Module 1

Database System Concepts, 6th Ed.

©Silberschatz, Korth and Sudarshan
See www.db-book.com for conditions on re-use

Outline

Module I

Introduction, Applications, Purpose of Database Systems, View of Data, Database Languages, Database Architecture, Database Users and Administrators

Database Design: The Entity Relationship Model, Constraints, Removing Redundant Attributes Entity Sets, Entity Relationship Diagrams, Reduction to Relational Schemas, Extended E-R Features

©Silberschatz, Korth and Sudarshan

Database Management System (DBMS)

- DBMS contains information about a particular enterprise
  - Collection of interrelated data
  - Set of programs to access the data
  - An environment that is both *convenient* and *efficient* to use
- Database Applications:
  - Banking: transactions
  - Airlines: reservations, schedules
  - Universities: registration, grades
  - Sales: customers, products, purchases
  - Online retailers: order tracking, customized recommendations
  - Manufacturing: production, inventory, orders, supply chain
  - Human resources: employee records, salaries, tax deductions
- Databases can be very large.
- Databases touch all aspects of our lives

Purpose of Database System

- In the early days, database applications were built on top of file systems
- Drawbacks of using file systems to store data:
  - Data redundancy and inconsistency
    - Multiple file formats, duplication of information in different files
  - Difficulty in accessing data
    - Need to write a new program to carry out each new task
  - Data isolation — multiple files and formats
  - Integrity problems
    - Integrity constraints (e.g. account balance > 0) become part of program code
    - Hard to add new constraints or change existing ones

©Silberschatz, Korth and Sudarshan

Simplified database system environment

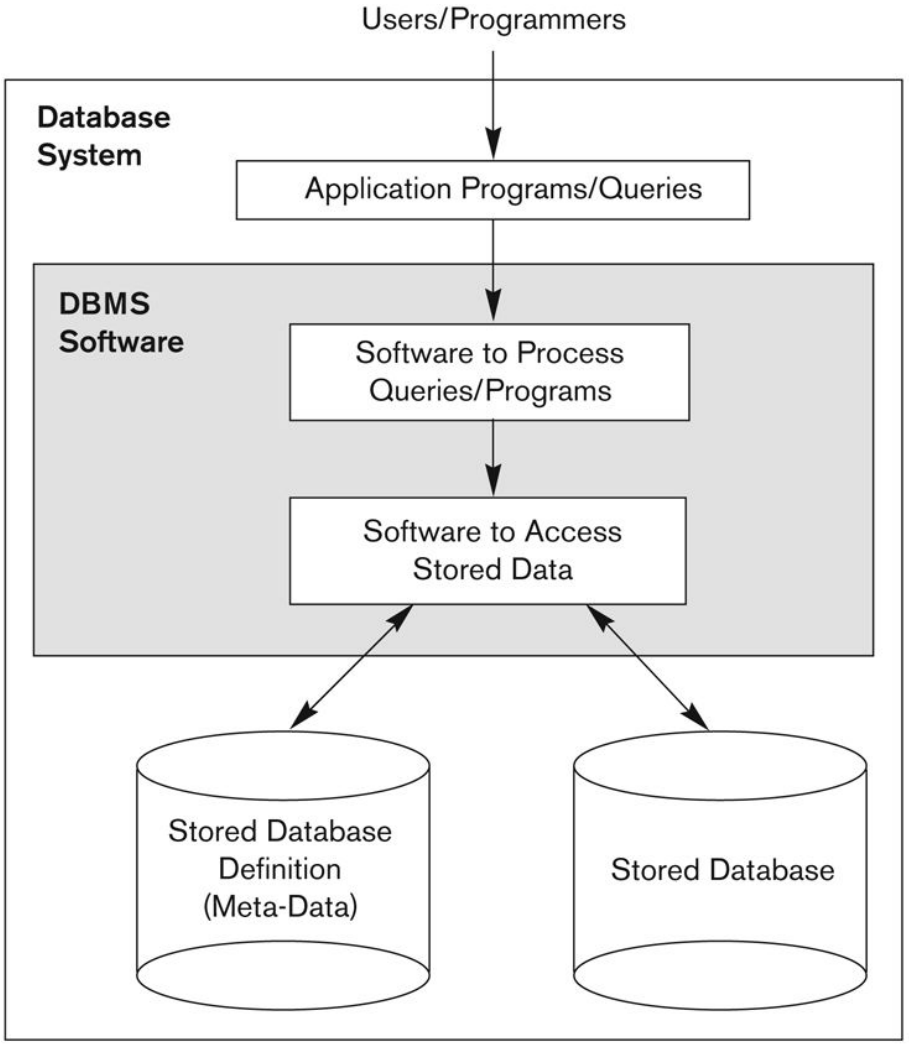


Figure 1.1
A simplified database system environment.

University Database Example

- Application program examples
  - Add new students, instructors, and courses
  - Register students for courses, and generate class rosters
  - Assign grades to students, compute grade point averages (GPA) and generate transcripts
- In the early days, database applications were built directly on top of file systems

©Silberschatz, Korth and Sudarshan

Drawbacks of using file systems to store data

- Data redundancy and inconsistency
  - Multiple file formats, duplication of information in different files
- Difficulty in accessing data
  - Need to write a new program to carry out each new task
- Data isolation
  - Multiple files and formats
- Integrity problems
  - Integrity constraints (e.g., account balance > 0) become "buried" in program code rather than being stated explicitly
  - Hard to add new constraints or change existing ones

©Silberschatz, Korth and Sudarshan

Drawbacks of using file systems to store data (Cont.)

- Atomicity of updates
  - Failures may leave database in an inconsistent state with partial updates carried out
  - Example: Transfer of funds from one account to another should either complete or not happen at all
- Concurrent access by multiple users
  - Concurrent access needed for performance
  - Uncontrolled concurrent accesses can lead to inconsistencies
    - Example: Two people reading a balance (say 100) and updating it by withdrawing money (say 50 each) at the same time
- Security problems
  - Hard to provide user access to some, but not all, data

Database systems offer solutions to all the above problems

Levels of Abstraction/Data Abstraction

- **Physical level:** describes how a record (e.g., instructor) is stored.
- **Logical level:** describes data stored in database, and the relationships among the data.

```
type instructor = record
        ID : string;
    name : string;
    dept_name : string;
    salary : integer;
        end;
```

Physical Data Independence – the ability to modify the physical schema without changing the logical schema

- **View level:** application programs hide details of data types. Views can also hide information (such as an employee's salary) for security purposes.

©Silberschatz, Korth and Sudarshan

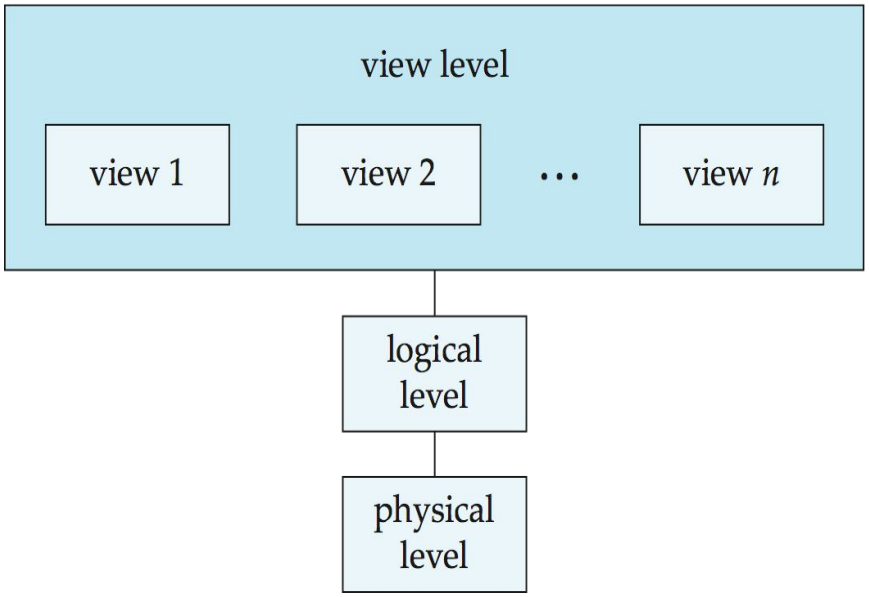
- At the physical level-->, an instructor,department or student record can be described as a block of consecutive storage locations (for example, words or bytes).
- The language compiler hides this level of detail from programmers.
- Database admin aware of this level

- Logical level-->Each record described by type definition as in the code,interrelationships. Programmers work in this level

- View level-->Users see set of application programs(hides details of data types)
- View also provide security mechanisms

View of Data

3 levels of data abstraction



©Silberschatz, Korth and Sudarshan

Instances and Schemas

- Similar to types and variables in programming languages
- **Logical Schema** – the overall logical structure of the database
  - Example: The database consists of information about a set of customers and accounts in a bank and the relationship between them
    - Analogous to type information of a variable in a program
- **Physical schema**– the overall physical structure of the database
- **Instance** – the actual content of the database at a particular point in time
  - Analogous to the value of a variable
- **Physical Data Independence** – the ability to modify the physical schema without changing the logical schema
  - Applications depend on the logical schema
  - In general, the interfaces between the various levels and components should be well defined so that changes in some parts do not seriously influence others.

Data Models

- A collection of tools for describing
  - Data
  - Data relationships
  - Data semantics
  - Data constraints
- Relational model
- Entity-Relationship data model (mainly for database design)
- Object-based data models (Object-oriented and Object-relational)
- Semistructured data model (XML)
- Other older models:
  - Network model
  - Hierarchical model

©Silberschatz, Korth and Sudarshan

Relational Model

- All the data is stored in various tables.
- Example of tabular data in the relational model

Columns

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

Rows

(a) The *instructor* table

©Silberschatz, Korth and Sudarshan

A Sample Relational Database Example:University DB

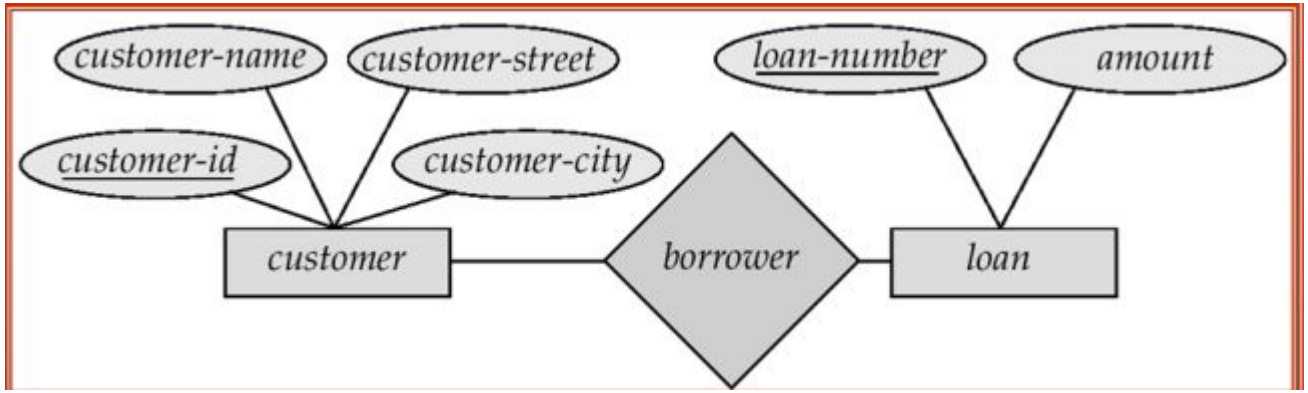
ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music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

(a) The *instructor* table

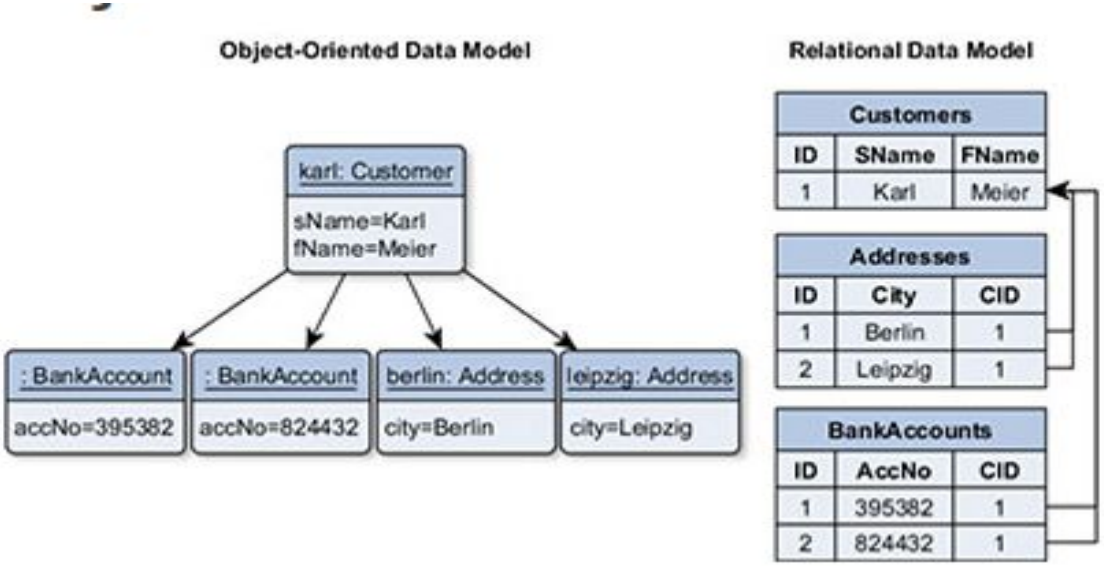
dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table

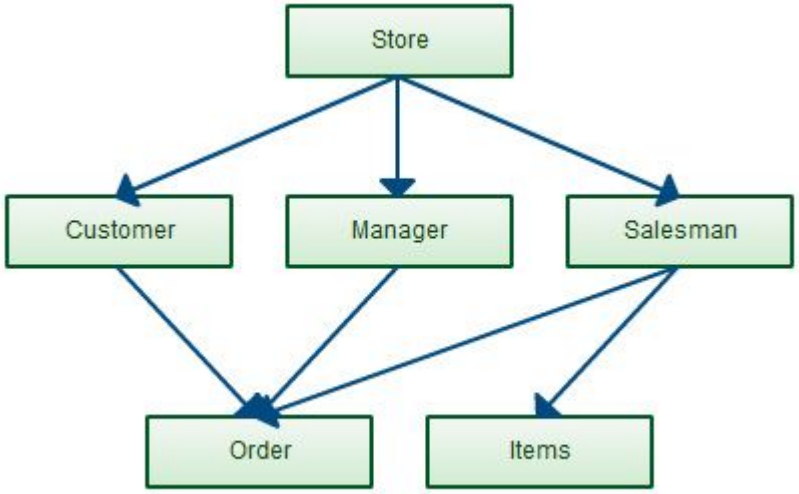
Entity-Relationship data model/ER Model



Object-based data models (Object-oriented and Object-relational)
Helps users who wishes to include OO concepts



Network model



Database Languages:DDL and DML

Data Definition language(DDL) -->TO specify database Schema

Data-Manipulation Language(DML)-->To express database queries and updates

©Silberschatz, Korth and Sudarshan

Data Definition Language (DDL)

- Set of definition expression
- Special type of DDL→Data storage and definition language(define the implementation details of the database schemas),storage structure,access methods
- Specification notation for defining the database schema

  Example: **create table** *instructor* (
  ID **char**(5),
  name **varchar**(20),
  dept_name **varchar**(20),
  salary **numeric**(8,2))

- DDL compiler generates a set of table templates stored in a ***data dictionary***
- Data dictionary contains metadata (i.e., data about data)
- Consistency Constraints

student(sno,sname)-schema

create table student(

Sno int,

Sname varchar(20),

Saddress varchar(60)

Sno	Sname

DDL-Consistency constraints

The data values stored in the database must satisfy certain consistency constraints.

For example, suppose the balance on an account should not fall below $100. The DDL provides facilities to specify such constraints. The database systems check these constraints every time the database is updated.

1)Domain constraints:the set of possible values that can be associated with an attribute

2)Referential Integrity:

3)Assertion:any condition that a database shd always satisfy

4)Authorization:

Data Manipulation Language (DML)

- Language for accessing and manipulating the data organized by the appropriate data model
  - DML also known as query language
  - Types of access are

- Retrieval of information stored in the database.
- Insertion of new information into the database.
- Deletion of information from the database.
- Modification of information stored in the database.

- Two types:

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

Two classes of languages

Pure – used for proving properties about computational power and for optimization

Relational Algebra

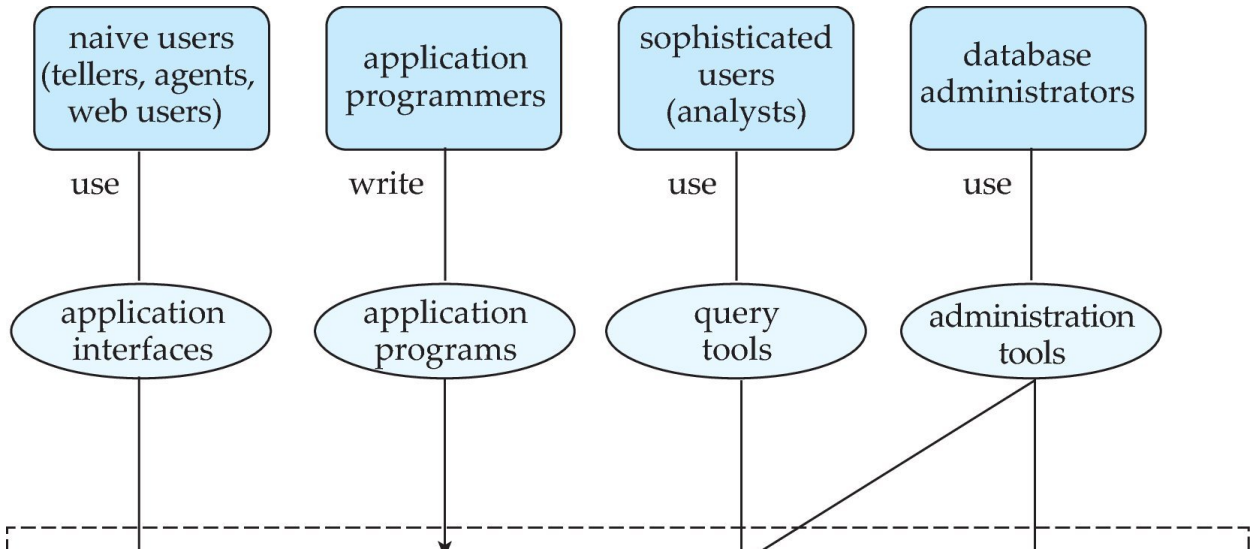
Tuple relational calculus

Domain relational calculus

Commercial – used in commercial systems

SQL is the most widely used commercial language

Database Users and Administrators



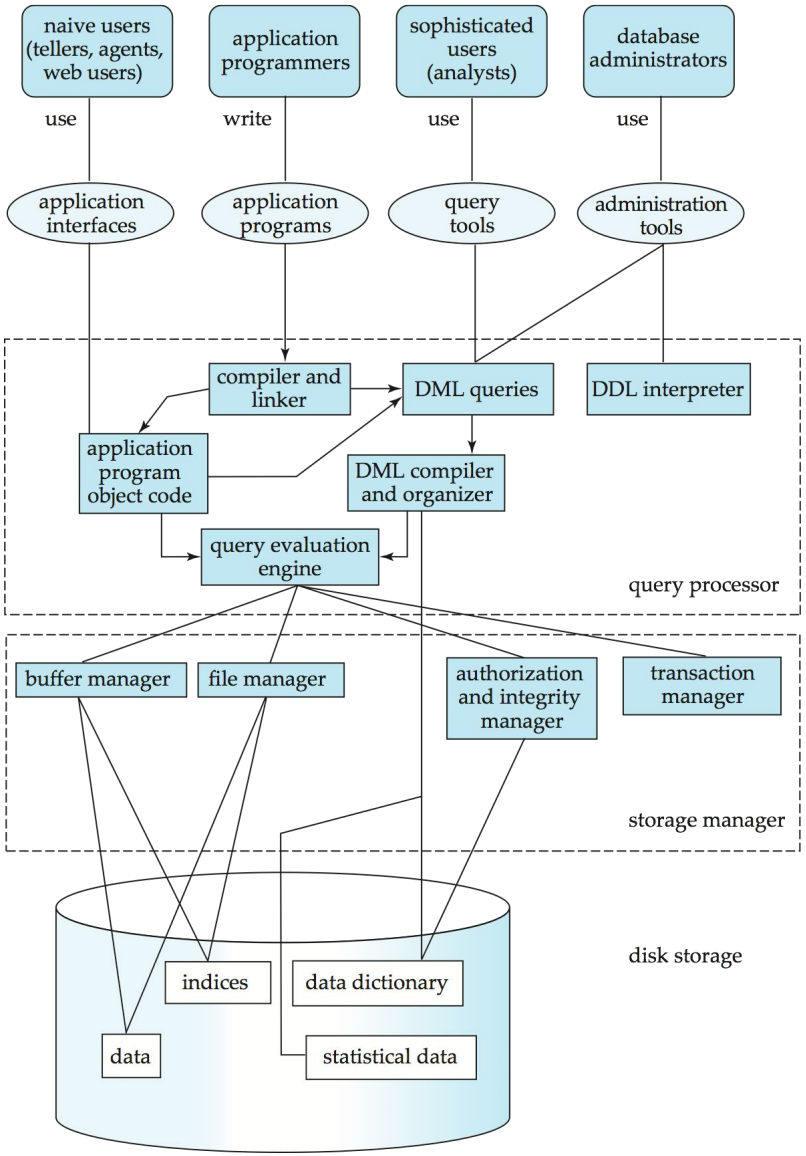
Database Administrator

- Coordinates all the activities of the database system; the database administrator has a good understanding of the enterprise's information resources and needs.
- Database administrator's duties include:
  - Schema definition
  - Storage structure and access method definition
  - Schema and physical organization modification
  - Granting user authority to access the database
  - Specifying integrity constraints
  - Acting as liaison with users
  - Monitoring performance and responding to changes in requirements

Database Users

- Users are differentiated by the way they expect to interact with the system
- Application programmers – interact with system through DML calls
- Sophisticated users – form requests in a database query language
- Specialized users – write specialized database applications that do not fit into the traditional data processing framework
- Naïve users – invoke one of the permanent application programs that have been written previously
  - E.g. people accessing database over the web, bank tellers, clerical staff

Database System Internals/System Structure



Database Architecture

The architecture of a database systems is greatly influenced by the underlying computer system on which the database is running:

- Centralized
- Client-server
- Parallel (multi-processor)
- Distributed

©Silberschatz, Korth and Sudarshan

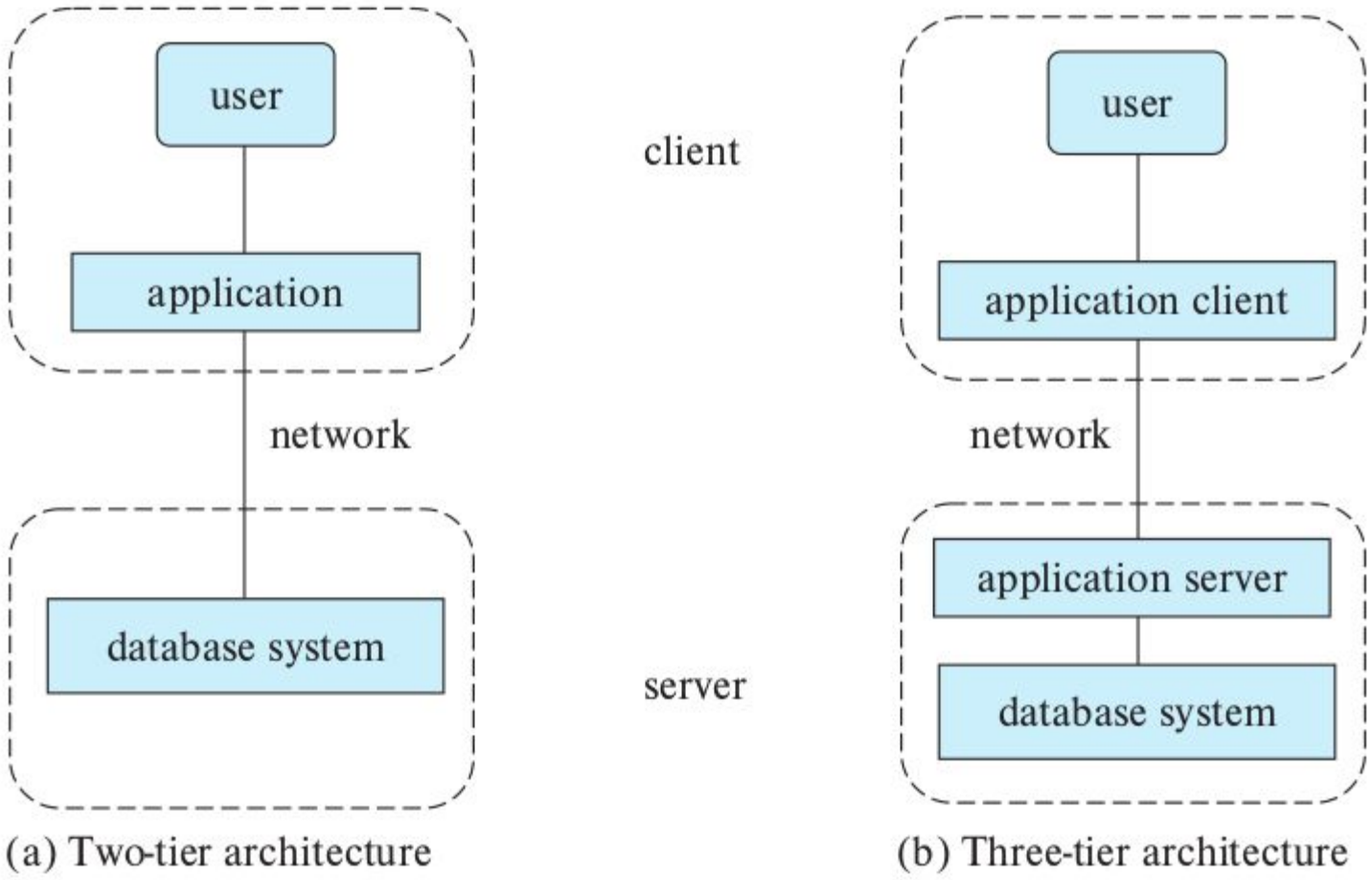


Figure 1.4 Two-tier and three-tier architectures.

Database Design Using the E-R Model

Design Phases

1)Initial phase --

characterize fully the data needs of the prospective database users

Designer interact with domain experts and users

Outcome of this phase is specification of **user requirements.**

2) conceptual design phase--

Selection of a data model and **make a conceptual schema**

Designer choose a data model

Translate the user requirements collected during the first first phase into a conceptual schema

ER model used to represent the conceptual design

Specifies:entities,attributes of entities,relationships among entities,constraints of entities and relationships

Outcome creation of **ER diagram**

©Silberschatz, Korth and Sudarshan

Design Phases

3)Fully developed conceptual schema

With functional requirements of the enterprise

Users specify the the kinds of operations (or transactions) that will be performed on the data.

Modifying or updating data,searching data,deleting data

©Silberschatz, Korth and Sudarshan

Design Phases

4)Final design phase

a)Logical Design:

- designer maps the high-level conceptual schema onto the implementation data model of the database system that will be used.
- The implementation data model is typically the relational data model,
- mapping the conceptual schema defined using the entity-relationship model into a relation schema

b)physical-design phase

- physical features of the database are specified.
- These features include the form of file organization and choice of index structures

©Silberschatz, Korth and Sudarshan

Design Alternatives:

Entity: a “thing” or “object” in the enterprise that is distinguishable from other objects. Described by a set of attributes.(eg:student,teacher,Course….)

In designing a database schema, we must ensure that we avoid two major pitfalls:

Redundancy:

Incompleteness:

The Entity-Relationship Model

- The ER data mode was developed to facilitate database design by allowing specification of an enterprise schema that represents the overall logical structure of a database.
- The ER model is very useful in mapping the meanings and interactions of real-world enterprises onto a conceptual schema. Because of this usefulness, many database-design tools draw on concepts from the ER model.
- The ER data model employs three basic concepts:

  (i)entity sets,

  (ii)relationship sets, attributes.

  (iii)The ER model also has an associated diagrammatic representation, the **ER diagram**, which can express the overall logical structure of a database graphically.

Entity Sets

An **entity** is an object that exists and is distinguishable from other objects.

Example: specific person, company, event, plant

An **entity set** is a set of entities of the same type that share the same properties.

Example: set of all persons, companies, trees, holidays

An entity is represented by a set of attributes; i.e., descriptive properties possessed by all members of an entity set.

Example:

instructor = (*ID, name, street, city, salary*)
course= (*course_id, title, credits*)

A subset of the attributes form a **primary key** of the entity set; i.e., uniquely identifiying each member of the set.

We use the term **extension** of the **entity set** to refer to the actual collection of entities belonging to the entity set. Thus, the set of actual instructors in the university forms the extension of the entity set instructor.

©Silberschatz, Korth and Sudarshan

Attributes

Attributes:

-An entity is represented by a set of attributes, that is descriptive properties possessed by all members of an entity set.

-each entity may have its own value for each attribute

-Possible attributes of the instructor entity set are ID , name, dept name, and salary.

-Possible attributes of the course entity set are course id, title, dept name, and credits.

-Domain – the set of permitted values for each attribute

-Simple attributes- that are simple— those not divided into subparts.

Attributes

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.

The ID attribute is used to identify instructors uniquely

Attribute types:

1)Simple and **composite** attributes.

Fname-→simple

Name→Composite(can be divided into Fname,Mini,Lastname)

Address-→composite(house no,house name,street name,place..)

Age -->simple

2)Single-valued and **multivalued** attributes

- Example: multivalued attribute: *phone_numbers*

Age-→single valued

3)Derived attributes

- Can be computed from other attributes
- Example: age, given date_of_birth

Domain – the set of permitted values for each attribute

Complex Attributes

Attribute types:

Simple and **composite** attributes.

the attributes have been simple; that is, they have not been divided into sub parts Composite attributes, on the other hand, can be divided into sub parts (i.e., other attributes). For example, an attribute name could be structured as a composite attribute consisting of first name, middle initial, and last name.

Single-valued and **multivalued** attributes

- Example: multivalued attribute: *phone_numbers*

Derived attributes

- Can be computed from other attributes
- Example: age, given date_of_birth
- Null

Attribute types:

Single-valued and **multivalued** attributes

Example: multi valued attribute: *phone_numbers*

The attributes in our examples all have a single value for a particular entity. For instance, the student ID attribute for a specific student entity refers to only one student ID . Such attributes are said to be single valued. There may be instances where an attribute has a set of values for a specific entity. Suppose we add to the instructor entity set a phone number attribute. An instructor may have zero, one, or several phone numbers, and different instructors may have different numbers of phones. This type of attribute is said to be multivalued.

Attribute types:

Derived attributes.

The value for this type of attribute can be derived from the values of other related attributes or entities. For instance, let us say that the instructor entity set has an attribute students advised, which represents how many students an instructor advises. We can derive the value for this attribute by counting the number of student entities associated with that instructor.

©Silberschatz, Korth and Sudarshan

Null value

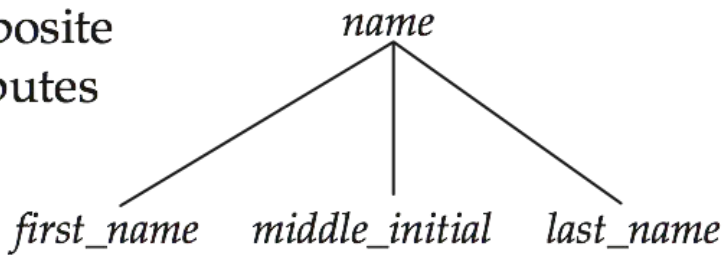
An attribute takes a null value when an entity does not have a value for it.

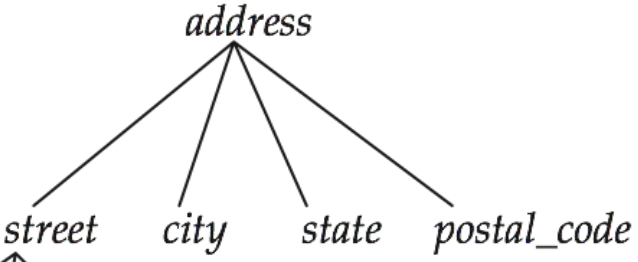
The null value may indicate “not applicable”— that is, the value does not exist for the entity. For example, a person who has no middle name may have the middle initial attribute set to null.

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).

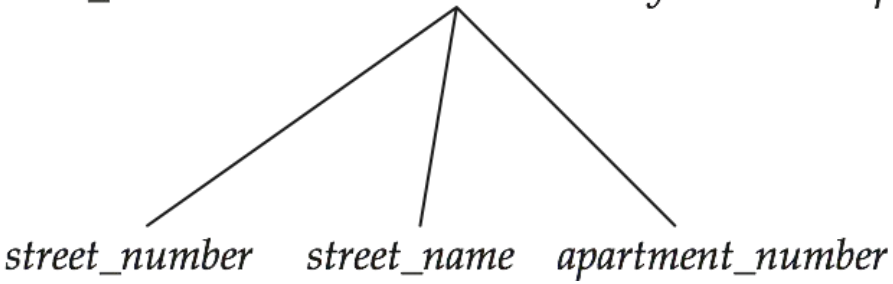
Composite Attributes

composite attributes





component attributes



E-R diagram showing entity sets

An entity set is represented in an E-R diagram by a rectangle,

The first part contains the name of the entity set.

The second part contains the names of all the attributes of the entity Set.

.1 E-R diagram showing entity sets *instructor* and *student*.

-shows two entity sets instructor and student.

The attributes associated with instructor are ID , name, and salary.

The attributes associated with student are ID , name, and tot_cred.

Attributes that are part of the primary key are underlined

E-R diagram showing entity sets

-shows two entity sets instructor and student.

The attributes associated with instructor are ID , name, and salary.

The attributes associated with student are ID , name, and tot_cred.

Attributes that are part of the primary key are underlined

©Silberschatz, Korth and Sudarshan

Entity Sets -- *instructor* and *student*

instructor_ID instructor_name

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

instructor

student-ID student_name

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

student

©Silberschatz, Korth and Sudarshan

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.

A relationship set is represented in an E-R diagram by a diamond, which is linked via lines to a number of different entity sets (rectangles).

©Silberschatz, Korth and Sudarshan

Relationship Sets

- A **relationship** is an association among several entities

  Example:
  Hayes *depositor* A-102
  customer entity relationship set *account* entity

- A **relationship set** (R)is a mathematical relation among $n \geq 2$ entities, each taken from entity sets

$$\{(e_1, e_2, \ldots e_n) \mid e_1 \in E_1, e_2 \in E_2, \ldots, e_n \in E_n\}$$

where $(e_1, e_2, \ldots, e_n)$ is a relationship

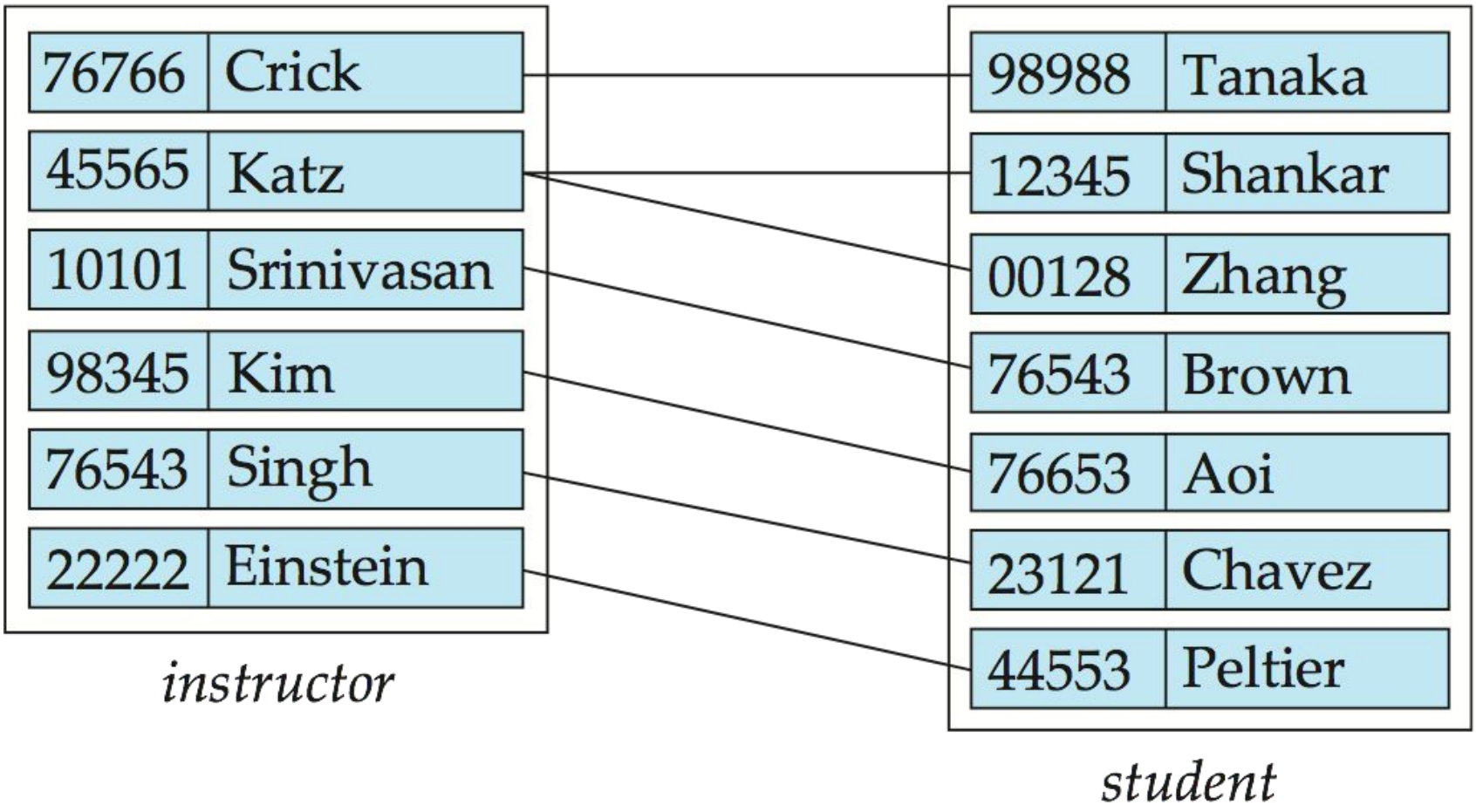
Example:

(Hayes, A-102) $\in$ *depositor*

the entity sets E 1 , E 2 , ... , E n participate in relationship set R

©Silberschatz, Korth and Sudarshan

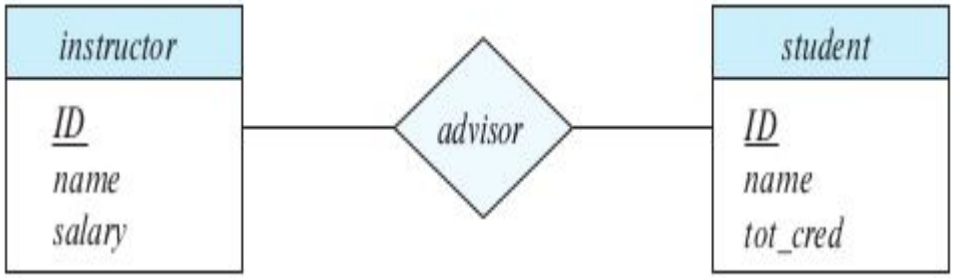
Relationship set advisor



©Silberschatz, Korth and Sudarshan

Relationship in ER diagram

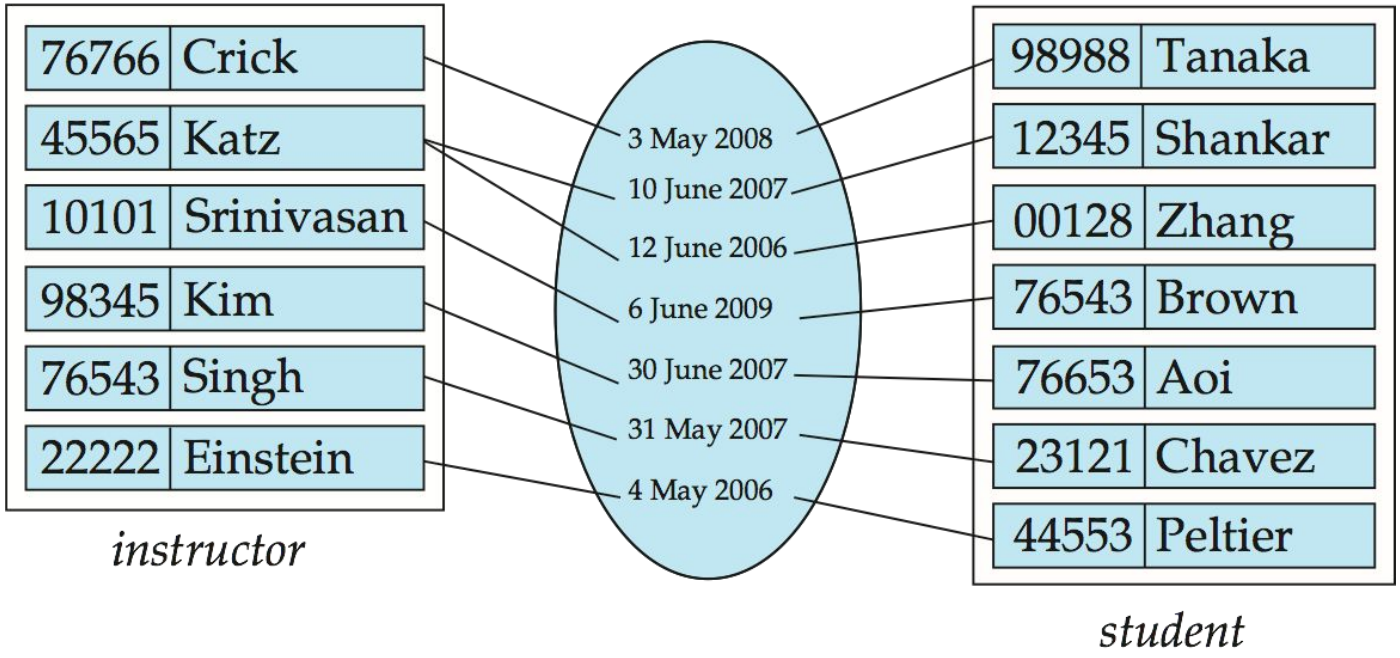
A relationship set is represented in an E-R diagram by a diamond, which is linked via lines to a number of different entity sets (rectangles).



Relationship Sets (Cont.)

An attribute can also be associated with a relationship set.

For instance, the *advisor* relationship set between entity sets *instructor* and *student* may have the attribute *date* which tracks when the student started being associated with the advisor



 ©Silberschatz, Korth and Sudarshan

Relationship instance

relationship instance in an E-R schema represents an association between the named entities in the real-world enterprise that is being modeled.

Redundant Attributes

Suppose we have entity sets:

instructor, with attributes: *ID*, *name*, *dept_name, salary*

department, with attributes: *dept_name, building, budget*

We model the fact that each instructor has an associated department using a relationship set *inst_dept*

The attribute *dept_name* appears in both entity sets. Since it is the primary key for the entity set *department*, it replicates information present in the relationship and is therefore redundant in the entity set *instructor* and needs to be removed.

BUT: when converting back to tables, in some cases the attribute gets reintroduced, as we will see later.

Degree of a Relationship Set

binary relationship

involve two entity sets (or degree two).

most relationship sets in a database system are binary.

Relationships between more than two entity sets are rare. Most relationships are binary. (More on this later.)

- Example: *students* work on research *projects* under the guidance of an *instructor*.
- relationship *proj_guide* is a ternary relationship between *instructor, student,* and *project*

Binary and Ternary Relationship

Figure 6.3 E-R diagram showing relationship set *advisor*.

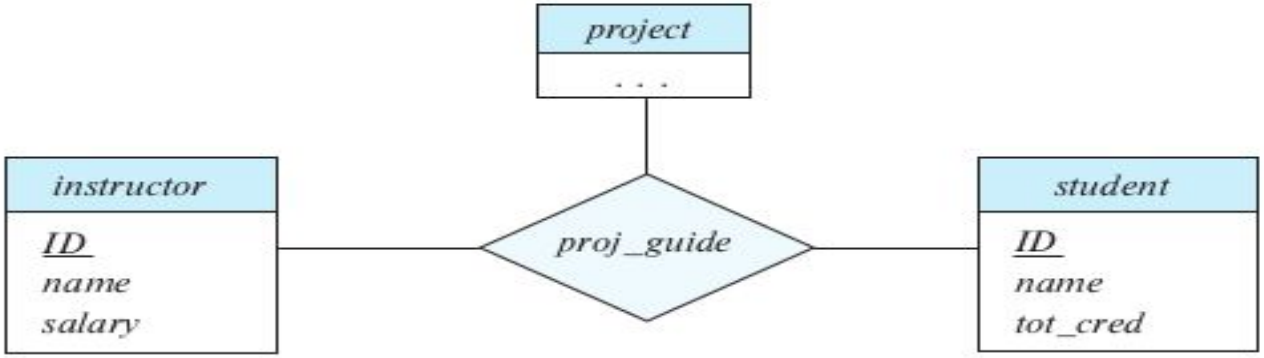


Figure 6.6 E-R diagram with a ternary relationship *proj_guide*.

Mapping Cardinality Constraints

Express the number of entities to which another entity can be associated via a relationship set.

Most useful in describing binary relationship sets.

For a binary relationship set the mapping cardinality must be one of th following types:

One to one

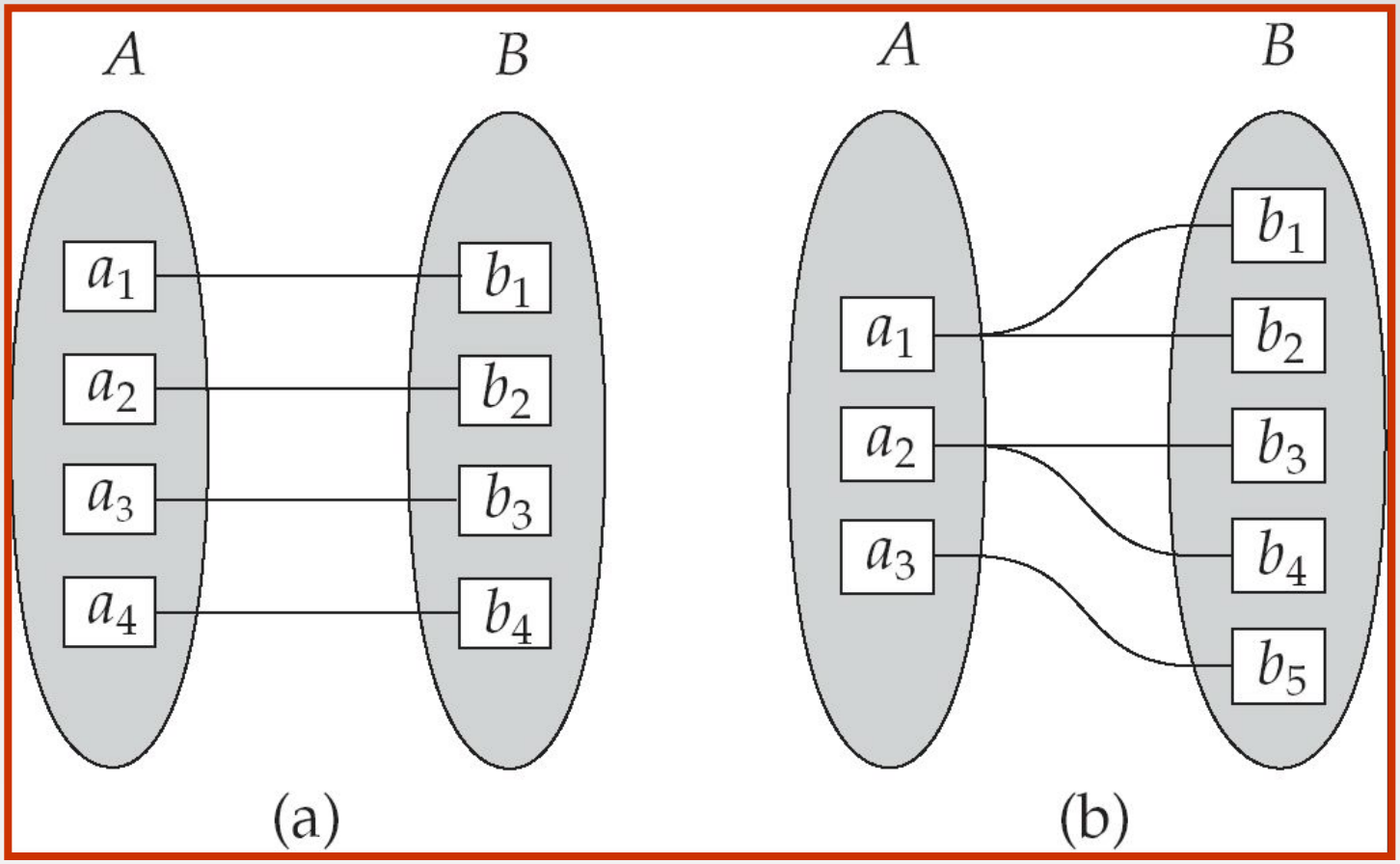
One to many

Many to one

Many to many

©Silberschatz, Korth and Sudarshan

Mapping Cardinalities



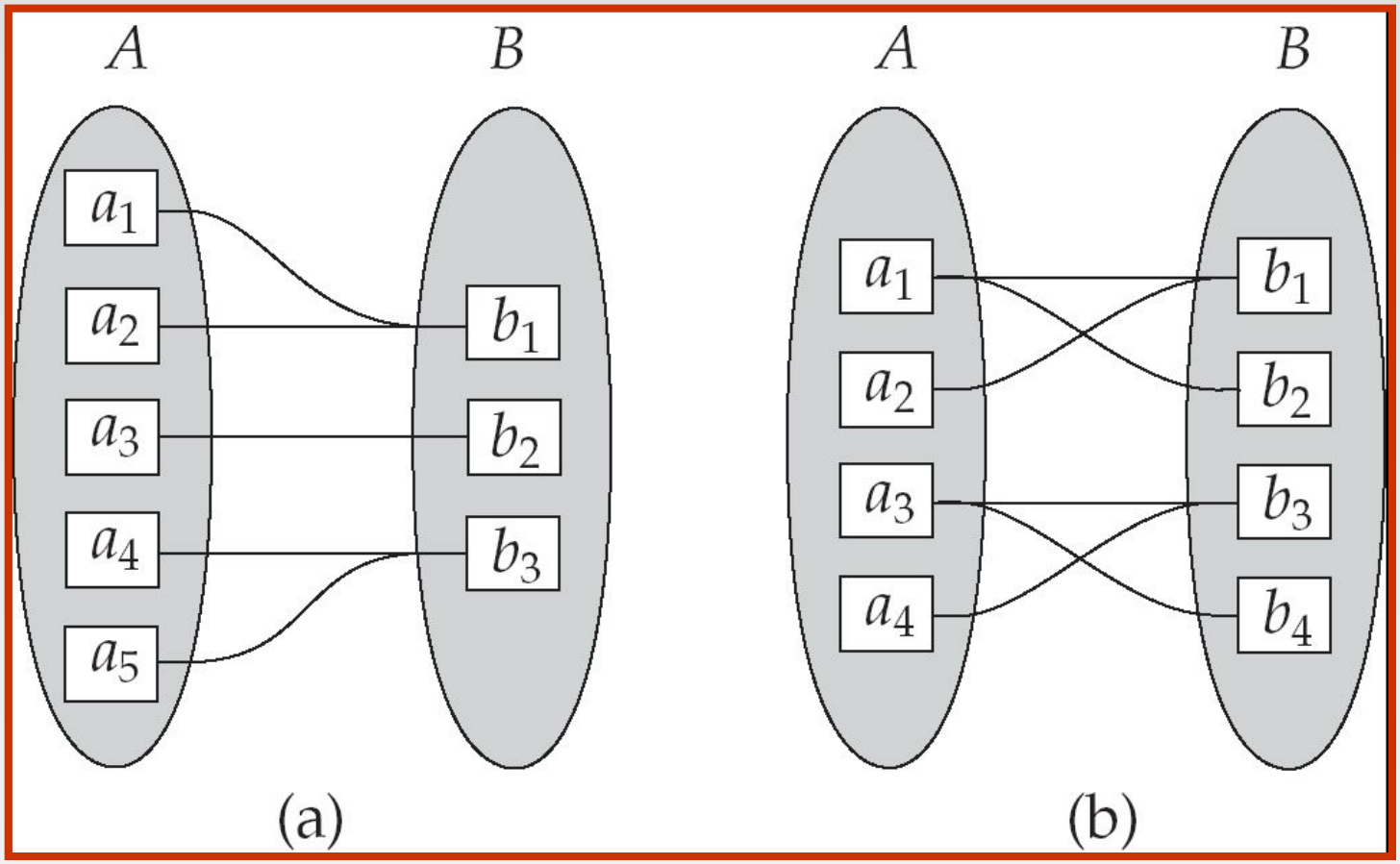
One to one

One to many

Note: Some elements in *A* and *B* may not be mapped to any elements in the other set

©Silberschatz, Korth and Sudarshan

Mapping Cardinalities



Many to one

Many to many

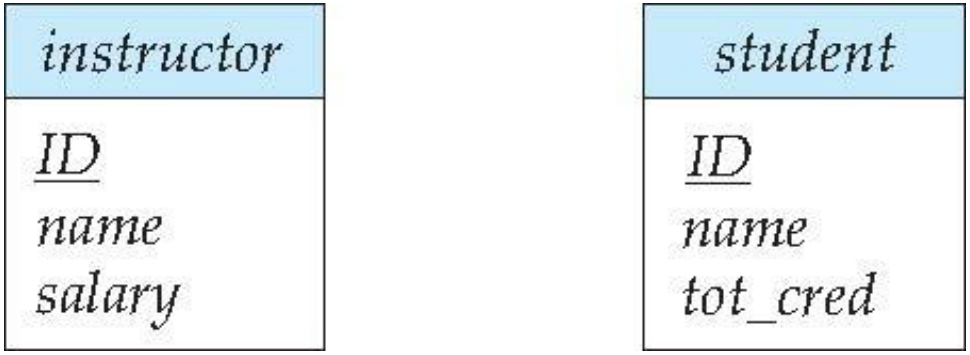
Note: Some elements in A and B may not be mapped to any elements in the other set

©Silberschatz, Korth and Sudarshan

E-R Diagrams

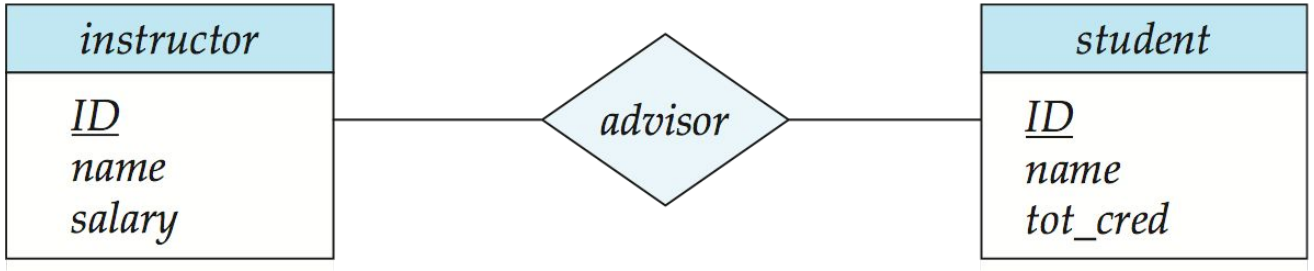
Entity Sets

- Entities can be represented graphically as follows:
  - Rectangles represent entity sets.
  - Attributes listed inside entity rectangle
  - Underline indicates primary key attributes

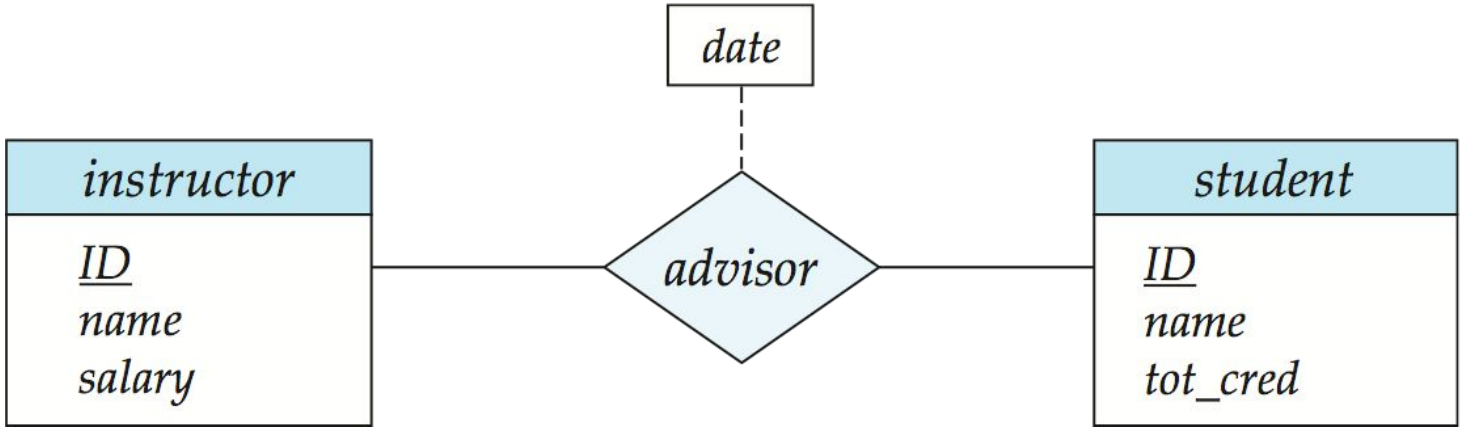


Relationship Sets

- Diamonds represent relationship sets.



Relationship Sets with Attributes

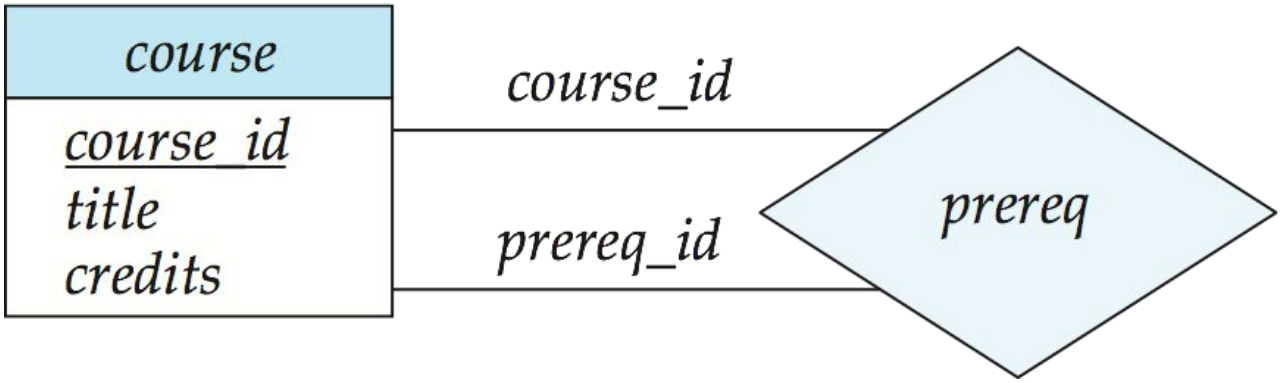


Roles

Entity sets of a relationship need not be distinct

Each occurrence of an entity set plays a “role” in the relationship

The labels “*course_id*” and “*prereq_id*” are called **roles**.



sometimes called a recursive relation-ship set, explicit role names are necessary to specify how an entity participates in a relationship instance.

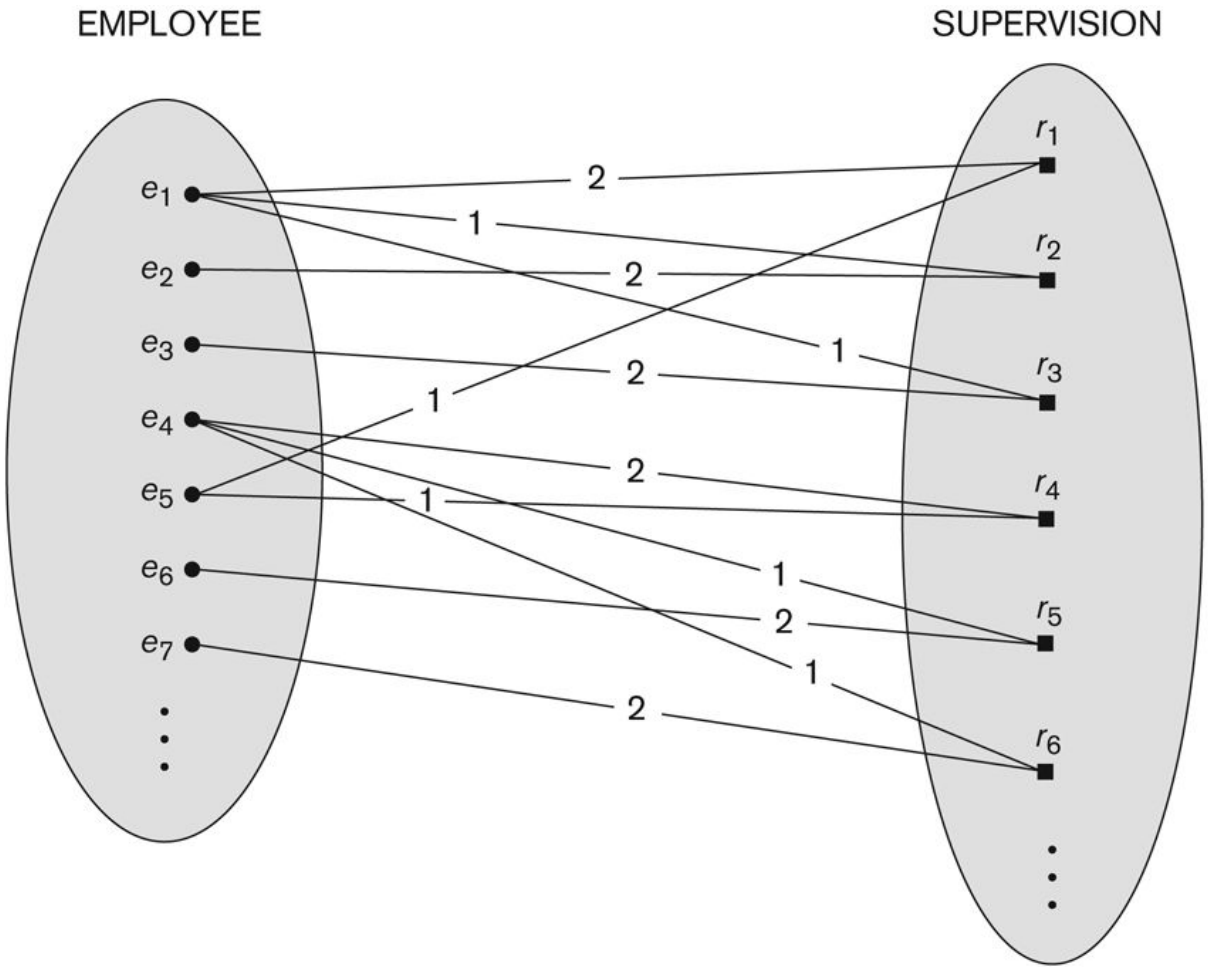


Figure 3.11
A recursive relationship SUPERVISION between EMPLOYEE in the *supervisor* role (1) and EMPLOYEE in the *subordinate* role (2).

©Silberschatz, Korth and Sudarshan

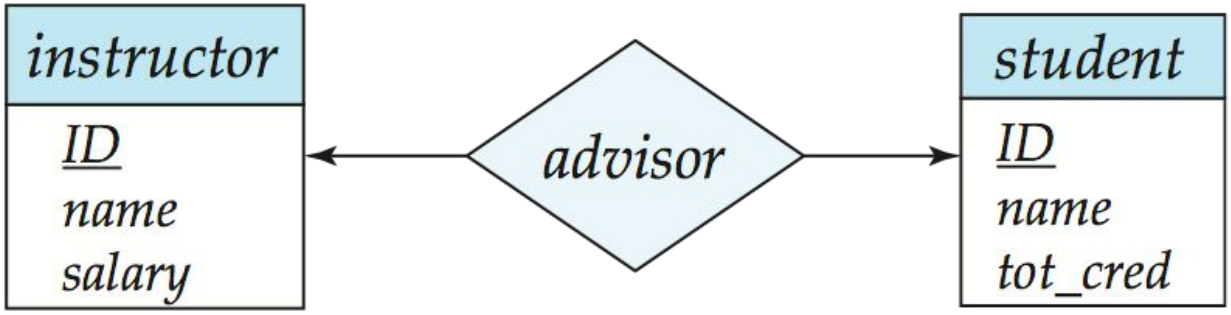
Mapping Cardinalities/Cardinality Constraints

We express cardinality constraints by drawing either a directed line (→), signifying “one,” or an undirected line (—), signifying “many,” between the relationship set and the entity set.

One-to-one relationship between an *instructor* and a *student* :

A student is associated with at most one *instructor* via the relationship *advisor*

A *student* is associated with at most one *department* via *stud_dept*

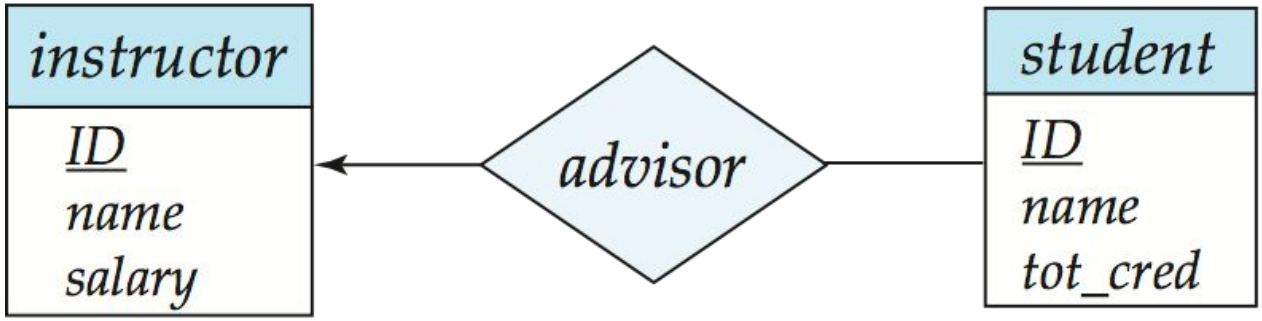


One-to-Many Relationship

one-to-many relationship between an *instructor* and a *student*

an instructor is associated with several (including 0) students via *advisor*

a student is associated with at most one instructor via advisor,

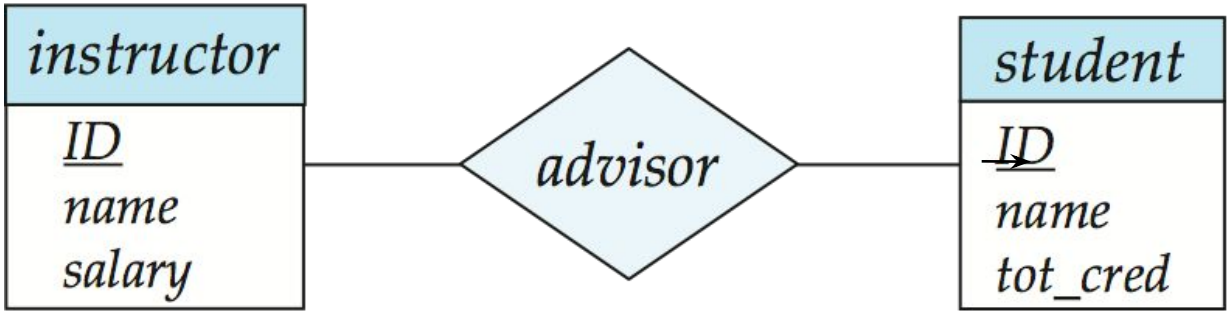


Many-to-One Relationships

In a many-to-one relationship between an *instructor* and a *student,*

an instructor is associated with at most one student via *advisor*,

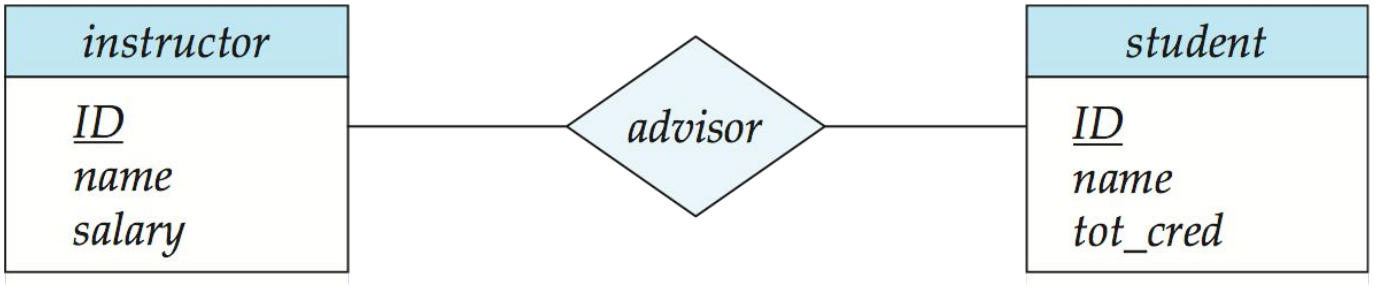
and a student is associated with several (including 0) instructors via *advisor*



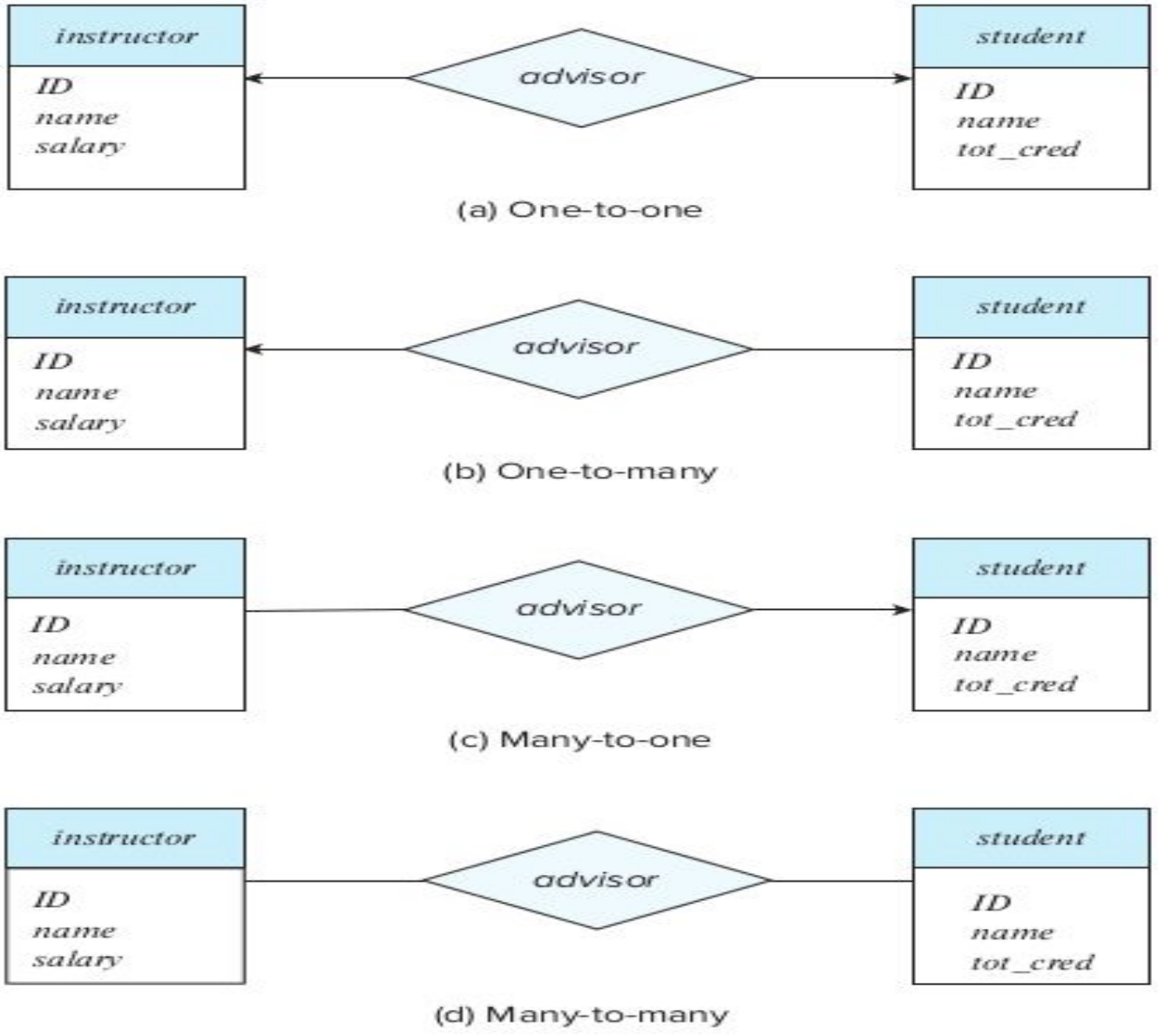
Many-to-Many Relationship

An instructor is associated with several (possibly 0) students via *advisor*

A student is associated with several (possibly 0) instructors via *advisor*

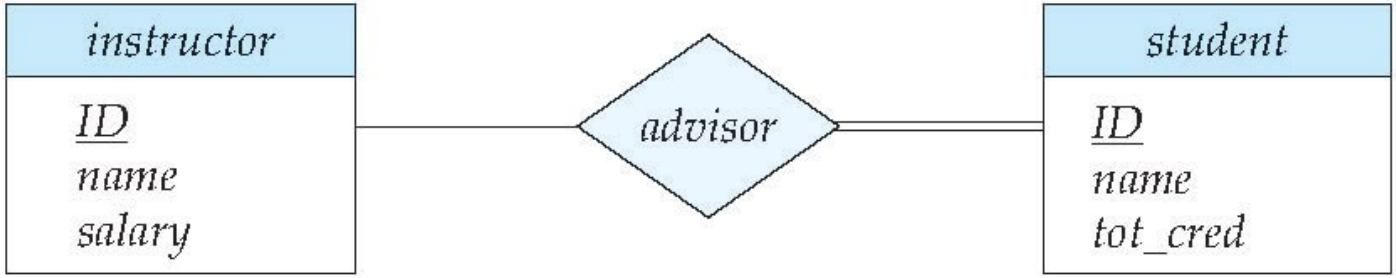


Relationship cardinalities



Total and Partial Participation

- Total participation (indicated by double line): every entity in the entity set participates in at least one relationship in the relationship set



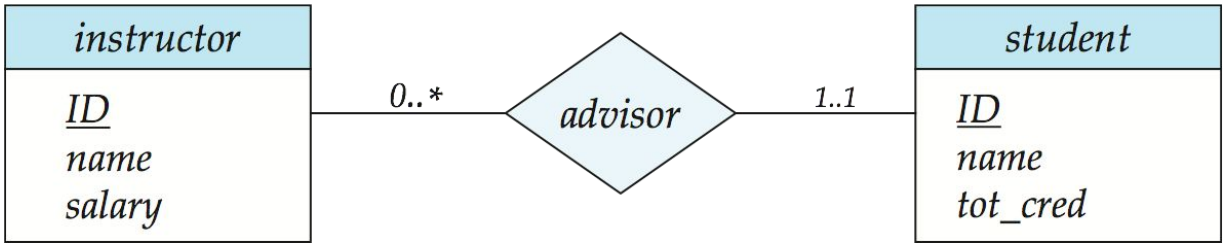
participation of *student* in *advisor* relation is total

- every *student* must have an associated instructor

- Partial participation: some entities may not participate in any relationship in the relationship set
  - Example: participation of *instructor* in *advisor* is partial

Notation for Expressing More Complex Constraints

- A line may have an associated minimum and maximum cardinality, shown in the form *l..h*, where *l* is the minimum and *h* the maximum cardinality
  - A minimum value of 1 indicates total participation.
  - A maximum value of 1 indicates that the entity participates in at most one relationship
  - A maximum value of * indicates no limit.



Instructor can advise 0 or more students. A student must have 1 advisor; cannot have multiple advisors

Cardinality Constraints on Ternary Relationship

- We allow at most one arrow out of a ternary (or greater degree) relationship to indicate a cardinality constraint
- For example, an arrow from *proj_guide* to *instructor* indicates each student has at most one guide for a project
- If there is more than one arrow, there are two ways of defining the meaning.
  - For example, a ternary relationship *R* between *A*, *B* and *C* with arrows to *B* and *C* could mean
    1. Each *A* entity is associated with a unique entity from B and *C* or
    2. Each pair of entities from (*A*, *B*) is associated with a unique *C* entity, and each pair (*A*, *C*) is associated with a unique *B*
  - Each alternative has been used in different formalisms
  - To avoid confusion we outlaw more than one arrow

©Silberschatz, Korth and Sudarshan

Keys

A **super key** of an entity set is a set of one or more attributes whose values uniquely determine each entity.

A **candidate key** of an entity set is a minimal super key

Customer_id is candidate key of *customer*

account_number is candidate key of *account*

Although several candidate keys may exist, one of the candidate keys is selected to be the **primary key**.

Primary key for Entity Sets

- By definition, individual entities are distinct.
- From database perspective, the differences among them must be expressed in terms of their attributes.
- The values of the attribute values of an entity must be such that they can uniquely identify the entity.
  - 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

Primary Key for Relationship Sets

- To distinguish among the various relationships of a relationship set we use the individual primary keys of the entities in the relationship set.
  - Let *R* be a relationship set involving entity sets E1, E2, .. En
  - The primary key for R is consists of the union of the primary keys of entity sets E1, E2, ..En
  - If the relationship set *R* has attributes a1, a2, .., am associated with it, then the primary key of *R* also includes the attributes a1, a2, .., am
- Example: relationship set "advisor".
  - The primary key consists of *instructor.ID* and *student.ID*
- The choice of the primary key for a relationship set depends on the mapping cardinality of the relationship set.

Choice of Primary key for Binary Relationship

- Many-to-Many relationships. The preceding union of the primary keys is a minimal superkey and is chosen as the primary key.
- One-to-Many relationships . The primary key of the "Many" side is a minimal superkey and is used as the primary key.
- Many-to-one relationships. The primary key of the "Many" side is a minimal superkey and is used as the primary key.
- One-to-one relationships. The primary key of either one of the participating entity sets forms a minimal superkey, and either one can be chosen as the primary key.

©Silberschatz, Korth and Sudarshan

Weak Entity Sets

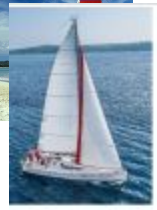
- Consider a *section* entity, which is uniquely identified by a *course_id*, *semester, year*, and *sec_id*.
- Clearly, section entities are related to course entities. Suppose we create a relationship set *sec_course* between entity sets *section* and *course*.
- Note that the information in *sec_course* is redundant, since *section* already has an attribute *course_id*, which identifies the course with which the section is related.
- One option to deal with this redundancy is to get rid of the relationship *sec_course*; however, by doing so the relationship between *section* and *course* becomes implicit in an attribute, which is not desirable.

©Silberschatz, Korth and Sudarshan

Weak Entity Sets (Cont.)

- An alternative way to deal with this redundancy is to not store the attribute *course_id* in the *section* entity and to only store the remaining attributes *section_id*, *year*, and *semester.*
  - However, the entity set *section* then does not have enough attributes to identify a particular *section* entity uniquely
- To deal with this problem, we treat the relationship *sec_course* as a special relationship that provides extra information, in this case, the *course_id*, required to identify *section* entities uniquely.
- A **weak entity set** is one whose existence is dependent on another entity, called its **identifying entity**
- Instead of associating a primary key with a weak entity, we use the identifying entity, along with extra attributes called **discriminator** to uniquely identify a weak entity.

©Silberschatz, Korth and Sudarshan



Weak Entity Sets (Cont.)

- An entity set that is not a weak entity set is termed a **strong entity set**.
- Every weak entity must be associated with an identifying entity; that is, the weak entity set is said to be **existence dependent** on the identifying entity set.
- The identifying entity set is said to **own** the weak entity set that it identifies.
- The relationship associating the weak entity set with the identifying entity set is called the **identifying relationship**.
- Note that the relational schema we eventually create from the entity set *section* does have the attribute *course_id*, for reasons that will become clear later, even though we have dropped the attribute *course_id* from the entity set *section*.

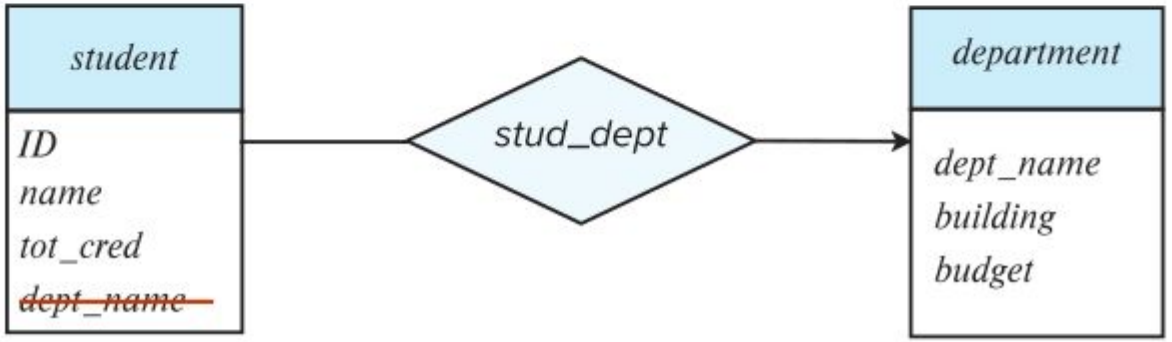
Expressing Weak Entity Sets

- In E-R diagrams, a weak entity set is depicted via a double rectangle.
- We underline the discriminator of a weak entity set with a dashed line.
- The relationship set connecting the weak entity set to the identifying strong entity set is depicted by a double diamond.
- Primary key for *section* – (*course_id, sec_id, semester, year*)

©Silberschatz, Korth and Sudarshan

Removing Redundant Attributes

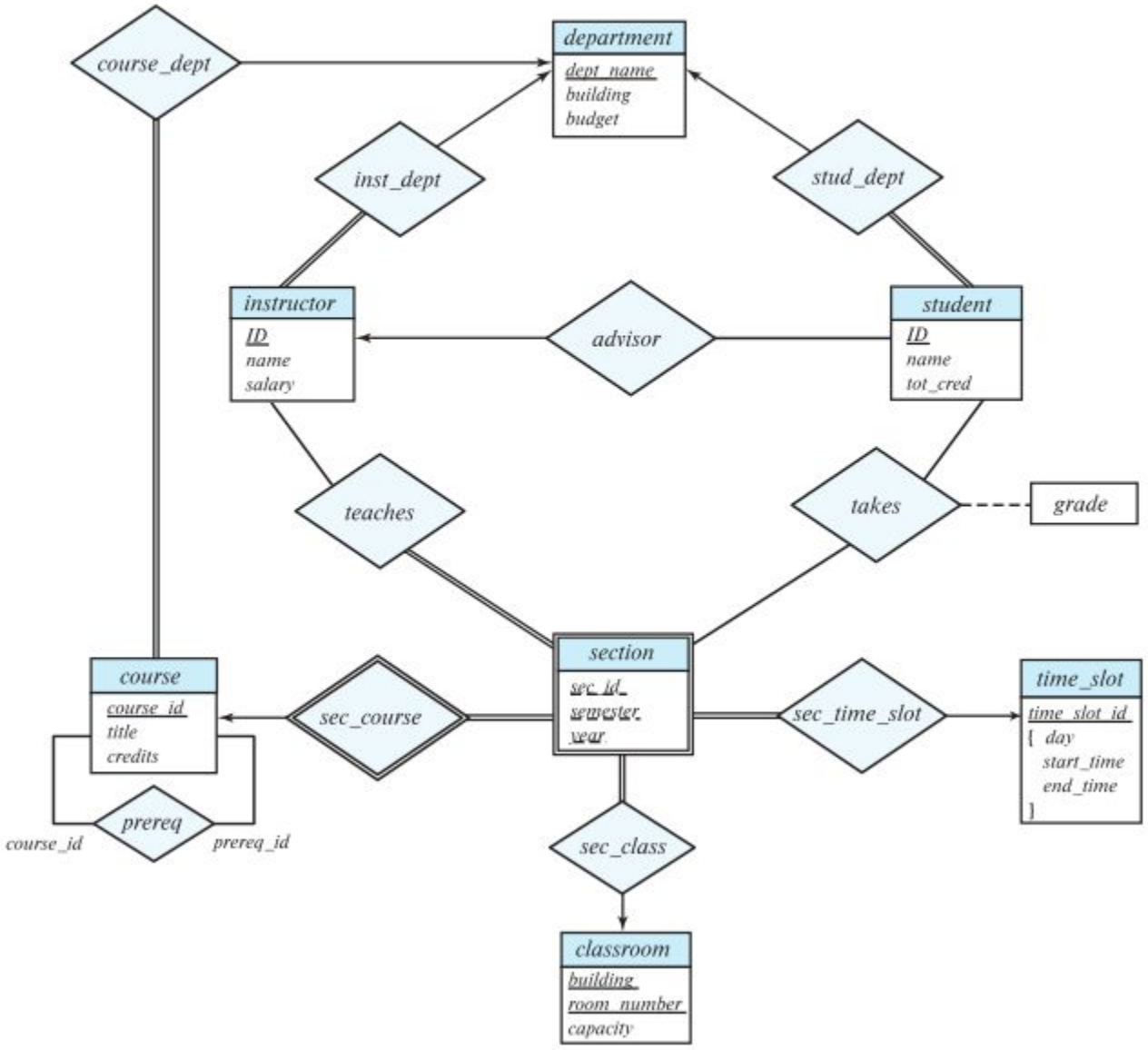
- Suppose we have entity sets:
  - *student*, with attributes: *ID*, *name*, *tot_cred*, *dept_name*
  - *department*, with attributes: *dept_name*, *building*, *budget*
- We model the fact that each student has an associated department using a relationship set *stud_dept*
- The attribute *dept_name* in *student* below replicates information present in the relationship and is therefore redundant
  - and needs to be removed.
- BUT: when converting back to tables, in some cases the attribute gets reintroduced, as we will see later.



(a) Incorrect use of attribute

©Silberschatz, Korth and Sudarshan

E-R Diagram for a University Enterprise



Reduction to Relation Schemas

- Entity sets and relationship sets can be expressed uniformly as *relation schemas* that represent the contents of the database.
- A database which conforms to an E-R diagram can be represented by a collection of schemas.
- For each entity set and relationship set there is a unique schema that is assigned the name of the corresponding entity set or relationship set.
- Each schema has a number of columns (generally corresponding to attributes), which have unique names.

©Silberschatz, Korth and Sudarshan

Representing Entity Sets

- A strong entity set reduces to a schema with the same attributes

  student(<u>ID</u>, name, tot_cred)

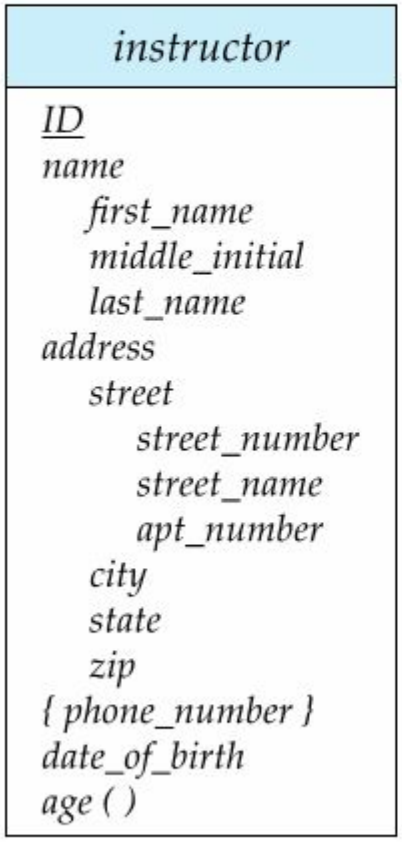
- A weak entity set becomes a table that includes a column for the primary key of the identifying strong entity set

  section (<u>course_id, sec_id, sem, year</u>)

- Example

©Silberschatz, Korth and Sudarshan

Representation of Entity Sets with Composite Attributes



- Composite attributes are flattened out by creating a separate attribute for each component attribute
  - Example: given entity set *instructor* with composite attribute *name* with component attributes *first_name* and *last_name* the schema corresponding to the entity set has two attributes *name_first_name* and *name_last_name*
    - Prefix omitted if there is no ambiguity (*name_first_name* could be *first_name)*
- Ignoring multivalued attributes, extended instructor schema is
  - *instructor(ID, first_name, middle_initial, last_name, street_number, street_name, apt_number, city, state, zip_code, date_of_birth)*

©Silberschatz, Korth and Sudarshan

Representation of Entity Sets with Multivalued Attributes

- A multivalued attribute *M* of an entity *E* is represented by a separate schema *EM*
- Schema *EM* has attributes corresponding to the primary key of *E* and an attribute corresponding to multivalued attribute *M*
- Example: Multivalued attribute *phone_number* of *instructor* is represented by a schema:
  inst_phone= (*ID*, *phone_number*)
- Each value of the multivalued attribute maps to a separate tuple of the relation on schema *EM*
  - For example, an *instructor* entity with primary key 22222 and phone numbers 456-7890 and 123-4567 maps to two tuples:
    (22222, 456-7890) and (22222, 123-4567)

©Silberschatz, Korth and Sudarshan

Representation of Weak Entity Sets

Let A be a weak entity set with attributes $a_1, a_2, \ldots, a_m$. Let B be the strong entity set on which A depends. Let the primary key of B consist of attributes $b_1, b_2, \ldots, b_n$. We represent the entity set A by a relation schema called A with one attribute for each member of the set:

$$\{a_1, a_2, \ldots, a_m\} \cup \{b_1, b_2, \ldots, b_n\}$$

As an illustration, consider the weak entity set *section* in the E-R diagram of Figure 6.15. 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*)

©Silberschatz, Korth and Sudarshan

Representing Relationship Sets

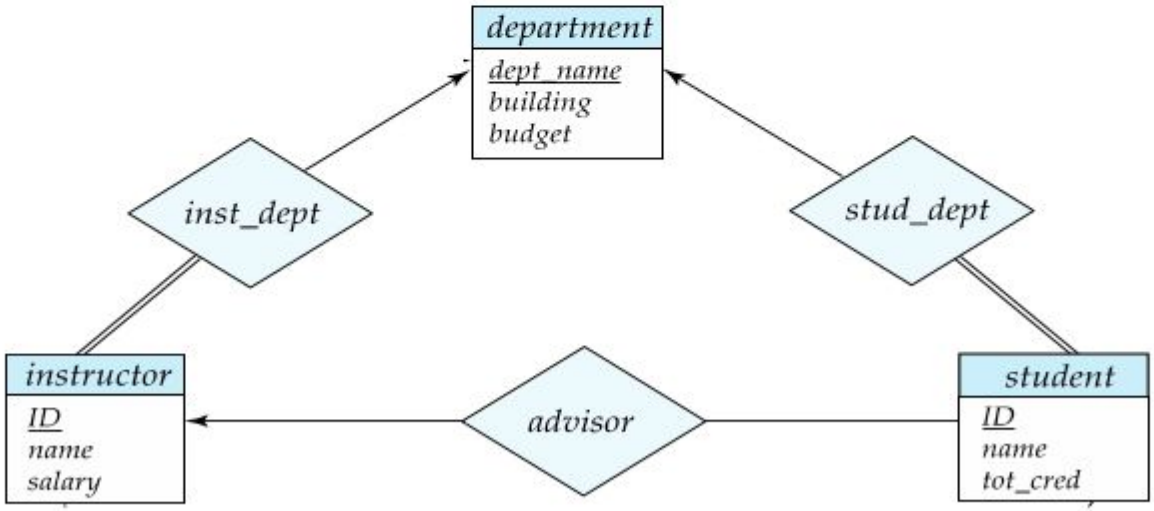
- A many-to-many relationship set is represented as a schema with attributes for the primary keys of the two participating entity sets, and any descriptive attributes of the relationship set.
- Example: schema for relationship set *advisor*

advisor = (*s_id*, *i_id*)

©Silberschatz, Korth and Sudarshan

Redundancy of Schemas

- Many-to-one and one-to-many relationship sets that are total on the many-side can be represented by adding an extra attribute to the "many" side, containing the primary key of the "one" side
- Example: Instead of creating a schema for relationship set *inst_dept*, add an attribute *dept_name* to the schema arising from entity set *instructor*
- Example



Redundancy of Schemas (Cont.)

- For one-to-one relationship sets, either side can be chosen to act as the "many" side
  - That is, an extra attribute can be added to either of the tables corresponding to the two entity sets
- If participation is *partial* on the "many" side, replacing a schema by an extra attribute in the schema corresponding to the "many" side could result in null values

©Silberschatz, Korth and Sudarshan

Redundancy of Schemas (Cont.)

- The schema corresponding to a relationship set linking a weak entity set to its identifying strong entity set is redundant.
- Example: The *section* schema already contains the attributes that would appear in the *sec_course* schema

Extended E-R Features

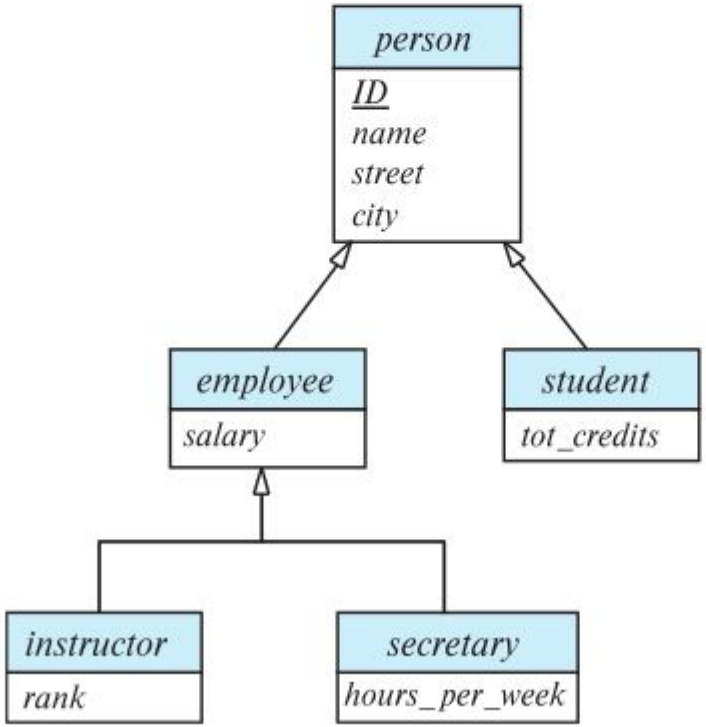
Specialization

- Top-down design process; we designate sub-groupings within an entity set that are distinctive from other entities in the set.
- These sub-groupings become lower-level entity sets that have attributes or participate in relationships that do not apply to the higher-level entity set.
- Depicted by a *triangle* component labeled ISA (e.g., *instructor* "is a" *person*).
- **Attribute inheritance** – a lower-level entity set inherits all the attributes and relationship participation of the higher-level entity set to which it is linked.

©Silberschatz, Korth and Sudarshan

Specialization Example

- **Overlapping** – *employee* and *student*
- **Disjoint** – *instructor* and *secretary*
- Total and partial



Representing Specialization as Schemas (Cont.)

- Method 2:
  - Form a schema for each entity set with all local and inherited attributes

schema	attributes
person	ID, name, street, city
student	ID, name, street, city, tot_cred
employee	ID, name, street, city, salary

  - Drawback: *name, street* and *city* may be stored redundantly for people who are both students and employees

©Silberschatz, Korth and Sudarshan

Representing Specialization via Schemas

- Method 1:
  - Form a schema for the higher-level entity
  - Form a schema for each lower-level entity set, include primary key of higher-level entity set and local attributes

schema	attributes
person	ID, name, street, city
student	ID, tot_cred
employee	ID, salary

  - Drawback: getting information about, an *employee* requires accessing two relations, the one corresponding to the low-level schema and the one corresponding to the high-level schema

©Silberschatz, Korth and Sudarshan

Generalization

- **A bottom-up design process** – combine a number of entity sets that share the same features into a higher-level entity set.
- Specialization and generalization are simple inversions of each other; they are represented in an E-R diagram in the same way.
- The terms specialization and generalization are used interchangeably.

Completeness constraint

- **Completeness constraint** -- 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 a generalization.
  - **total**: an entity must belong to one of the lower-level entity sets
  - **partial**: an entity need not belong to one of the lower-level entity sets

©Silberschatz, Korth and Sudarshan

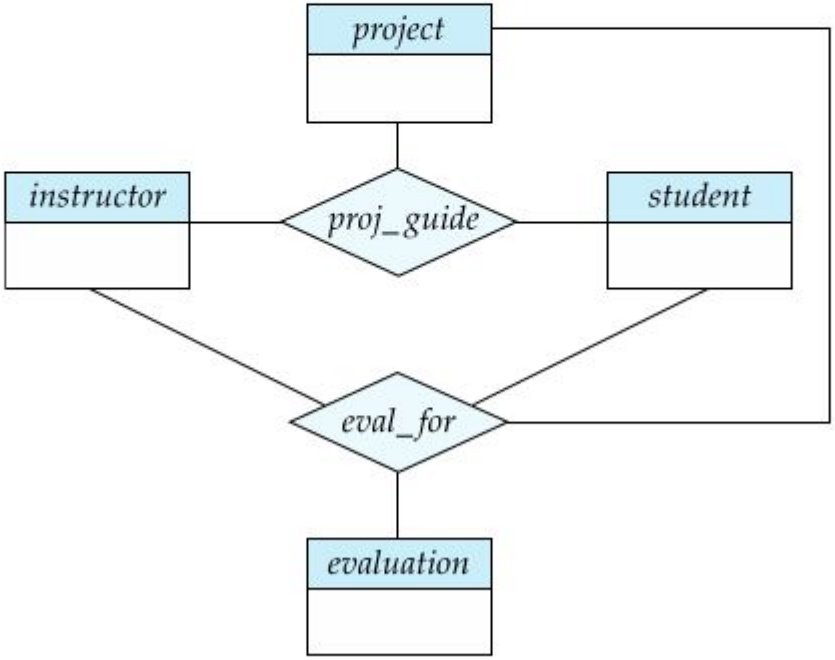
Completeness constraint (Cont.)

- Partial generalization is the default.
- We can specify total generalization in an ER diagram by adding the keyword **total** in the diagram and drawing a dashed line from the keyword to the corresponding hollow arrow-head to which it applies (for a total generalization), or to the set of hollow arrow-heads to which it applies (for an overlapping generalization).
- The *student* generalization is total: All student entities must be either graduate or undergraduate. Because the higher-level entity set arrived at through generalization is generally composed of only those entities in the lower-level entity sets, the completeness constraint for a generalized higher-level entity set is usually total

©Silberschatz, Korth and Sudarshan

Aggregation

- Consider the ternary relationship *proj_guide*, which we saw earlier
- Suppose we want to record evaluations of a student by a guide on a project



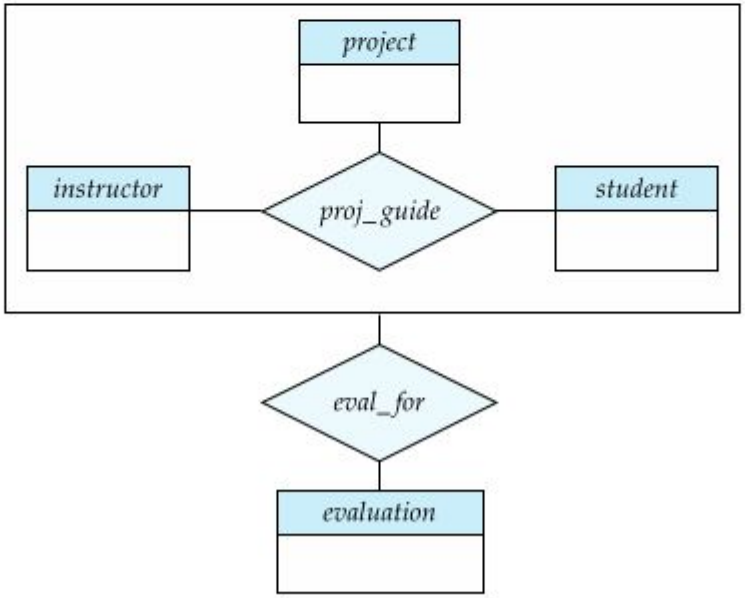
Aggregation (Cont.)

- Relationship sets *eval_for* and *proj_guide* represent overlapping information
  - Every *eval_for* relationship corresponds to a *proj_guide* relationship
  - However, some *proj_guide* relationships may not correspond to any *eval_for* relationships
    - So we can't discard the *proj_guide* relationship
- Eliminate this redundancy via *aggregation*
  - Treat relationship as an abstract entity
  - Allows relationships between relationships
  - Abstraction of relationship into new entity

©Silberschatz, Korth and Sudarshan

Aggregation (Cont.)

- Eliminate this redundancy via *aggregation* without introducing redundancy, the following diagram represents:
  - A student is guided by a particular instructor on a particular project
  - A student, instructor, project combination may have an associated evaluation



Reduction to Relational Schemas

- To represent aggregation, create a schema containing
  - Primary key of the aggregated relationship,
  - The primary key of the associated entity set
  - Any descriptive attributes
- In our example:
  - The schema *eval_for* is:

    eval_for (*s_ID, project_id, i_ID, evaluation_id*)
  - The schema *proj_guide* is redundant.

©Silberschatz, Korth and Sudarshan