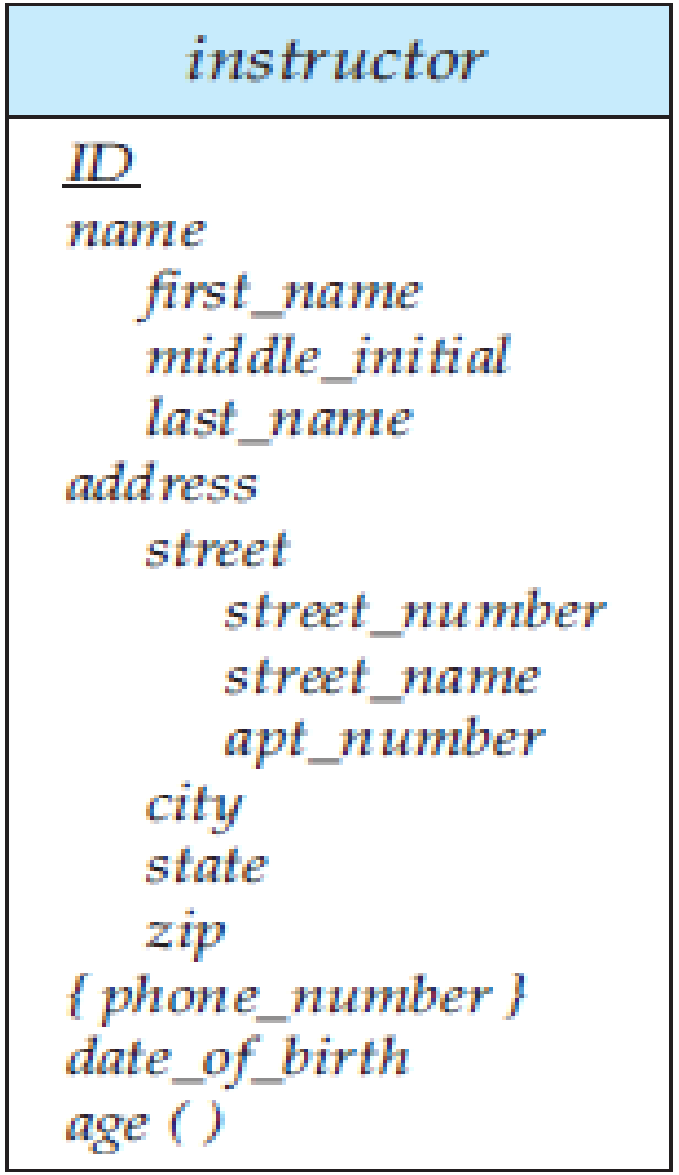


Figure 7.8 E-R diagram with an attribute attached to a relationship set.

ER Model

Complex Attibutes

- In general, composite and multi valued attributes can be nested arbitrarily.
- **For eg,** the address can be defined as the composite attribute *address with the attributes street, city, state, and zip code.*
- The attribute street is itself a composite attribute whose component attributes are street number, street name, and apartment number



ER Model

Weak Entity Set

- An entity set that does not have sufficient attributes to form a primary key 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 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 relationship associating the weak entity set with the identifying entity set is called the **identifying relationship.**
- The **discriminator of a weak entity** set is a set of attributes that allows to distinguish entities in weak entity set

ER Model

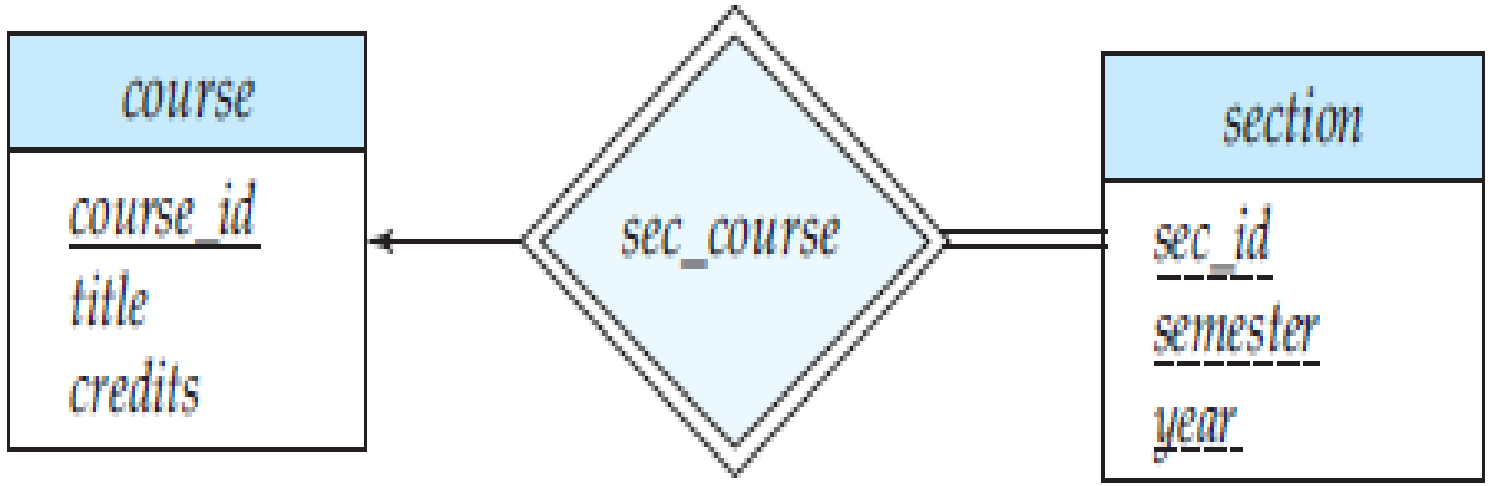


Figure 7.14 E-R diagram with a weak entity set.

ER Model

Roles

- We indicate roles in E-R diagrams by labeling the lines that connect diamonds to rectangles
- Figure shows the role indicators *course id and prereq id between* the *course entity set and the prereq relationship set.*

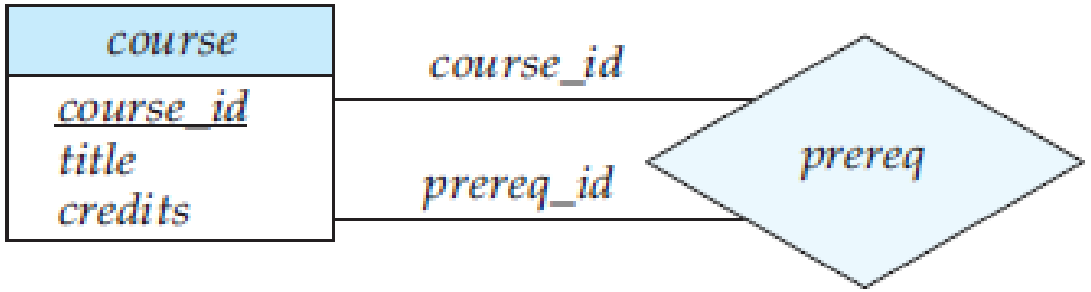


Figure 7.12 E-R diagram with role indicators.

ER Model

Non binary Relationship Sets

- Non binary relationship sets can be specified easily in an E-R diagram.
- Figure consists of the three entity sets *instructor, student, and project, related through the* relationship set *proj guide*.

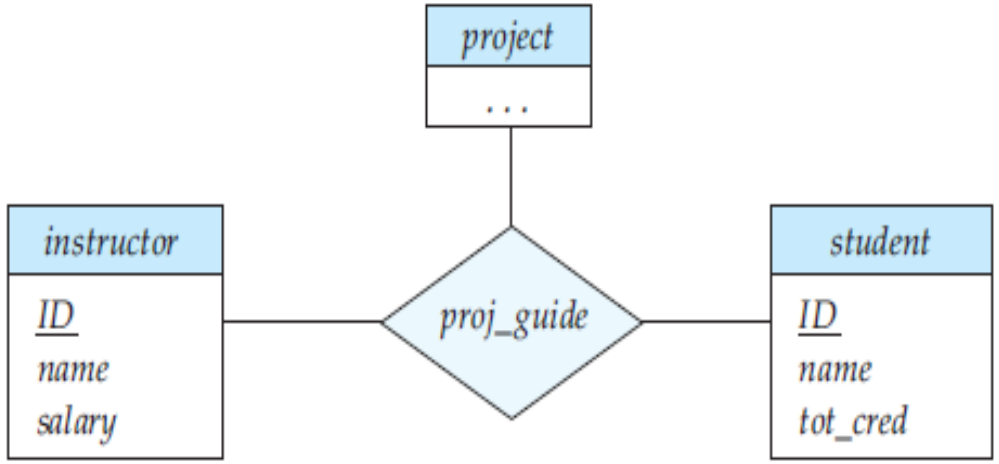


Figure 7.13 E-R diagram with a ternary relationship.

ER Model

Extended E-R Features

1 Specialization

- An entity set may include sub groupings of entities that are distinct in some way from other entities in the set.
- For instance, a subset of entities within an entity set may have attributes that are not shared by all the entities in the entity set.
- The E-R model provides a means for representing these distinctive entity groupings
- As an example, the entity set *person may be further classified as one of the* following:
- *employee.*
- *student.*

ER Model

Extended E-R Features

1 Specialization

- Each of these person types is described by a set of attributes that includes all the attributes of entity set person plus possibly additional attributes.
- For example, employee entities may be described further by the attribute salary, whereas student entities may be described further by the attribute tot cred.
- The process of designating sub groupings within an entity set is called **specialization**
- The specialization relationship may also bereferred to as a **super class-subclass relationship**

ER Model

2 **Generalization**

- The refinement from an initial entity set into successive levels of entity sub groupings 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.
- The database designer may have first identified:
- instructor entity set with attributes instructor id, instructor name, instructor salary, and rank.
- secretary entity set with attributes secretary id, secretary name, secretary salary, and hours per week.

ER Model

2 **Generalization**

- There are similarities between the *instructor entity set and the secretary entity* set in the sense that they have several attributes that are conceptually the same across the two entity sets: namely, the identifier, name, and salary attributes.
- 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.

ER Model

ER Model

3 Attribute Inheritance

- Property of the higher- and lower-level entities created by specialization and generalization is attribute inheritance.
- The attributes of the higher-level entity sets are said to be inherited by the lower-level entity sets.
- For example, student and employee inherit the attributes of person. Thus, student is described by its ID, name, and address attributes, and additionally a tot cred attribute; employee is described by its ID, name, and address attributes, and additionally a salary attribute.
- Attribute inheritance applies through all tiers of lower-level entity sets; thus, instructor and secretary, which are subclasses of employee, inherit the attributes ID, name, and address from person, in addition to inheriting salary from employee

ER Model

3 Attribute Inheritance

Whether a given portion of an E-R model was arrived at by specialization or generalization, the outcome is basically the same:

- A higher-level entity set with attributes and relationships that apply to all of its lower-level entity sets.
- Lower-level entity sets with distinctive features that apply only within a particular lower-level entity set.

ER Model

3 Attribute Inheritance

- Figure depicts a **hierarchy of entity sets. In the figure, employee is a** lower-level entity set of person and a higher-level entity set of the instructor and secretary entity sets.
- In a hierarchy, a given entity set may be involved as a lower level entity set in only one ISA relationship; that is, entity sets in this diagram have only **single inheritance.**
- **If an entity set is a lower-level entity set in more** than one ISA relationship, then the entity set has **multiple inheritance, and the** resulting structure is said to be a lattice.

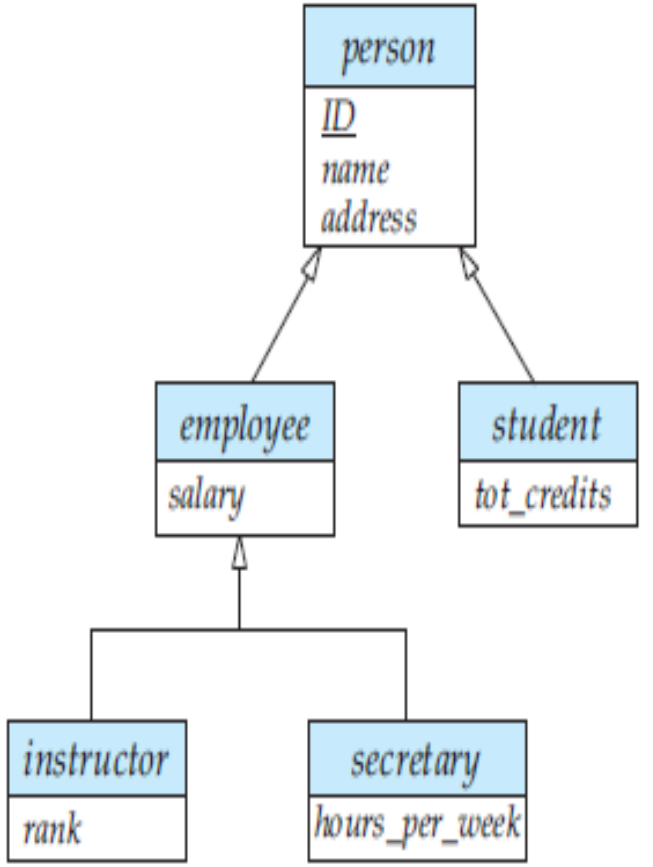


Figure 7.21 Specialization and generalization.

ER Model

4 Constraints on Generalizations

- To model an enterprise more accurately, the database designer may choose to place certain constraints on a particular generalization
- One type of constraint involves determining which entities can be members of a given lower-level entity set. Such membership may be one of the following:

1 Condition-defined:- In condition-defined lower-level entity sets, membership is evaluated on the basis of whether or not an entity satisfies an explicit condition or predicate

- For example, assume that the higher-level entity set student has the attribute student type. All student entities are evaluated on the defining student type attribute.
- Only those entities that satisfy the condition student type = “graduate” are allowed to belong to the lower-level entity set graduate student.

ER Model

4 Constraints on Generalizations

2 **User-defined.**

- The database user assigns entities to a given entity set.
- For instance, let us assume that, after 3 months of employment, university employees are assigned to one of four work teams.
- We therefore represent the teams as four lower-level entity sets of the higher-level *employee* entity set.
- A given employee is not assigned to a specific team entity automatically on the basis of an explicit defining condition. Instead, the user in charge of this decision makes the team assignment on an individual basis.
- The assignment is implemented by an operation that adds an entity to an entity set

ER Model

4 Constraints on Generalizations

A second type of constraint relates to whether or not entities may belong to more than one lower-level entity set within a single generalization. The lower level entity sets may be one of the following

1 Disjoint. A disjointness constraint requires that an entity belong to no more than one lower-level entity set.

- Consider student entity that can satisfy only one condition for the student type attribute; an entity can be either a post graduate student or an undergraduate student, but cannot be both.

ER Model

4 Constraints on Generalizations

2 Overlapping.

- In overlapping generalizations, the same entity may belong to more than one lower-level entity set within a single generalization.
- For an illustration, consider the employee work-team example, and assume that certain employees participate in more than one work team.
- A given employee may therefore appear in more than one of the team entity sets that are lower level entity sets of employee. Thus, the generalization is overlapping.

ER Model

4 Constraints on Generalizations

- A final constraint, the **completeness constraint on a generalization or specialization,** specifies whether or not an entity in the higher-level entity set must belong to at least one of the lower-level entity sets within the generalization/specialization.
- This constraint may be one of the following:

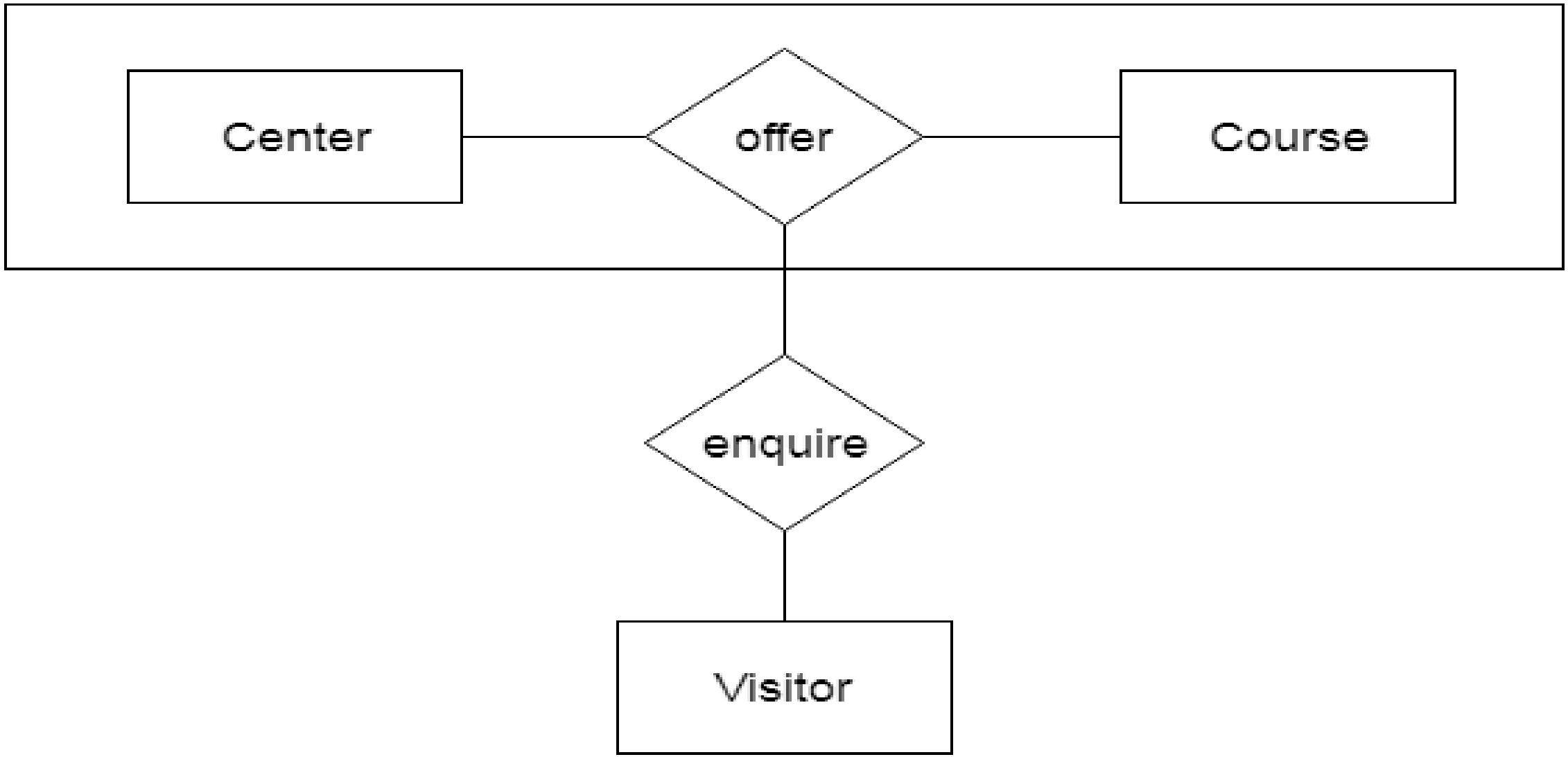
1 **Total generalization or specialization**. Each higher-level entity must belong to a lower-level entity set.

2 **Partial generalization or specialization**. Some higher-level entities may not belong to any lower-level entity set.

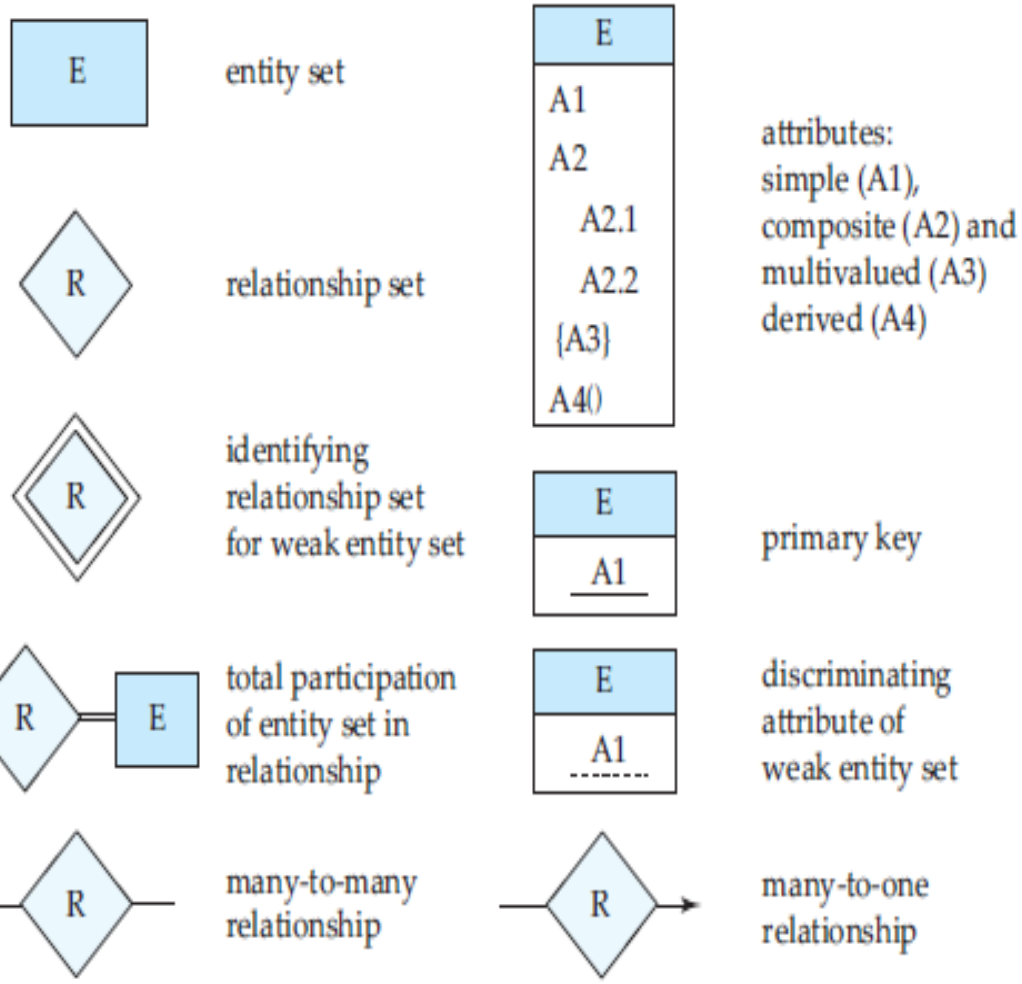
ER Model

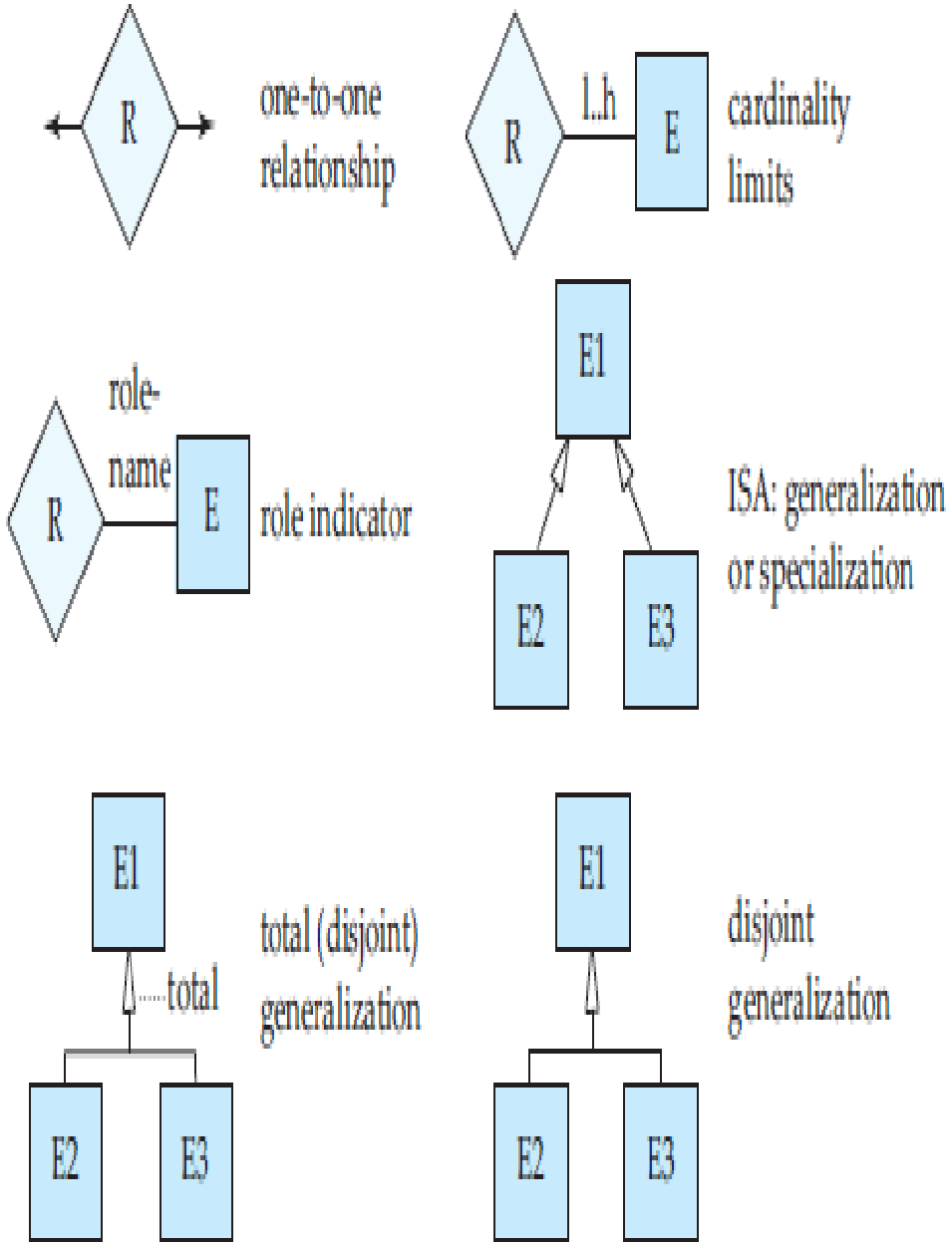
5 Aggregation

- An ER diagram is not capable of representing relationship between an entity and a relationship which may be required in some scenario
- In those cases, a relationship with its corresponding entities is aggregated into a higher level entity
- Aggregation is an abstraction through which relationships are treated as higher-level entities.
- In aggregation, the relation between two entities is treated as a single entity. In aggregation, relationship with its corresponding entities is aggregated into a higher level entity.
- **For example:** Center entity offers the Course entity act as a single entity in the relationship which is in a relationship with another entity visitor. In the real world, if a visitor visits a coaching center then he will never enquiry about the Course only or just about the Center instead he will ask the enquiry about both.


Center
offer
Course
enquire
Visitor

Symbols Used in ER Diagram





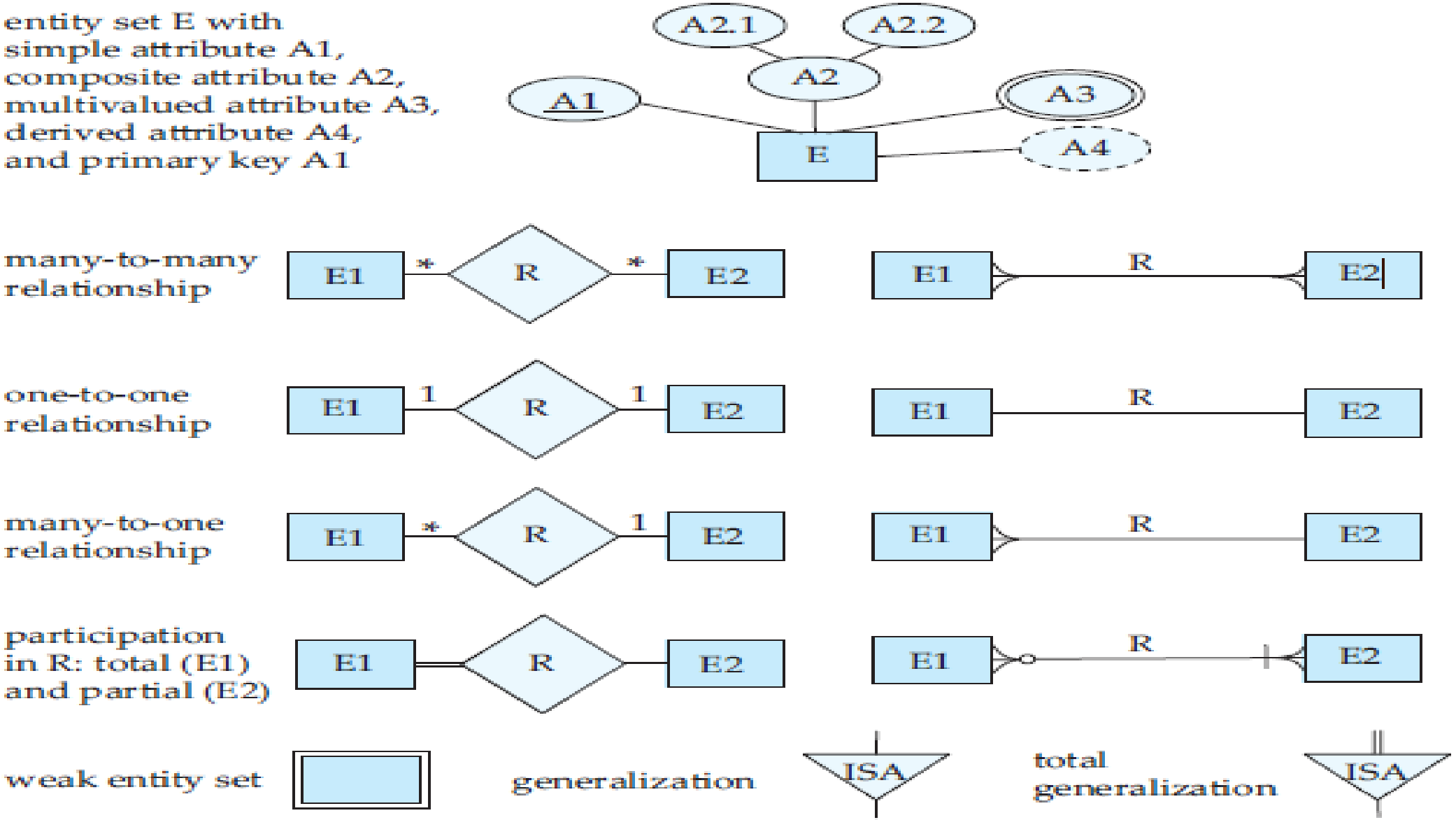
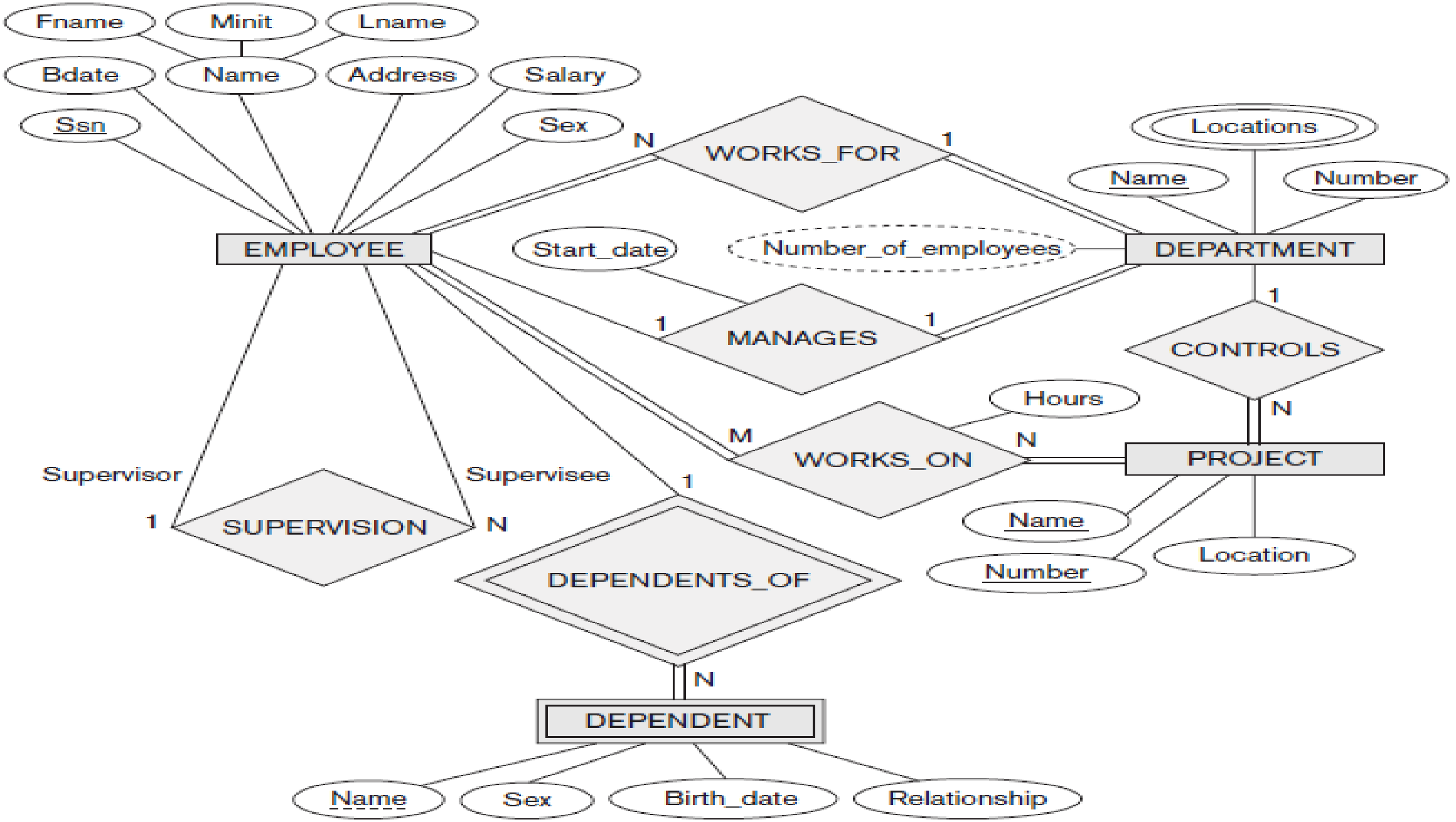


Figure 7.25 Alternative E-R notations.


Fname
Minit
Lname
Bdate
Name
Address
Salary
Ssn
Sex
N
WORKS_FOR
1
Locations
Name
Number
EMPLOYEE
Start_date
Number_of_employees
DEPARTMENT
1
1
MANAGES
1
CONTROLS
Hours
M
N
WORKS_ON
N
PROJECT
Supervisor
Supervisee
1
1
SUPERVISION
N
Name
Location
Number
DEPENDENTS_OF
N
DEPENDENT
Name
Sex
Birth_date
Relationship

Relational Database Design Using ER-to-Relational Mapping

1 Representation of Strong Entity Sets with Simple Attributes

2 Representation of Strong Entity Sets with Complex Attributes

3 Representation of Weak Entity Sets

4 Representation of Relationship Sets

Representation of Strong Entity Sets with Simple Attributes

- 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

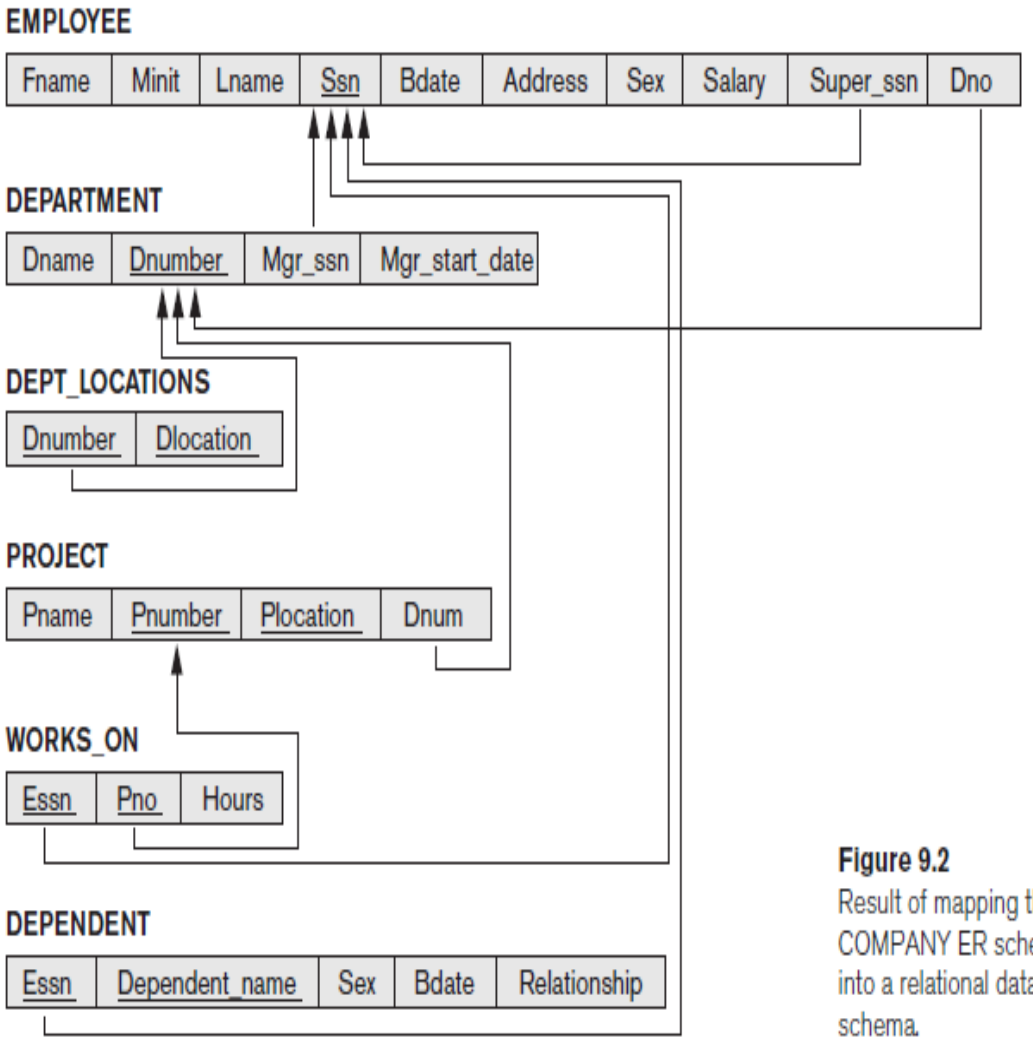


Figure 9.2
Result of mapping the COMPANY ER schema into a relational database schema.

Representation of Weak Entity Set

- Let A be a weak entity set with attributes a1, a2, . . . , am. Let B be the strong entity set on which A depends. Let the primary key of B consist of attributes b1, b2, . . . , bn. We represent the entity set A by a relation schema called A with one attribute for each member of the set:
- $\{a1, a2, \ldots, am\} \cup \{b1, b2, \ldots, bn\}$
- For schemas derived from a weak entity set, the combination of the primary key of the strong entity set and the discriminator of the weak entity set serves as the primary key of the schema
- Consider the weak entity set This entity set has the attributes: sec id, semester, and year. The primary key of the course entity set, on which section depends, is course id. Thus, we represent section by a schema with the following attributes:
- section (course id, sec id, semester, year)

Representation of Strong Entity Sets with Complex Attributes

- Composite Attribute in ER Diagram is replaced by its constituents
- For the composite attribute name, the schema generated for instructor contains the attributes first name, middle name, and last name; there is no separate attribute or schema for name.
- Similarly, for the composite attribute address, the schema generated contains the attributes street, city, state, and zip code.
- Since street is a composite attribute it is replaced by street number, street name, and apt number
- The relational schema derived from the version of entity set *instructor with* complex attributes are
- instructor (ID, first name, middle name, last name,street number, street name, apt number,city, state, zip code, date of birth)

Representation of Strong Entity Sets with Complex Attributes

- For a multi valued attribute M, we create a relation schema R with an attribute A that corresponds to M and attributes corresponding to the primary key of the entity set or relationship set of which M is an attribute
- The primary key of instructor is ID. For this multi valued attribute, we create a relation schema
- instructor phone (ID, phone number)

Representation of Relationship Sets

- Let R be a relationship set, let a1, a2, . . . , am be the set of attributes formed by the union of the primary keys of each of the entity sets participating in R, and
- Let the descriptive attributes (if any) of R be b1, b2, . . . , bn.We represent this relationship set by a relation schema called R with one attribute for each member of the set:
- {a1, a2, . . . , am} ∪ {b1, b2, . . . , bn}
- The primary key is instead chosen as follows:
- For a binary many-to-many relationship, the union of the primary-key attributes from the participating entity sets becomes the primary key.
- For a binary one-to-one relationship set, the primary key of either entity set can be chosen as the primary key. The choice can be made arbitrarily.