DSA5104
Principles of Data Management and Retrieval

Lecture 5: Entity-Relationship Model

Recap

- Aggregation with Null Values
- Nested Subqueries
  - Test for Empty Relations - **exists / not exists**
  - Correlation Name
  - Test for Absence of Duplicate Tuples - **unique / not unique**
  - **with** Clause
  - Scalar Subqueries
- Modification of the Database
  - **delete / insert / update**
    - **Case** Statement for Conditional Updates
- Join Expressions
  - Natural join
  - Inner join
  - Outer join (left, right, full)
- Integrity Constraints
  - **Not null / unique / check(P)**
  - FK constraint

How to Use a DBMS?

- Answer: SQL queries!
- Problem: Only if someone has already defined the schema!
- Question: How hard could that be? (Just define some tables and columns…)

How to Design a Database?

Not a database system

Steps in Database Design

- **Requirements Analysis**
  - Data needs; what must database do?
  - For small applications with requirements *fully* understood, db designers may decide the relations (attributes and constraints) to create directly
  - For real-world applications with high complexity, db designers need to interact extensively with domain experts and prospective users of the application to gain the full picture
- **Conceptual Design**
  - High level description (often done w/ER model)
- **Logical Design**
  - Translate ER into DBMS data model
- **Schema Refinement**
  - Consistency, normalization
- **Physical Design** - Indexes, disk layout
- **Security Design** - Who accesses what, and how

Steps in Database Design

- **Requirements Analysis**
  - User needs; what must database do?
- **Conceptual Design**
  - High level description (often done w/ER model)
  - Choose a data model, apply the concepts of the chosen data model to translate user requirements into a conceptual schema of the database
  - E.g., Entity-Relationship (ER) model → Specify entities (attributes) and their relationships, as well as constraints on entities and relationships.
- **Logical Design**
  - Translate ER into DBMS data model
- **Schema Refinement**
  - Consistency, normalization
- **Physical Design** - Indexes, disk layout
- **Security Design** - Who accesses what, and how

Steps in Database Design

- **Requirements Analysis**
  - User needs; what must database do?
- **Conceptual Design**
  - High level description (often done w/ER model)
- **Logical Design**
  - Translate ER into DBMS data model
  - Map the high-level conceptual schema onto the implementation data model of the database system
  - E.g., Map the conceptual schema defined using Entity-Relationship Model (ER Diagram) into a relation schema based on relational model (Schema Diagram)
- **Schema Refinement**
  - Consistency, normalization
- **Physical Design** - Indexes, disk layout
- **Security Design** - Who accesses what, and how

Steps in Database Design

- **Requirements Analysis**
  - User needs; what must database do?
- **Conceptual Design**
  - High level description (often done w/ER model)
- **Logical Design**
  - Translate ER into DBMS data model
- **Schema Refinement**
  - Consistency, normalization
- **Physical Design** - Indexes, disk layout
- **Security Design** - Who accesses what, and how

Steps in Database Design

- **Requirements Analysis**
  - User needs; what must database do?
- **Conceptual Design**
  - High level description (often done w/ER model)
- **Logical Design**
  - Translate ER into DBMS data model
- **Schema Refinement**
  - Consistency, normalization
- **Physical Design** - Indexes, disk layout
- **Security Design** - Who accesses what, and how

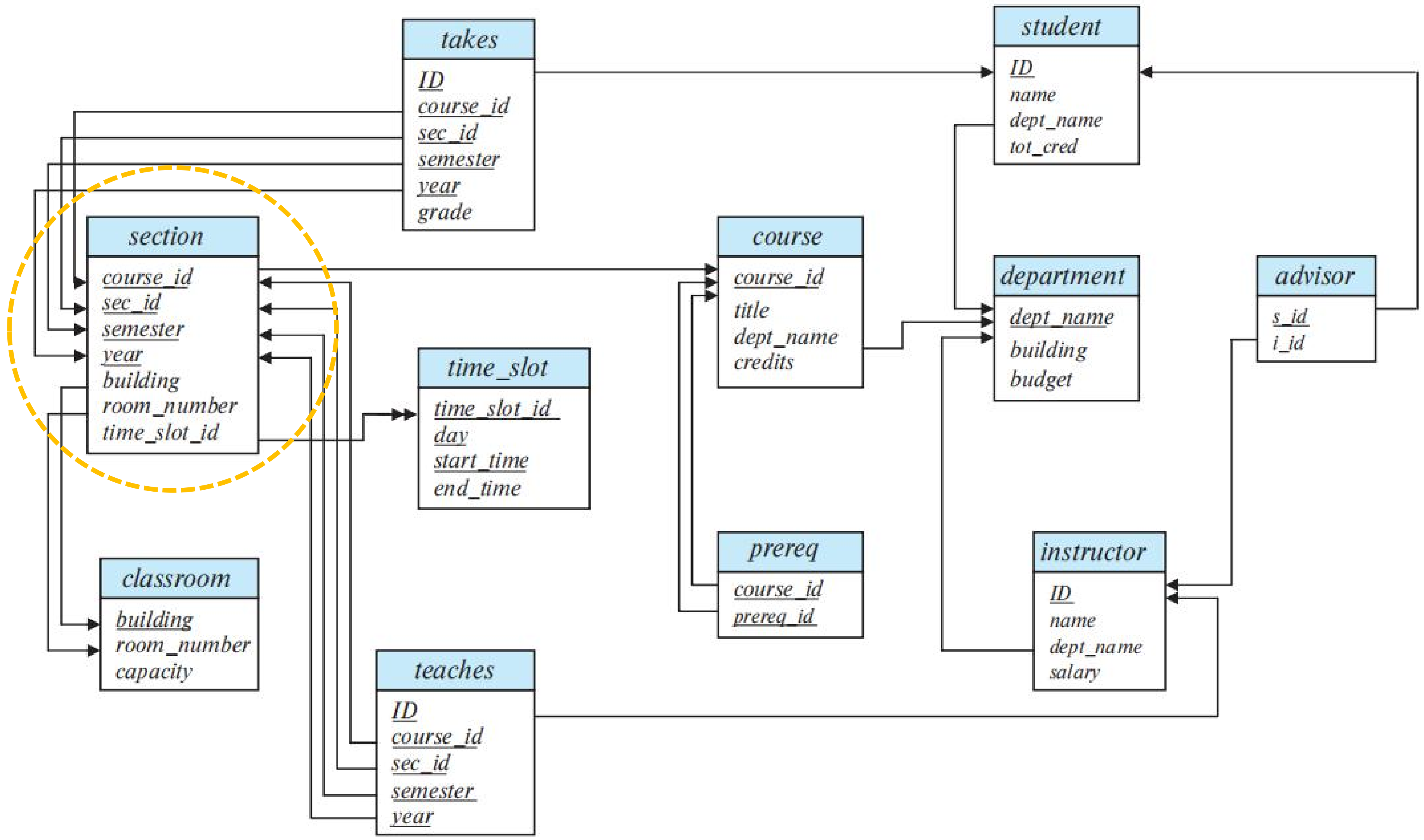
Design Alternatives

- In designing a database schema, we must ensure that we avoid two major pitfalls:
  - **Redundancy**: a bad design may result in repeat information.
    - Redundant representation of information may lead to data inconsistency among the various copies of information
    - E.g., store course name repeatedly in *section* relation
  - **Incompleteness**: a bad design may make certain aspects of the enterprise difficult or impossible to model.
    - E.g., maintain all course info in *section* relation without *course* relation → Hard to store info for new course unless offered (or, use null values)
    - Lose info of a particular course if all sections are deleted.

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

$$\begin{tikzcd}\text{Customer} \arrow[r, leftrightarrow] & \text{Sale} \arrow[r, leftrightarrow] & \text{Product}\end{tikzcd}$$

Schema Diagram for University Database



Design Approaches

- Entity Relationship (ER) Model
  - Models an enterprise as a collection of *entities* and *relationships*
    - **Entity**: a “thing” or “object” in the enterprise that is distinguishable from other objects
      - Described by a set of *attributes*
    - **Relationship**: an association among several entities
  - Represented diagrammatically by an *entity-relationship diagram*

- Normalization Theory
  - Formalize what designs are bad, and test for them

Describing data - Data Models

- **Data Model** - collection of *concepts* for describing data.

- **Schema -** description of a particular collection of data, using a given data model.

- **Relational model**
  - Main concepts - *relation* (table), attributes and tuples
  - Every relation has a schema
    - Describes the attributes
    - Specifies attribute names and domains

- **Entity-Relationship (ER) model**
  - Main concepts - entity set, relationship set & attributes
  - Schema - ER diagram (visual schema)

Data Models

- **Relational model** is a great formalism
  - Clean & common
  - But a bit detailed for design time
  - A bit fussy for brainstorming
  - Hard to communicate to “customers”

- **Entity-Relationship (ER) model** - a graphical “shim” over relational model
  - Translates to relational
  - Handy for design
  - Visual
  - Slightly higher level

Steps in Database Design

- **Requirements Analysis**
  - User needs; what must database do?
- **Conceptual Design**
  - High level description (often done w/ER model)
- **Logical Design**
  - Translate ER into DBMS data model
- **Schema Refinement**
  - Consistency, normalization
- **Physical Design** - Indexes, disk layout
- **Security Design** - Who accesses what, and how

Conceptual Design

- ER model fits in at the conceptual design step of database design

- What are the entities and relationships?
  - And what info about E’s & R’s should be in DB?

- What integrity constraints (“business rules”) hold? i.e.,
  - What must be true of these entities?
  - What must be true of the way entities relate to each other?

Outline of the ER Model

ER Model - Database Modeling

- 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 data model employs three basic concepts:
    - Entity sets,
    - Relationship sets,
    - Attributes.

- 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

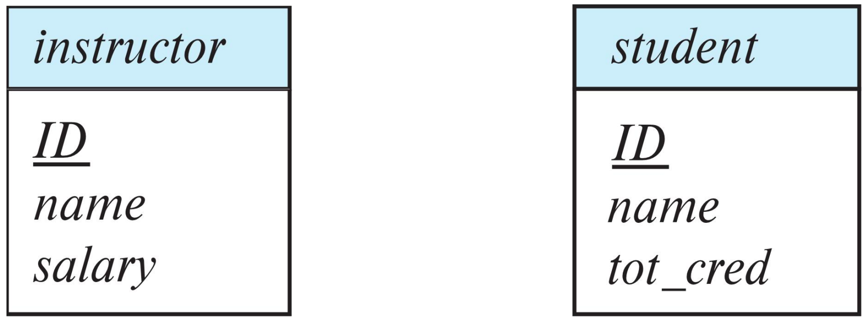
- An entity is represented by a set of **attributes**; i.e., descriptive properties possessed by all members of an entity set.
  - Example:

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

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

Representing Entity sets in ER Diagram

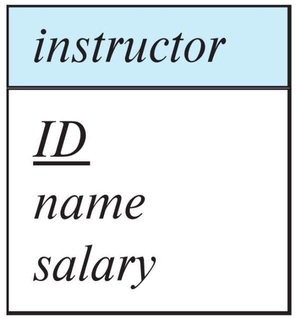
- Entity sets can be represented graphically as follows:
  - Rectangles represent entity sets.
  - Attributes listed inside entity rectangle
  - Underline indicates primary key attributes

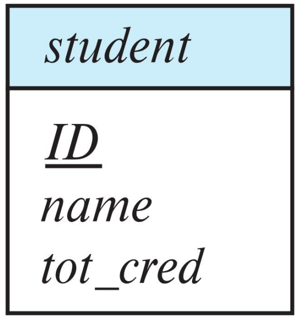


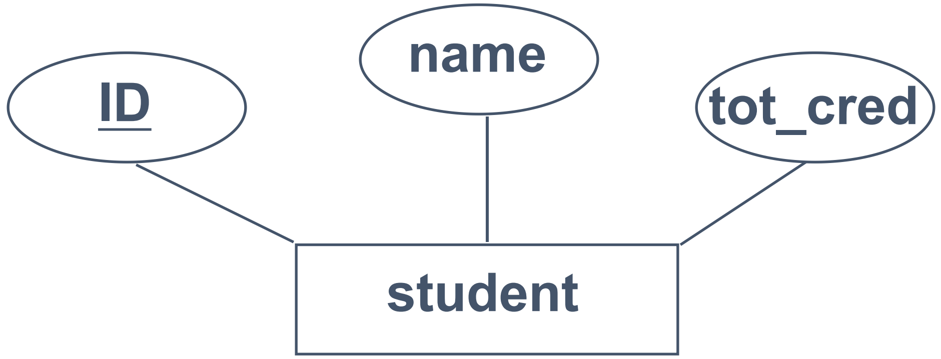
  - **Extension** of the entity set - the actual collection of entities belonging to the entity set.
    - Extension: entity set → Instance: relation

Representing Entity sets in ER Diagram

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







- **Extension** of the entity set - the actual collection of entities belonging to the entity set.
  - Extension: entity set → Instance: relation

Relationship Sets

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

  Example:

  Instructor Crick (ID = 76766, instructor entity) is an *advisor* (relationship) of to student Tanaka (ID = 98988, student entity).

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

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

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

  - Example:

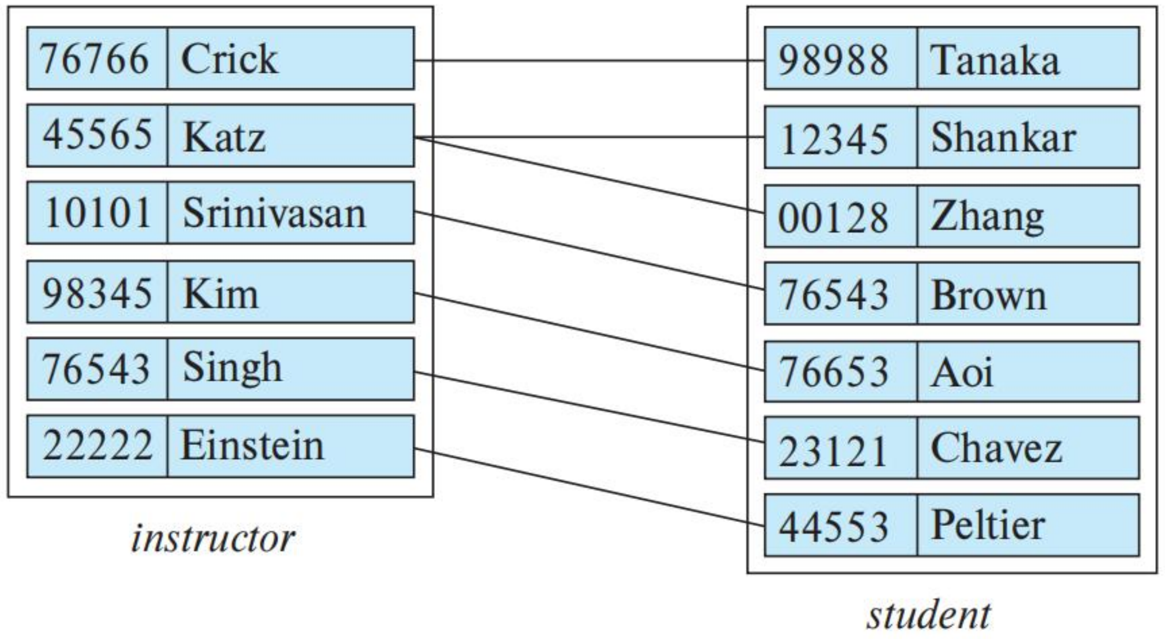
    $(98988, 76766) \in \textit{advisor}$

- **Participation** - the entity sets $E_1, E_2, \dots, E_n$ participate in relationship set R.

Relationship Sets (Cont.)

- *Example*: we define the relationship set *advisor* to denote the associations between students and the instructors who act as their advisors.

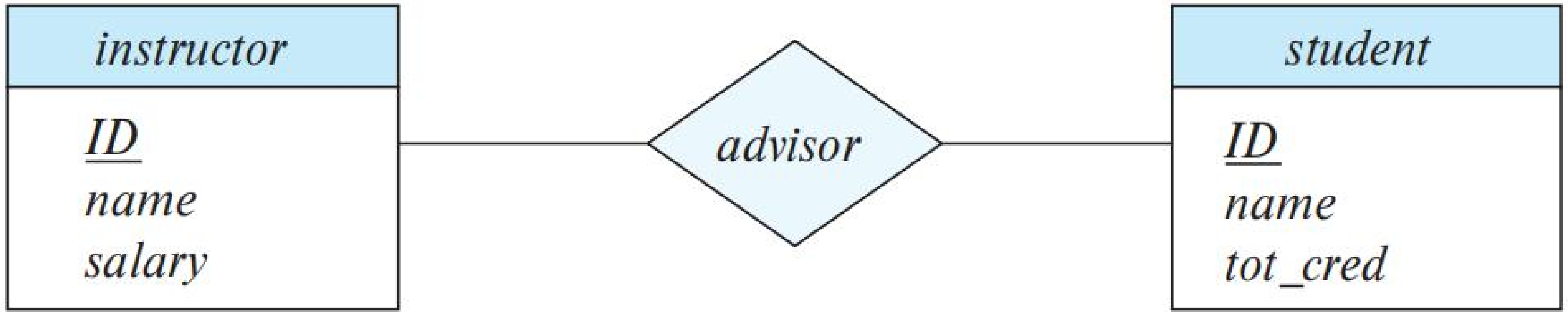
- Pictorially, we draw a line between related entities.



The *instructor* entity Crick (ID = 76766) and the *student* entity Tanaka (ID = 98988) **participates** in a **relationship instance** of *advisor.*

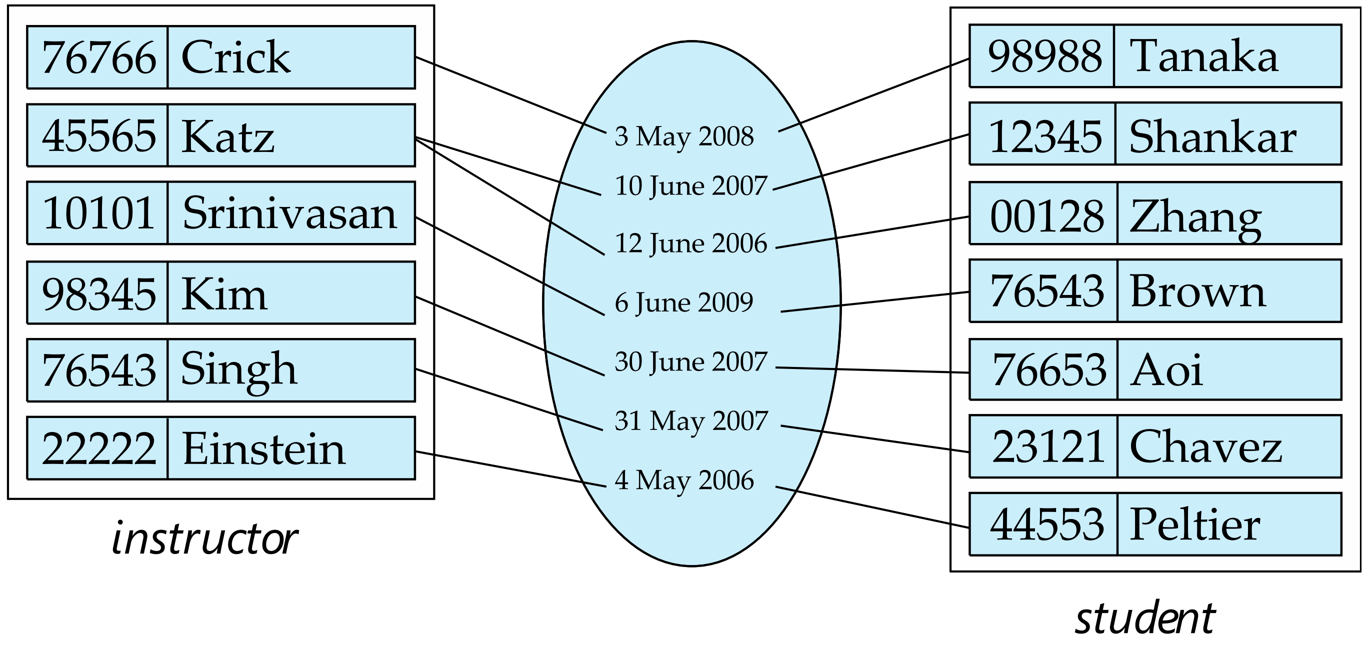
Representing Relationship Sets via ER Diagram

- Diamonds represent relationship sets.

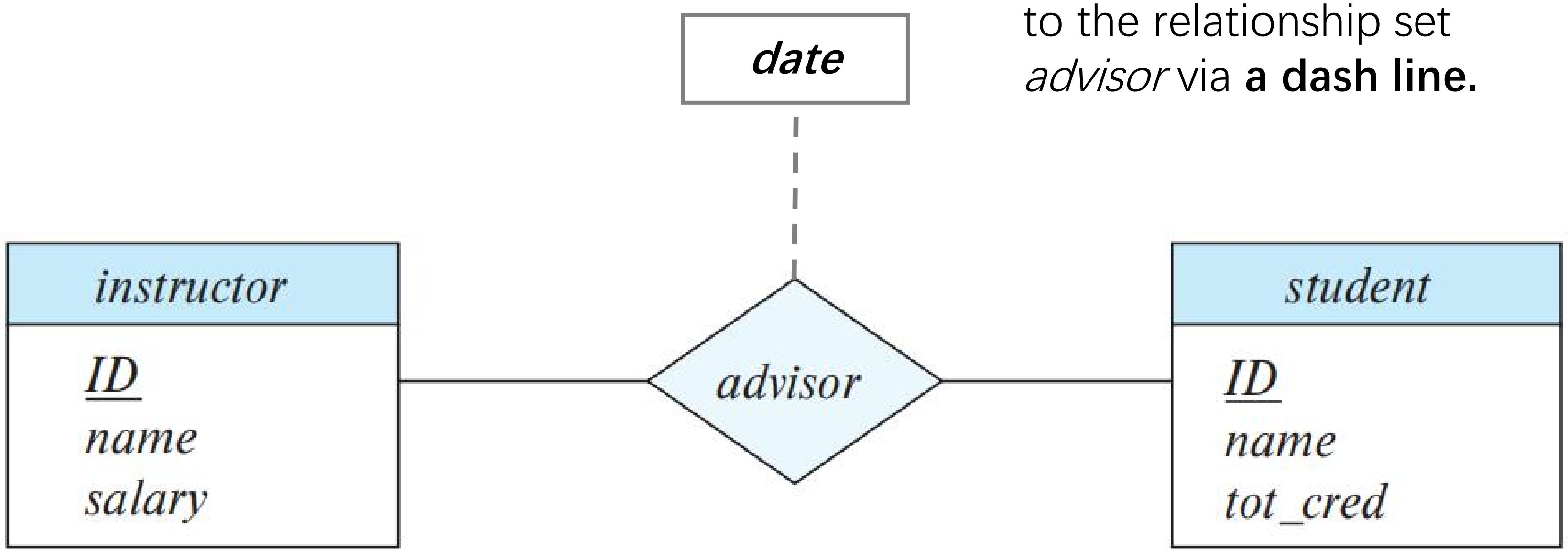


Relationship Sets (Cont.)

- 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



Relationship Sets with Attributes in ER Diagram



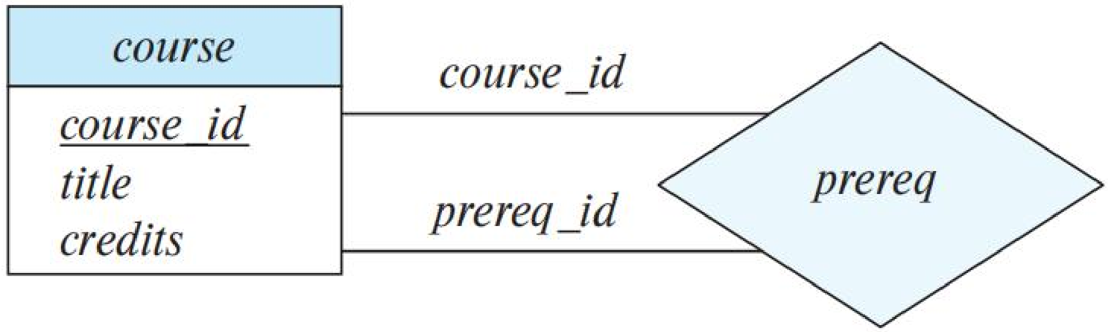
Attribute date is connected to the relationship set *advisor* via **a dash line.**

Roles

- **Role** - the function that an entity plays in a relationship

- Entity sets of a relationship need not be distinct
  - The *same* entity set participates in a relationship set more than once, in different roles.
  - Sometimes called a ‘recursive’ relationship set
  - Needs to specify the role names explicitly

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



Degree of a Relationship Set

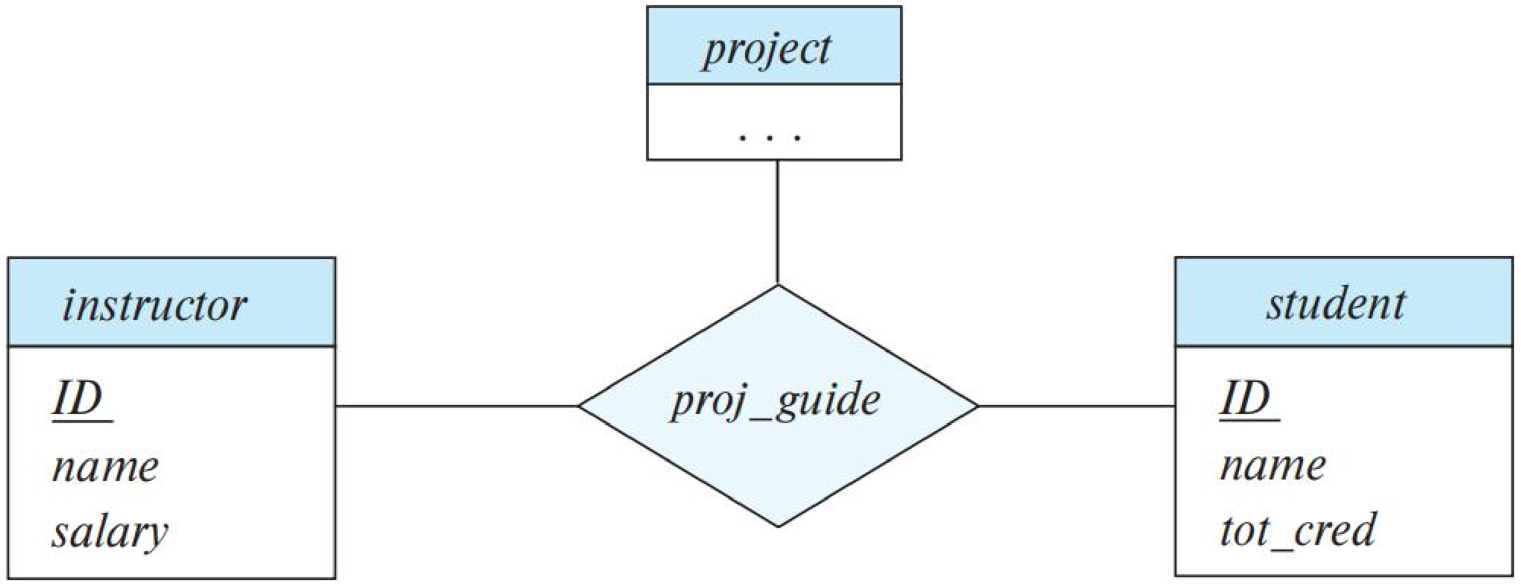
- Binary relationship
  - Involve two entity sets (or degree two).
  - Most relationship sets in a database system are binary.
  - E.g., binary relationship sets:
    - *advisor* between *student* & *instructor*
    - *takes* between *student* & *section*

- Relationships between more than two entity sets are rare.
  - Example: *students* work on research *projects* under the guidance of an *instructor*.
  - Relationship *proj_guide* is a ternary relationship between *instructor, student,* and *project*

Non-binary Relationship Sets

- There are occasions when it is more convenient to represent relationships as non-binary.
  - E.g., *proj_guide*

- ER Diagram with a Ternary Relationship



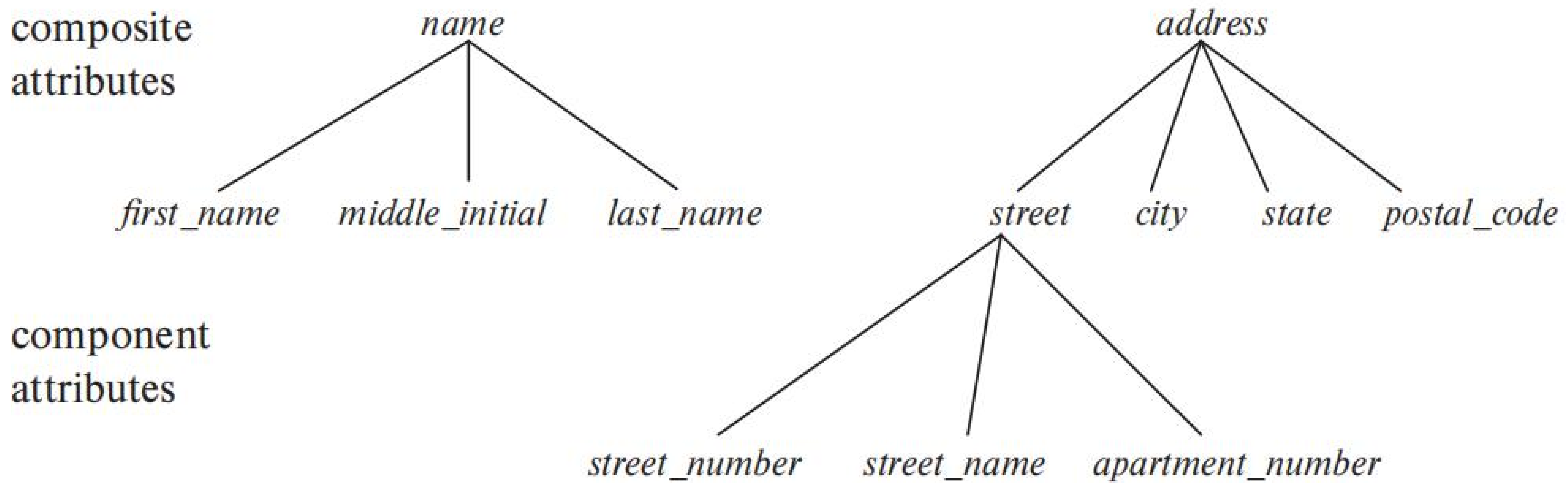
Complex Attributes

- Attribute types:
  - **Simple** and **composite** attributes.
  - **Single-valued** and **multivalued** attributes
    - Example: multivalued attribute: *phone_numbers*
  - **Derived** attributes
    - Can be computed from other attributes
    - Example: *age* could be calculated given *date_of_birth* and the current date
    - Example: *no_students_advised* could be calculated from *student.ID* of *advisor*
    - The value of a derived attribute could be computed when required.

- **Domain** (or **value set**) – the set of permitted values for each attribute

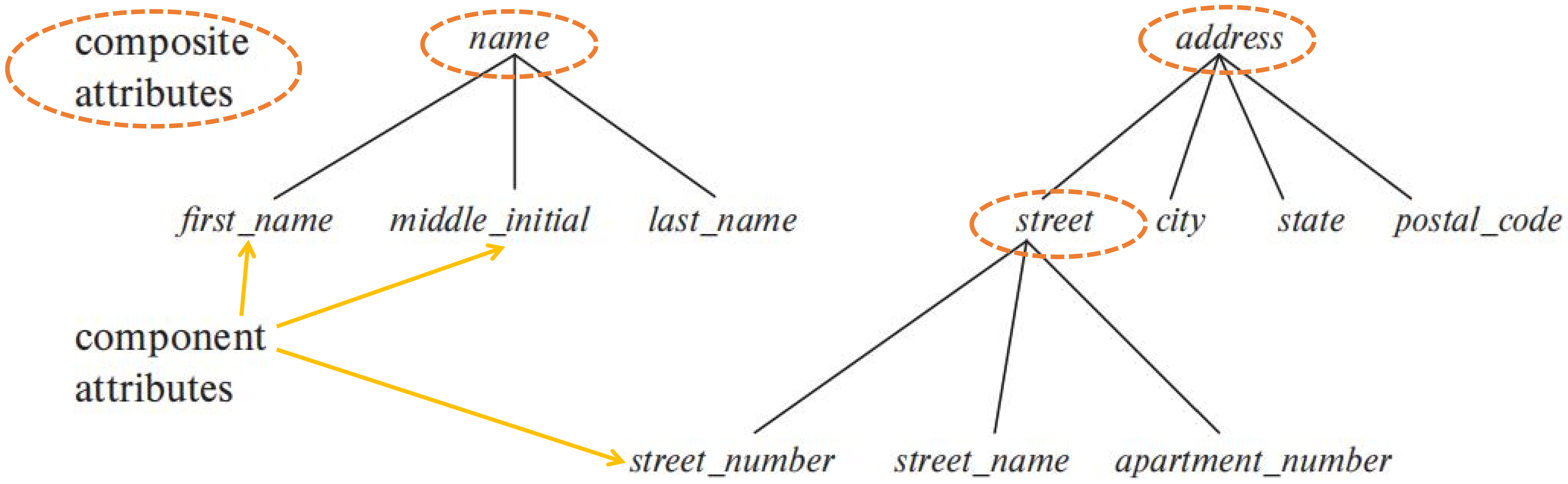
Composite Attributes

- Composite attributes allow us to divided attributes into subparts (other attributes).

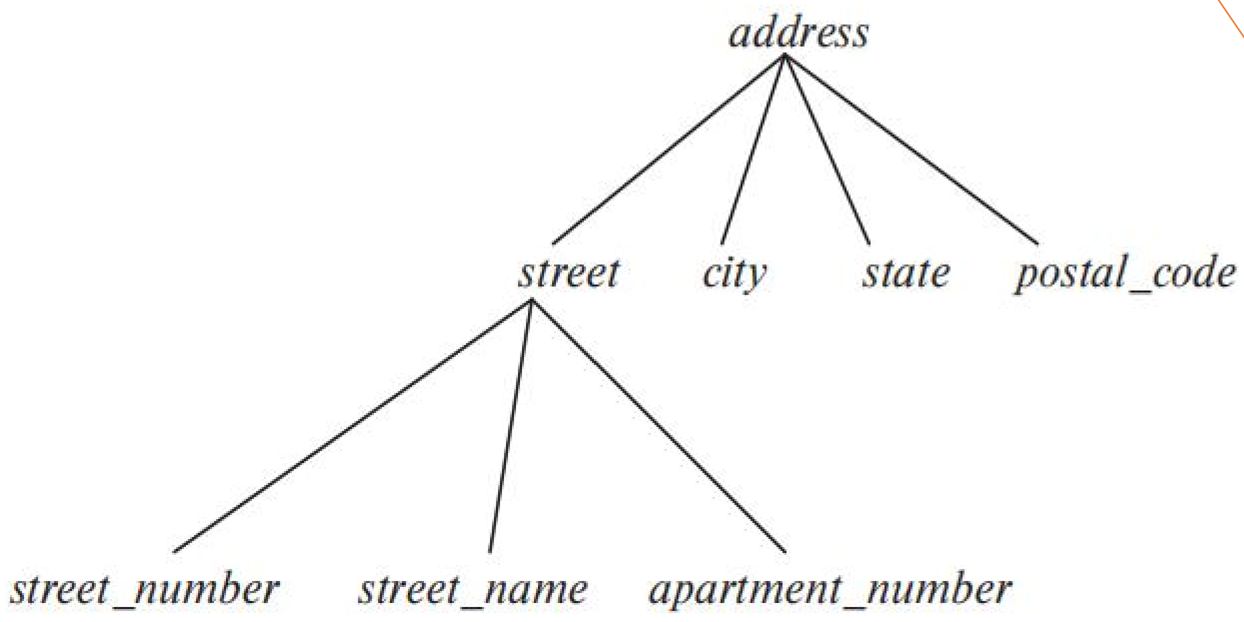


Composite Attributes

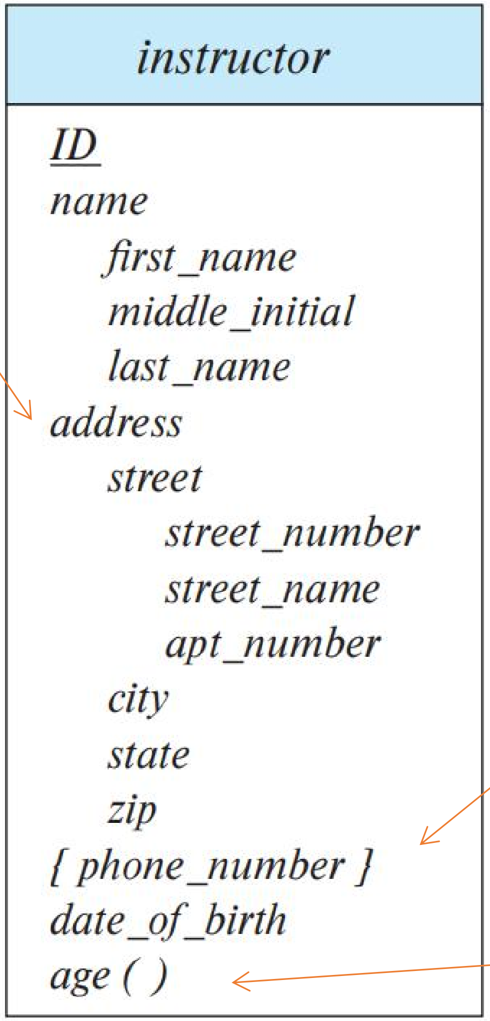
- Composite attributes allow us to divided attributes into subparts (other attributes).



Representing Complex Attributes in ER Diagram



A composite attribute *address*

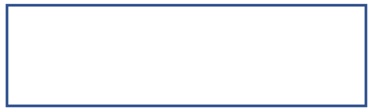


A multivalued attribute denoted as "{phone_number}"

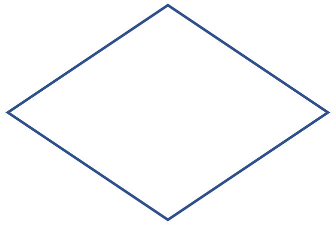
A derived attribute, *"age* ( )"

Quick Check

- In ER diagrams, a rectangle is:
  - Entity Set? Relationship Set? Attribute?



- In ER diagrams, a diamond is:
  - Entity Set? Relationship Set? Attribute?



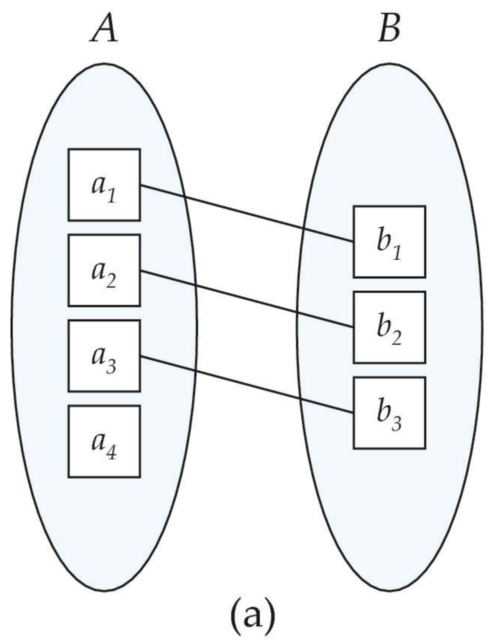
- Which of the following can exist on its own?
  - **Entity Set**
  - Relationship Set
  - Attribute

Mapping Cardinality Constraints

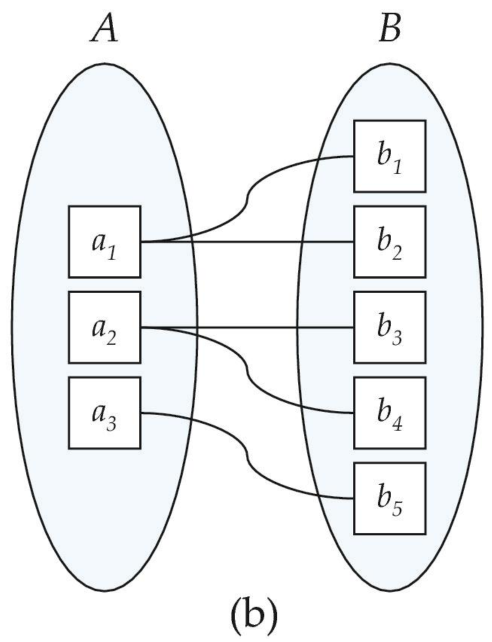
- **Mapping Cardinalities** (or cardinality ratios)
  - Express the number of entities to which another entity can be associated via a relationship set.
  - Most useful in describing binary relationship sets.

- For a binary relationship set, the mapping cardinality must be one of the following types:
  - One to one
  - One to many
  - Many to one
  - Many to many

Mapping Cardinalities



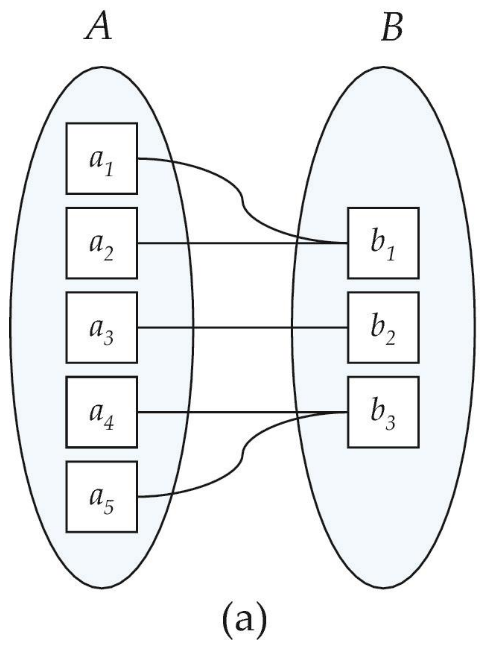
One to one



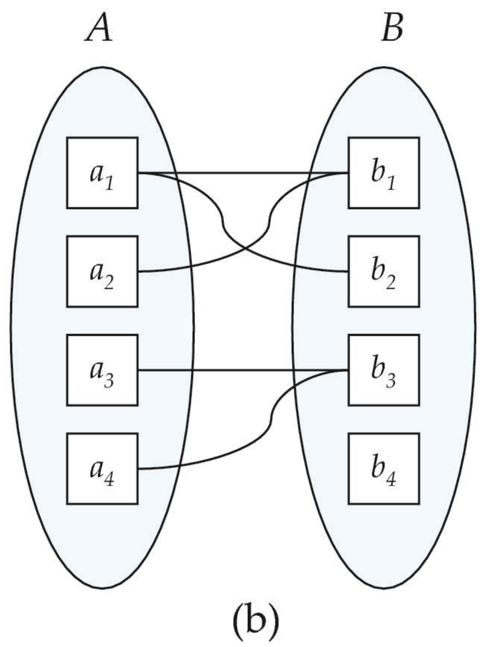
One to many

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

Mapping Cardinalities (Cont.)



Many to one



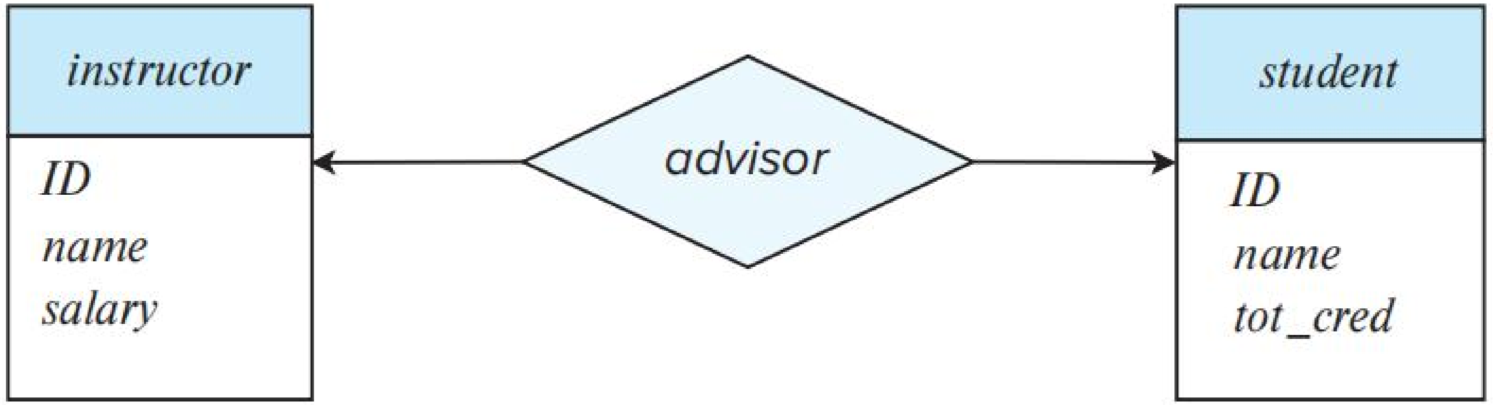
Many to many

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

Representing Cardinality Constraints in ER Diagram

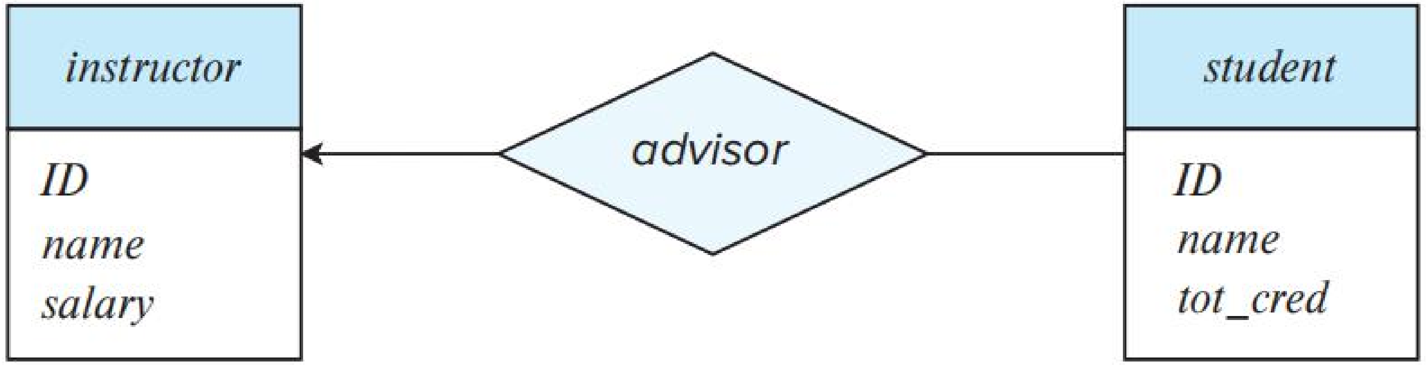
- We express cardinality constraints by drawing either a directed line (→), signifying “one,” or an undirected line (—), signifying “many,” between the relationship set and the entity set.

- 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

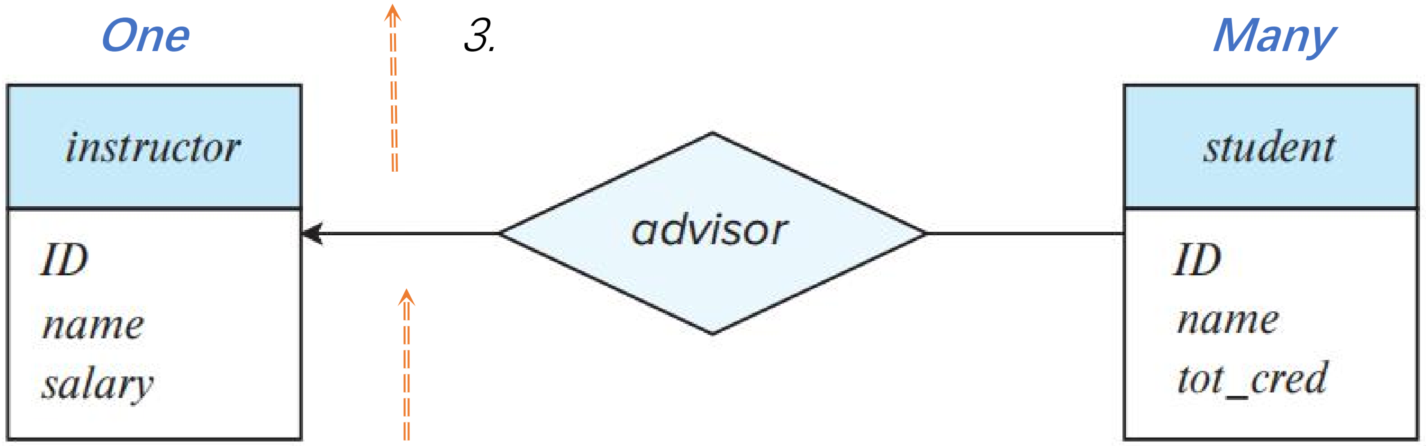
- One-to-many relationship between an *instructor* and a *student*
  - An instructor is associated with several (including 0) students via *advisor*
  - A student is associated with at most one instructor via *advisor*,



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*,

We draw a directed line from the relationship set to the ***"one"*** side of the relationship.

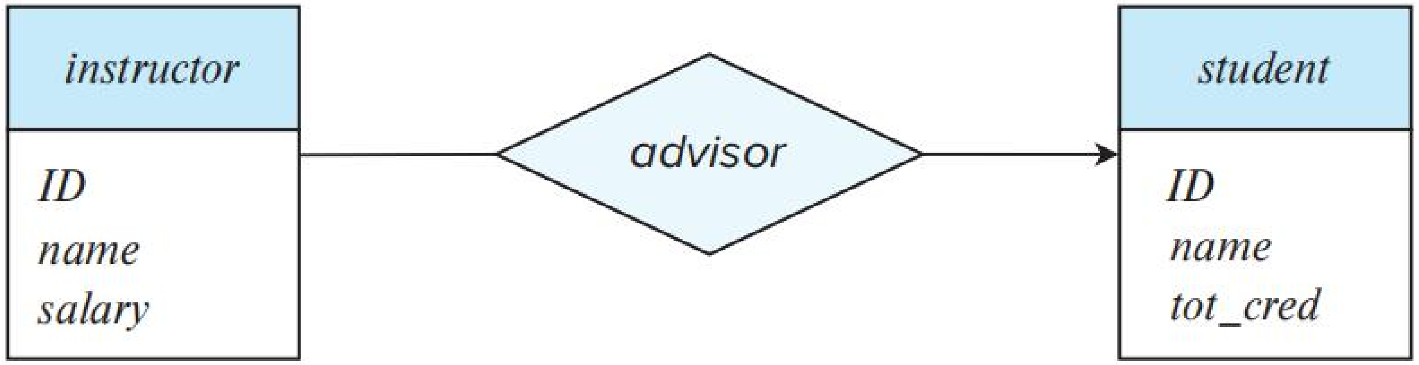


2. Looking at the line connecting instructor.

1. How many instructors can be associated with a student?

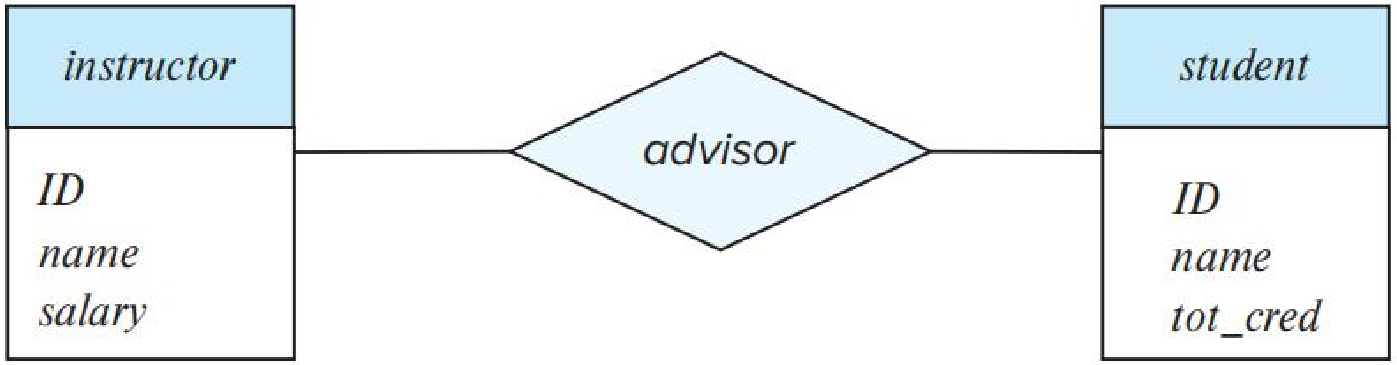
Many-to-One Relationships

- Many-to-one relationship between an *instructor* and a *student,*
  - An instructor is associated with at most one student via *advisor*,
  - A student is associated with several (including 0) instructors via *advisor*



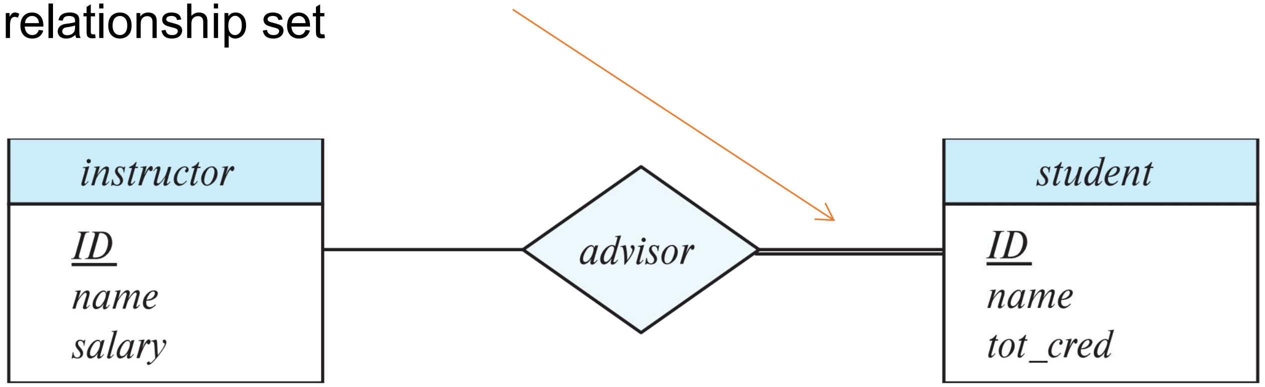
Many-to-Many Relationship

- Many-to-many relationship between an *instructor* and a *student,*
  - An instructor is associated with several (possibly 0) students via *advisor*
  - A student is associated with several (possibly 0) instructors via *advisor*



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

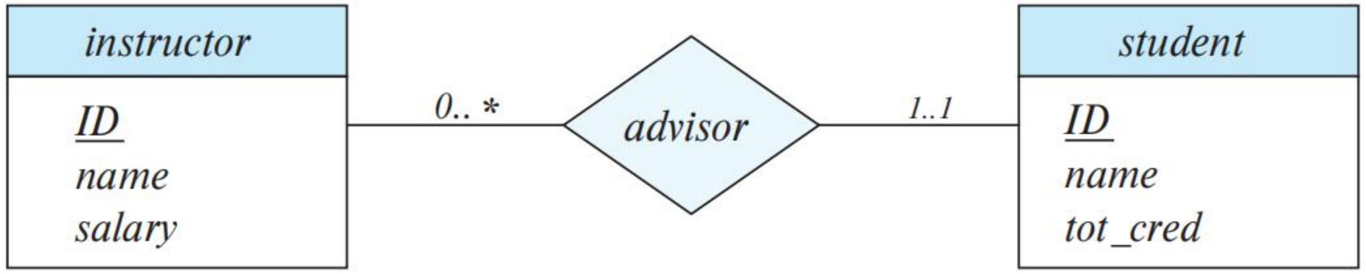


  - E.g., participation of *student* in *advisor r*elation is total
    - i.e., every *student* must have an associated *instructor*

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

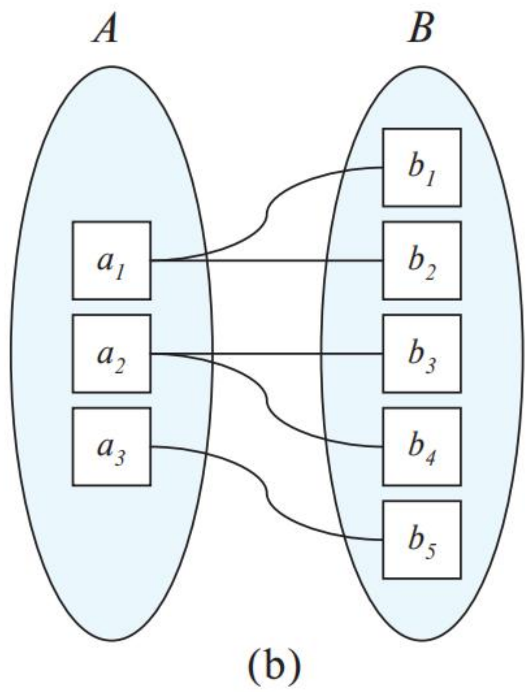
Notation for Expressing More Complex Constraints

- 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.
- Example



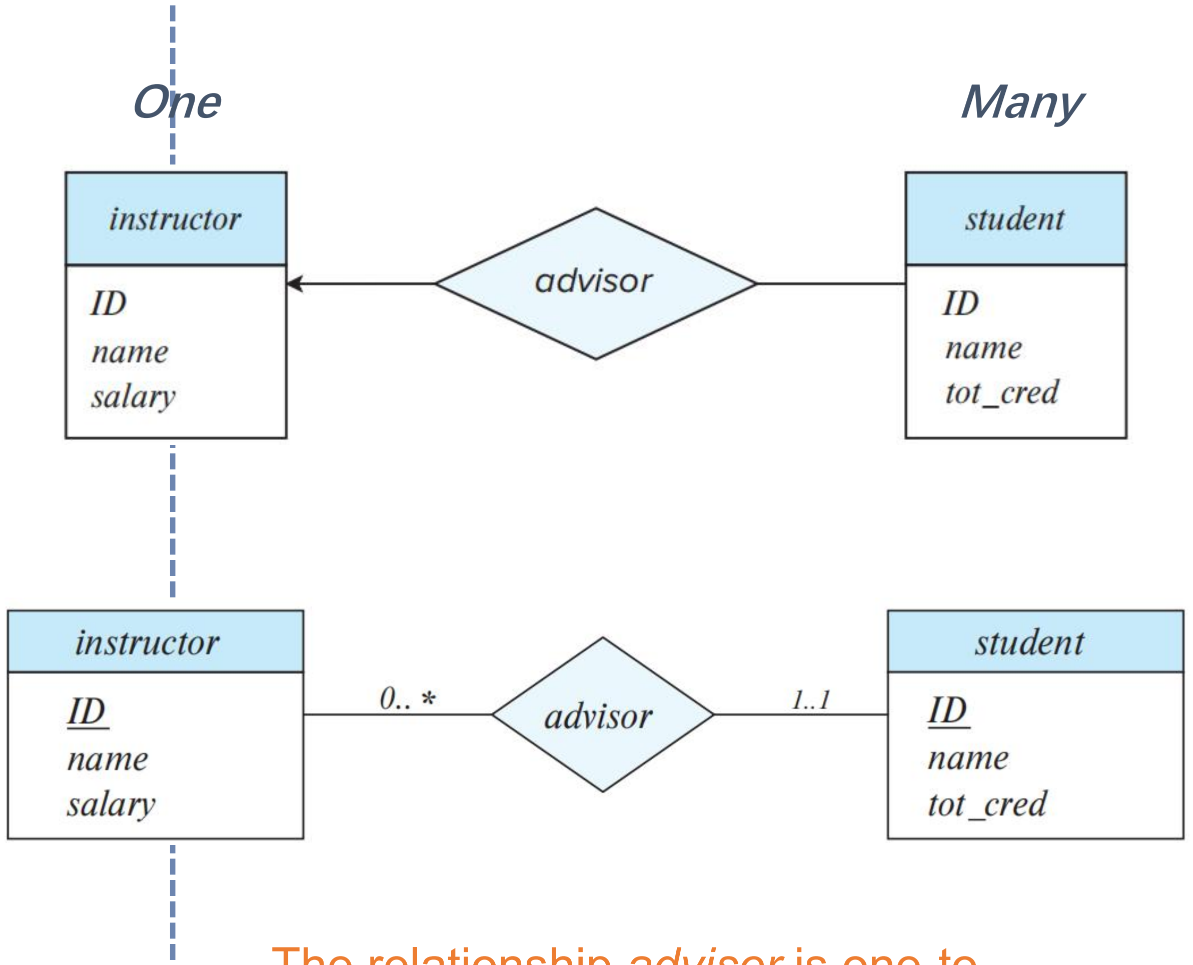
  - Instructor can advise 0 or more students. A student must have 1 advisor; cannot have multiple advisors
  - Note: The relationship *advisor* is one-to-many from *instructor* to *student*

Different Representations in ER Digram



One to many

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



The relationship *advisor* is one-to-many from *instructor* to *student*

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*

Primary Key

- Primary keys provide a way to specify how entities and relationships are distinguished.

- We will consider:
  - Entity sets
  - Relationship sets.
  - Weak entity sets

Primary key for Entity Sets

- Conceptually, individual entities are distinct.
- From database perspective, the differences among them must be expressed in terms of their attributes.
- The values of attributes 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.
  - The same concepts apply as keys (*superkeys*, *candidate keys* and *primary keys*) for relational schemas

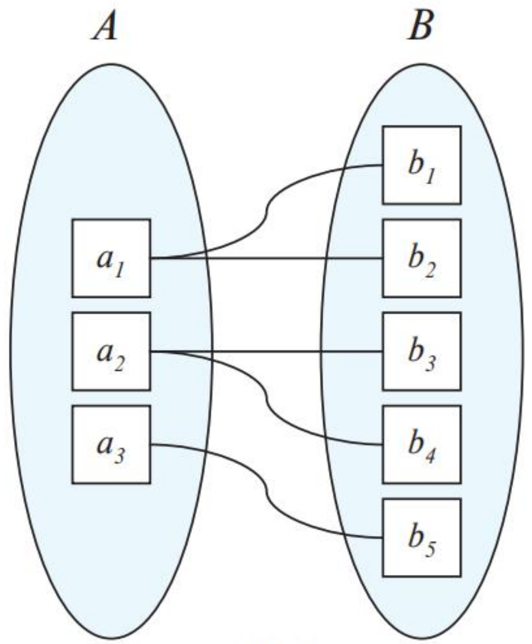
Primary Key for Relationship Sets

- 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 E_1, E_2, .. E_n
  - The primary key for *R* consists of the union of the primary keys of entity sets E_1, E_2, ..E_n

- 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.

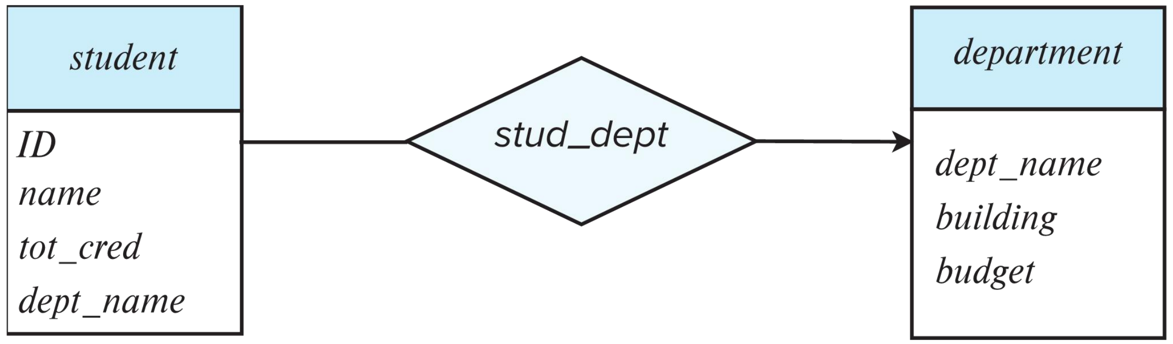


(b)

One to many

Redundant Attributes

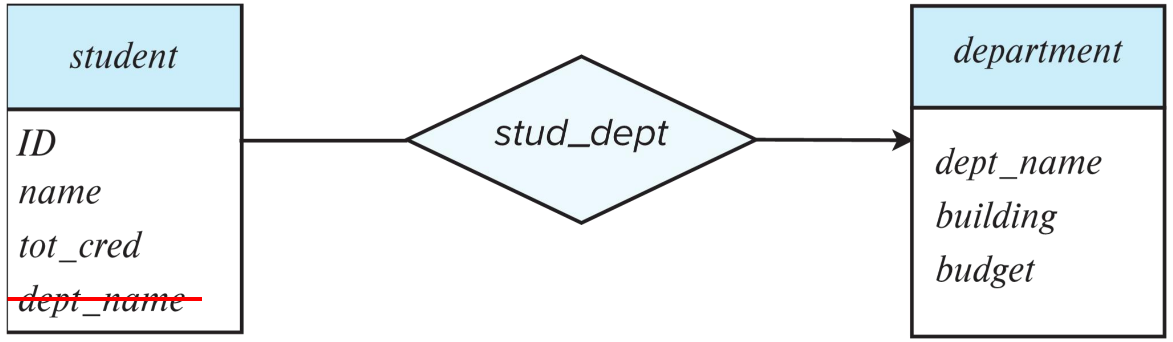
- Suppose we have entity sets:
  - *student*, with attributes: *ID*, *name, tot_cred*, *dept_name*
  - *department,* with attributes: *dept_name, building, budget*
- We model the fact that each student has an associated department using a relationship set *stud_dept*
  - The attribute *dept_name* in 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

Redundant Attributes

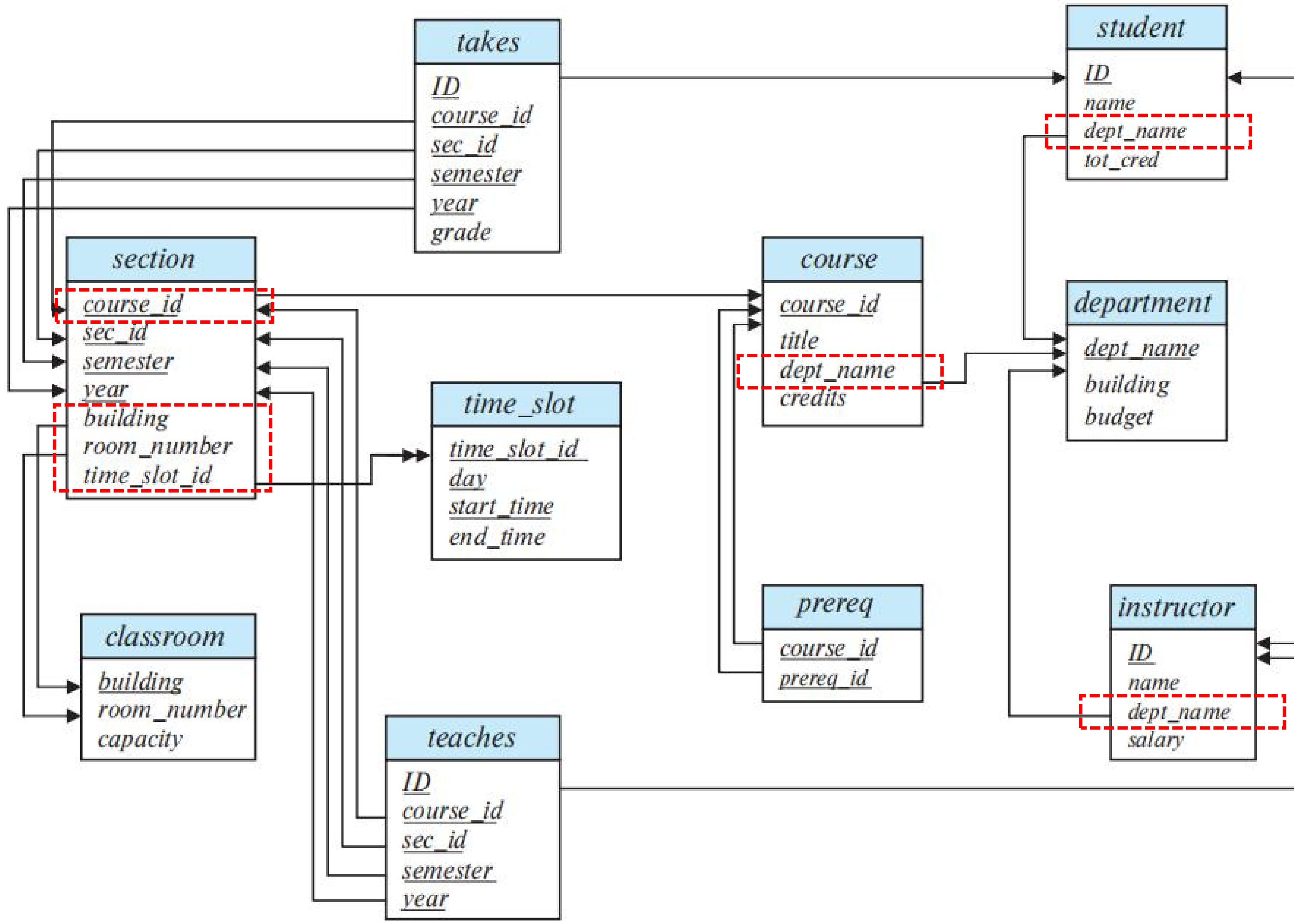
- Suppose we have entity sets:
  - *student*, with attributes: *ID*, *name, tot_cred*, *dept_name*
  - *department,* with attributes: *dept_name, building, budget*
- We model the fact that each student has an associated department using a relationship set *stud_dept*
  - The attribute *dept_name* in 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

Note: When converting ER diagram to relation schema, we add back dept_name to student relation schema only if each student has at most one associated department. If an student has more than one associated department, the relationship between student and departments is recorded in a separate relation stud_dept.

More Examples of Redundant Attributes



These attributes replicates information present in relationships between entities, but were reintroduced into relations.

Weak Entity Sets

- Consider a *section* entity, which is uniquely identified by a *course_id*, *semester, year*, and *sec_id*.

- There is a relationship set *sec_course* between entity sets *section* and *course*.
  - Clearly, section entities are related to course entities.
  - The attribute *course_id* replicates information present in the relationship sec_course and is therefore **redundant.**
  - ***Or:*** The information in *sec_course* seems **redundant**, since *section* already has an attribute *course_id*

- One option to deal with this redundancy is to get rid of the relationship s*ec_course*; however, by doing so the relationship between *section* and *course* becomes implicit in an attribute, which is not desirable.

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, i.e., {*section_id*, *year*, *semester*} is not a superkey of *section*

- 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* that is 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.

The *Section* Relation

- Each course in a university may be offered multiple times, across different semesters, or even within a semester.
- *Section* - a relation to describe each individual offering, or section, of the class.
- Instance

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2017	Painter	514	B
BIO-301	1	Summer	2018	Painter	514	A
CS-101	1	Fall	2017	Packard	101	H
CS-101	1	Spring	2018	Packard	101	F
CS-190	1	Spring	2017	Taylor	3128	E
CS-190	2	Spring	2017	Taylor	3128	A
CS-315	1	Spring	2018	Watson	120	D
CS-319	1	Spring	2018	Watson	100	B
CS-319	2	Spring	2018	Taylor	3128	C
CS-347	1	Fall	2017	Taylor	3128	A
EE-181	1	Spring	2017	Taylor	3128	C
FIN-201	1	Spring	2018	Packard	101	B
HIS-351	1	Spring	2018	Painter	514	C
MU-199	1	Spring	2018	Packard	101	D
PHY-101	1	Fall	2017	Watson	100	A

{*section_id*, *year*, *semester*} is not a superkey of *section*

Schema

section (course_id, sec_id, semester, year, building, room number, time slot id)

Weak Entity Sets (Cont.)

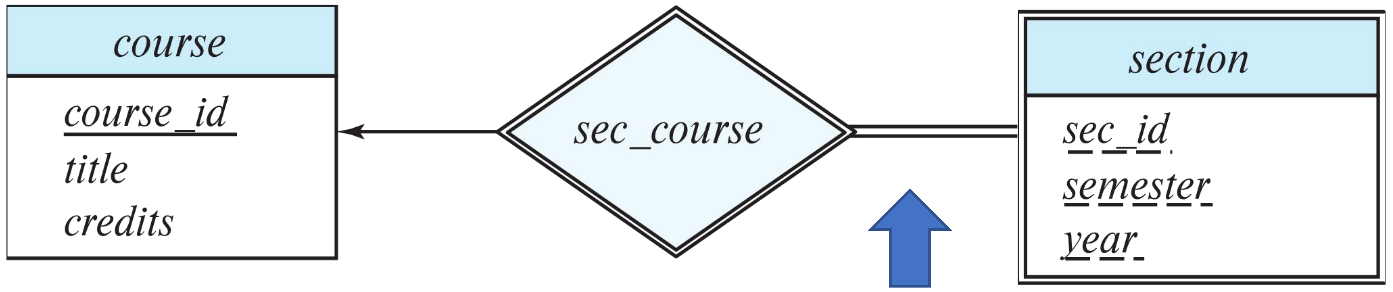
- An entity set that is not a weak entity set is termed a **strong entity set**.
- Every weak entity must be associated with an identifying entity; that is, the weak entity set is said to be **existence dependent** on the identifying entity set.
- The identifying entity set is said to **own** the weak 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.

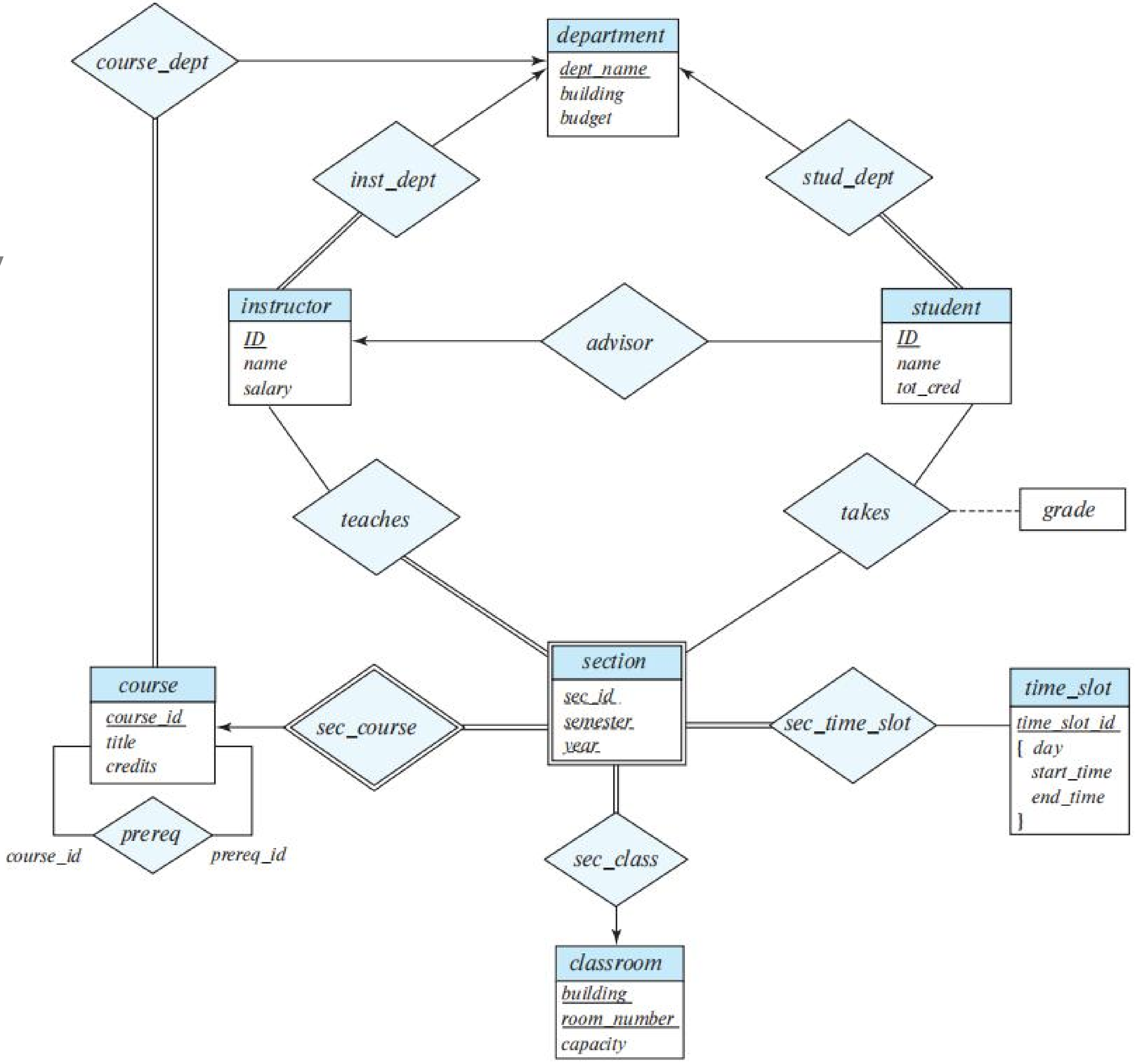
- The relationship set connecting the weak entity set to the identifying strong entity set is depicted by a double diamond.

- Primary key for *section* – (*course_id, sec_id, semester, year*)



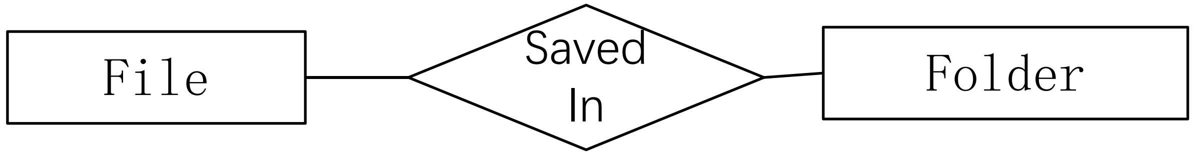
Double line → Total participation
i.e., every section must be related with some course

ER Diagram for a University Enterprise



Quick Check

- What do we change below to label the weak entity?



Steps in Database Design

- **Requirements Analysis**
  - User needs; what must database do?
- **Conceptual Design**
  - High level description (often done w/ER model)
- **Logical Design**
  - Translate ER into DBMS data model
- **Schema Refinement**
  - Consistency, normalization
- **Physical Design** - Indexes, disk layout
- **Security Design** - Who accesses what, and how

Reducing ER Diagram to Relational Schemas

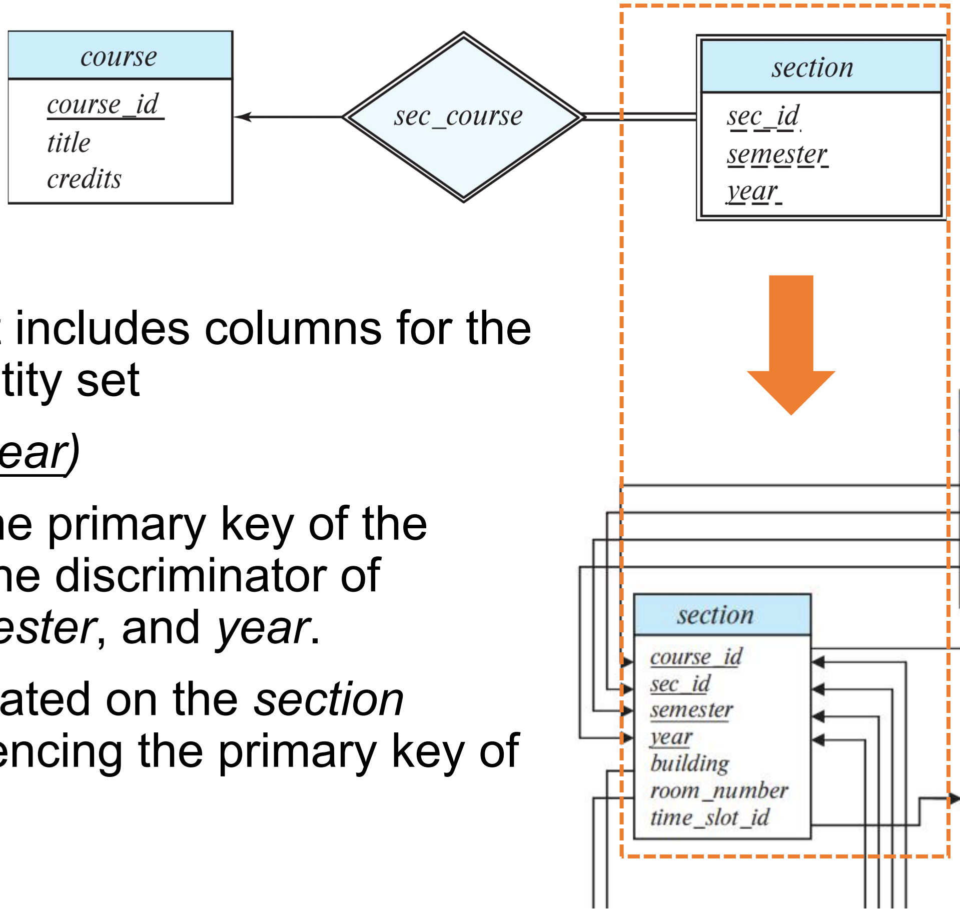
Reducing ER Diagram to Relational Schemas

- Entity-Relationship Model & Relational Model
  - Abstract, logical representations of real-world enterprises
  - Employ similar design principles

- Entity-Relationship Model → Relational Model
  - 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, i.e.,
    - Create one table for each entity set with the same name
    - Create one table for each relationship set with the same name

Representing Entity Sets

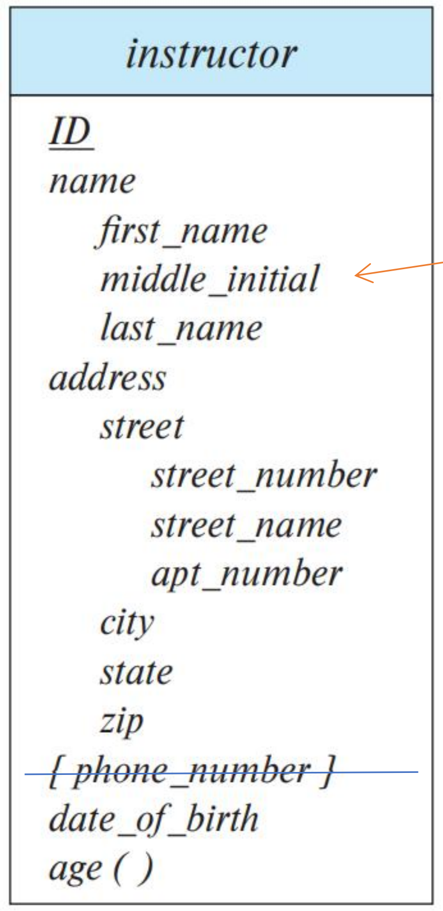
- A **strong** entity set reduces to a relation with the same attributes
  - Entity → Tuple
  - Attribute → Attribute
  - E.g., *student (<u>ID</u>, name, tot cred)*



Converting entity set *section* to relation schema *section*

- A **weak** entity set becomes a table that includes columns for the primary key of the identifying strong entity set
  - *section (<u>course_id</u>, <u>sec_id</u>, <u>sem</u>, <u>year</u>)*
    - The primary key consists of the primary key of the entity set *course*, along with the discriminator of *section*, which is *sec_id, semester*, and *year*.
    - A foreign-key constraint is created on the *section* schema, with *course_id* referencing the primary key of the *course* schema

Representation of Entity Sets with Complex Attributes



- Composite attributes are flattened out by creating a separate attribute for each component attribute
  - E.g., 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(<u>ID</u>, first_name, middle_initial, last_name, street_number, street_name, apt_number, city, state, zip_code, date_of_birth)*

Representation of Entity Sets with Multivalued Attributes

- A multivalued attribute *M* of an entity *E* is represented by a separate relation schema *EM*

- Schema *EM* has attributes corresponding to the primary key of *E* and an attribute corresponding to multivalued attribute *M*
  - The primary key of EM consists of all attributes of the relation schema
  - Example: Multivalued attribute *phone_number* of *instructor* is represented by a schema:
    inst_phone= (*ID*, *phone_number*)
  - A foreign-key constraint is created on *EM* referencing the primary key of *E,*
  - Example: *inst_phone.ID* → *instructor.ID*

- Each value of the multivalued attribute maps to a separate tuple of the relation on schema *EM*
  - 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)

Representing Relationship Sets

- Let *R* be a relationship set, let $a_1, a_2, \ldots, a_m$ 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 $b_1, b_2, \ldots, b_n$. We represent this relationship set by a relation schema called *R* with one attribute for each member of the set:

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

- Example: schema for relationship set *takes*

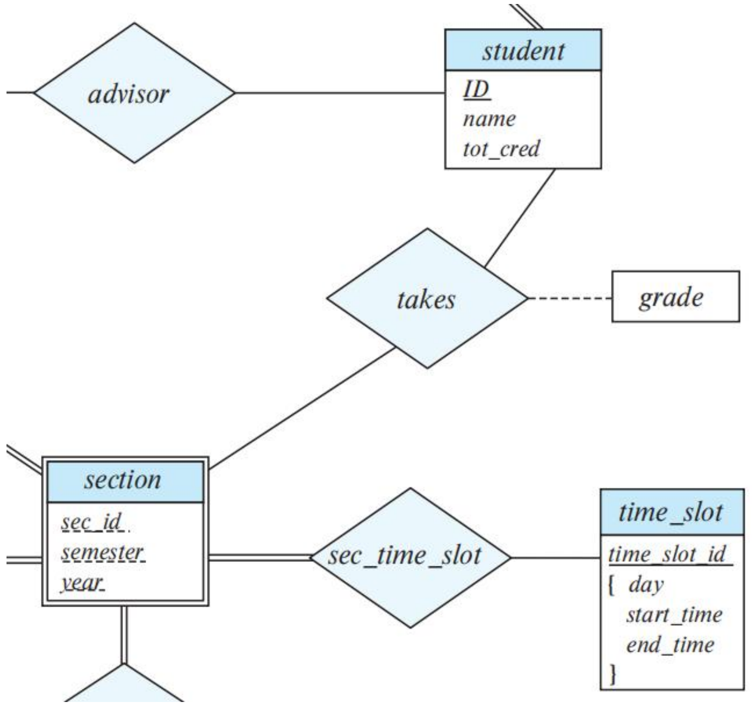
    takes (ID, course id, sec id, semester, year, grade)

  given:

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

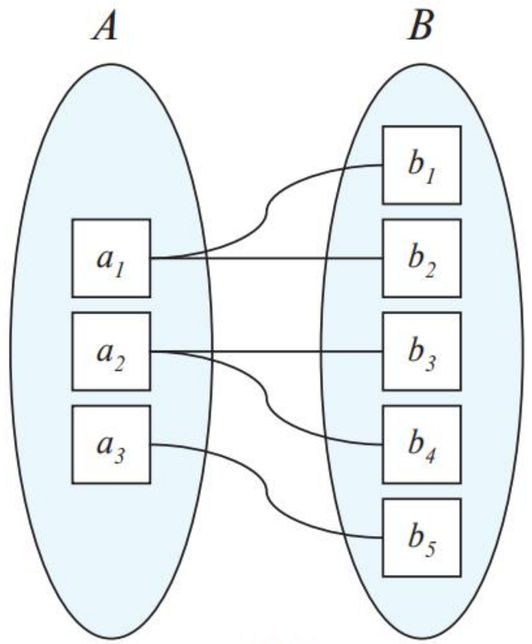
    student(<u>ID</u>, name, tot cred)

- What is the the primary key for this relation R?



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.



(b)

One to many

Representing Relationship Sets

- Let *R* be a relationship set, let $a_1, a_2, \ldots, a_m$ 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 $b_1, b_2, \ldots, b_n$. We represent this relationship set by a relation schema called *R* with one attribute for each member of the set:

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

- Example: schema for relationship set *takes*

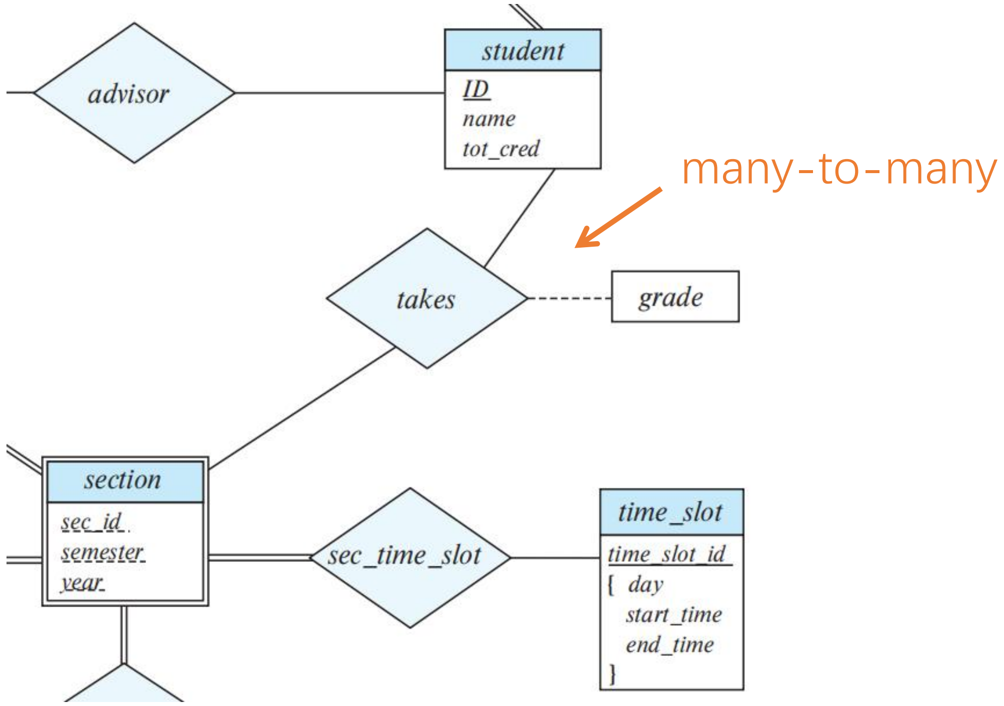
  takes (<u>ID</u>, <u>course_id</u>, <u>sec_id</u>, <u>semester</u>, <u>year</u>, grade)

  given:

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

  student(<u>ID</u>, name, tot cred)

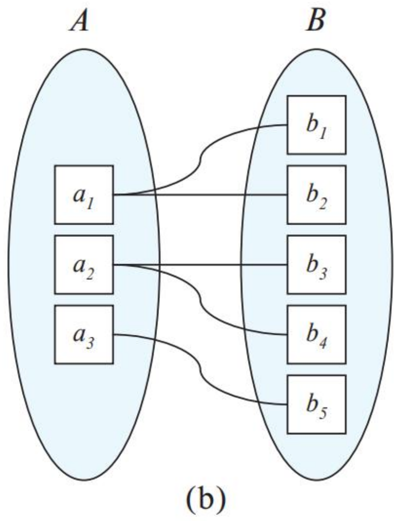
- What is the the primary key for this relation *R*?



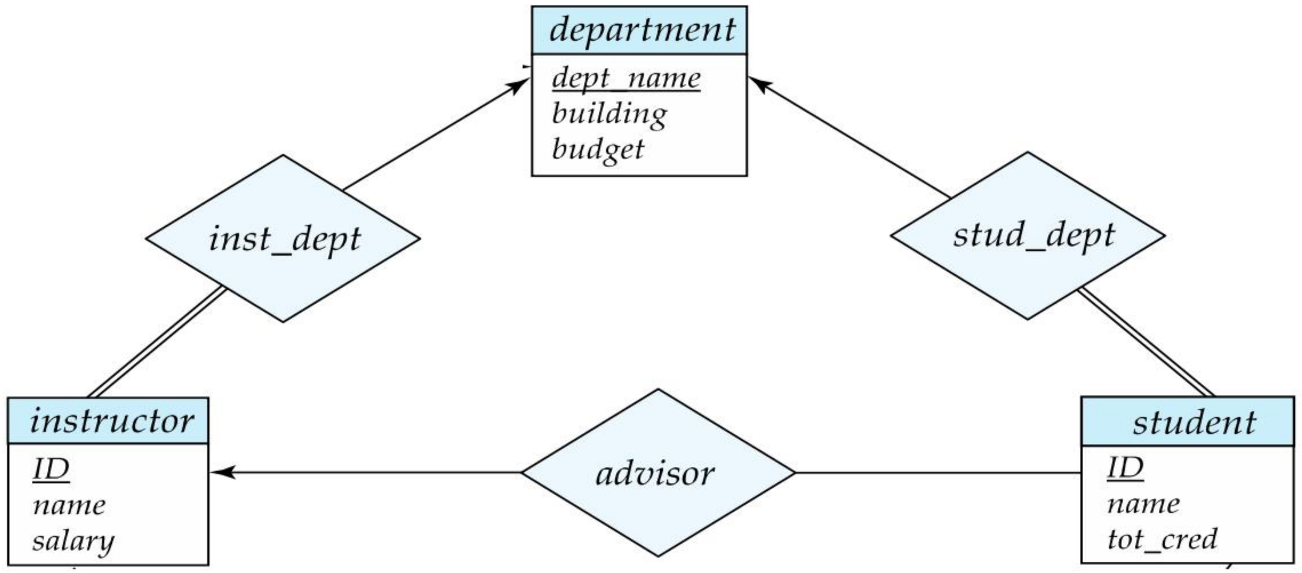
Redundancy of Schemas

- 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*



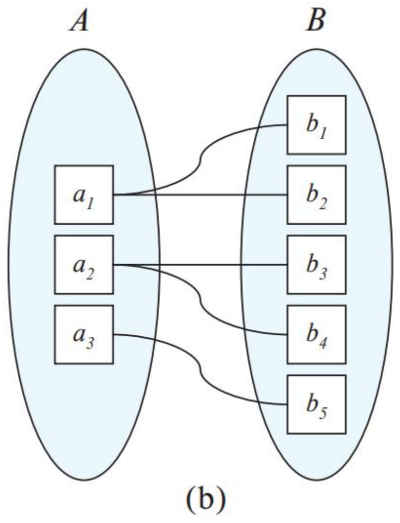
A = *department*
B = *instructor*



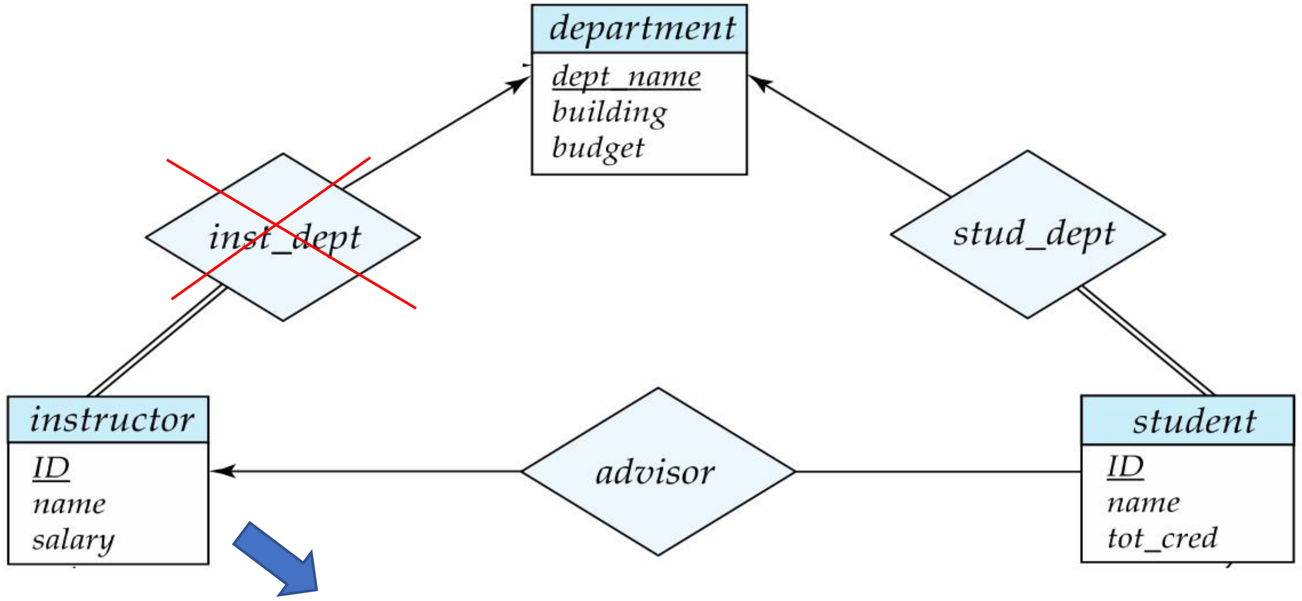
Redundancy of Schemas

- 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*



A = *department*
B = *instructor*

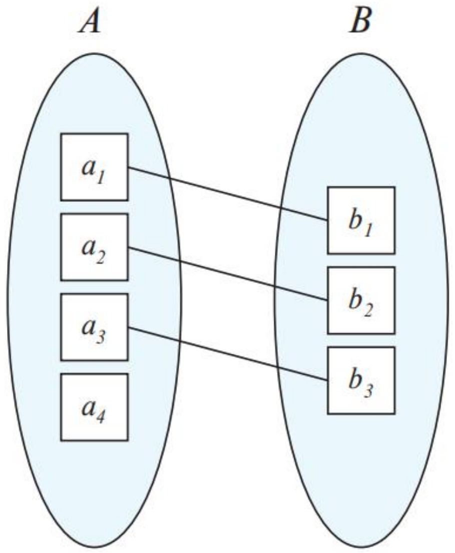


instructor(<u>ID</u>, name, ***dept_name****, salary)*

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



Example

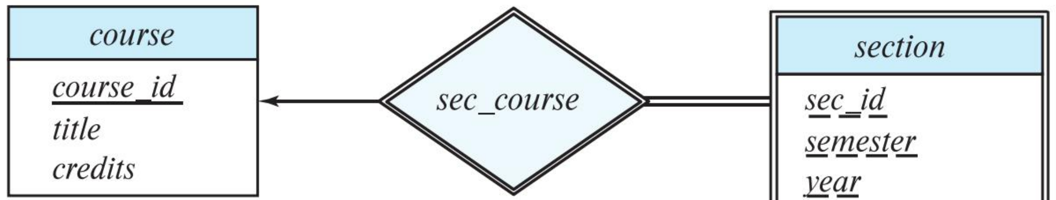
- Entity a_4 does not participate in the binary relationship.
- If we add the PK of *B* to relation *A* to reprensent this relationship, the tuple corresponding to a_4 would have null value for this column.
- In this case, choose *B* as the "many" side to avoid null values

Redundancy of Schemas (Cont.)

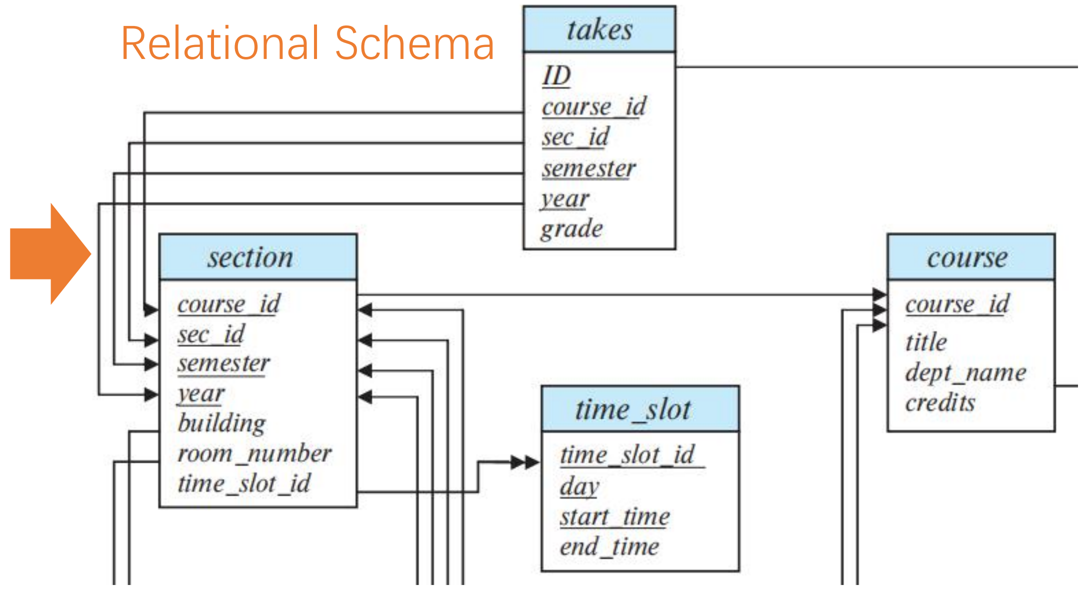
- The schema corresponding to a relationship set linking a weak entity set to its identifying strong entity set is redundant.

- Example: The *section* schema already contains the attributes that would appear in the *sec_course* schema

ER Diagram



Relational Schema

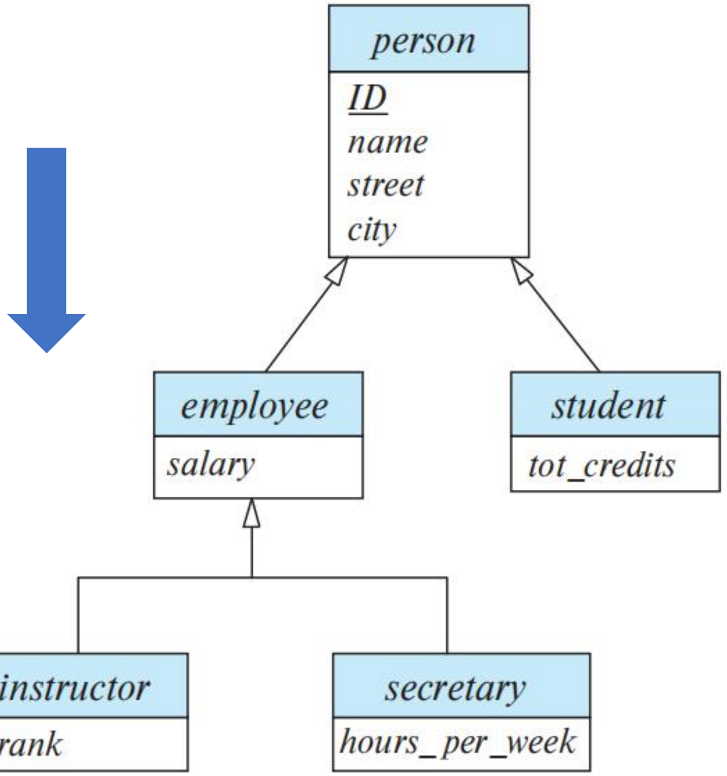


No table is created in the schema for relationship *sec_course*

Extended ER Features

Specialization

- ***Top-down design process*** - 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 (e.g., *salary* of *employee*)
- **ER Diagram** - depicted by a *hollow arrow-head* pointing from the specialized entity to the other, referred as ISA relationship (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.

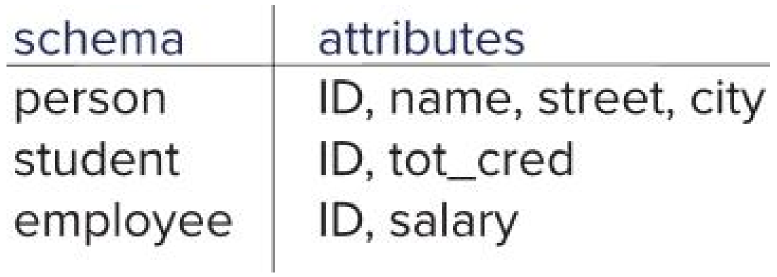


Constraint on specialization

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

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

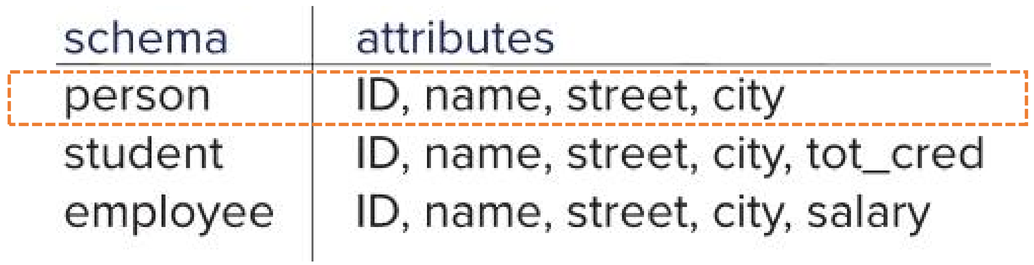


schema	attributes
person	ID, name, street, city
student	ID, tot_cred
employee	ID, salary

  - *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

Representing Specialization as Schemas (Cont.)

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

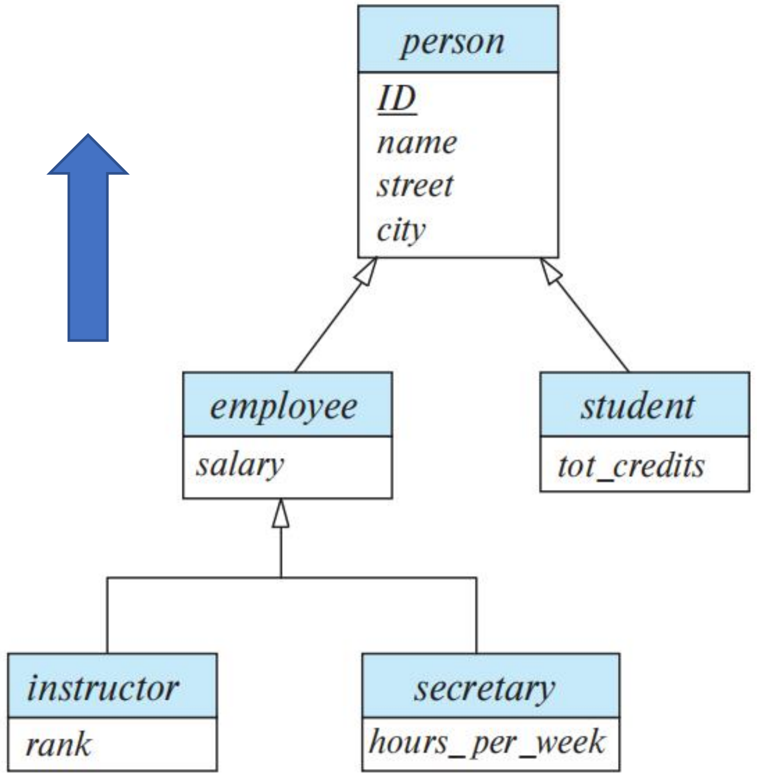


schema	attributes
person	ID, name, street, city
student	ID, name, street, city, tot_cred
employee	ID, name, street, city, salary

  - *Drawback*: *name, street* and *city* may be stored redundantly for people who are both students and employees
  - If the generalization is disjoint and complete, may remove the *person* relation
    - No FK constraints referencing *person.ID*

Generalization

- **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 ER diagram in the same way.
- In terms of the ER diagram, the terms specialization and generalization are used interchangeably.
- **Specialization** - focus on differences among entities within one set
- **Generalization** - focus on the similarities among lower-level entity sets
  - Hide difference
  - Economy of representation i.e., shared attributes are not repeated



Completeness Constraint

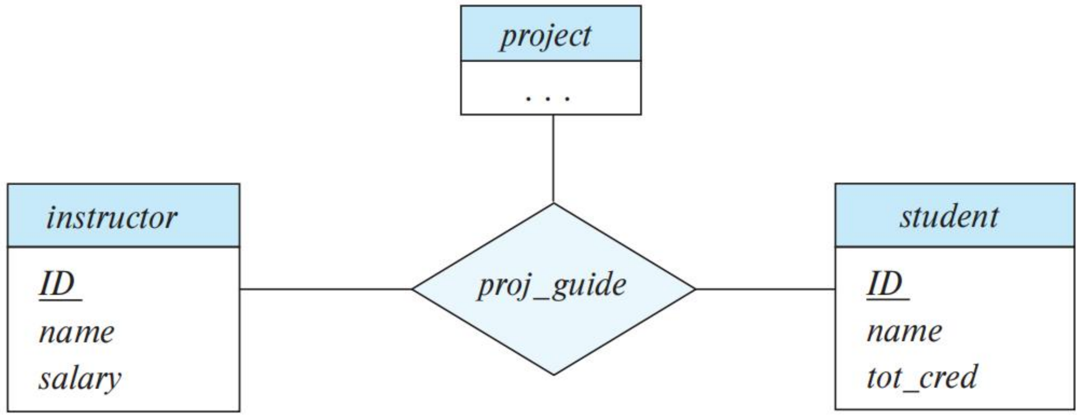
- **Completeness constraint** -- specifies whether or not an entity in the higher-level entity set must belong to at least one of the lower-level entity sets within a generalization/specialization .

  - **Total**: an entity must belong to one of the lower-level entity sets
    - ER Diagram: We can specify total specialization by adding the keyword “total” and drawing a dashed line from the keyword to the corresponding hollow arrowhead to which it applies (for a disjoint specialization), or to the set of hollow arrowheads to which it applies (for an overlapping specialization).

  - **Partial**: an entity need not belong to one of the lower-level entity sets
    - Default

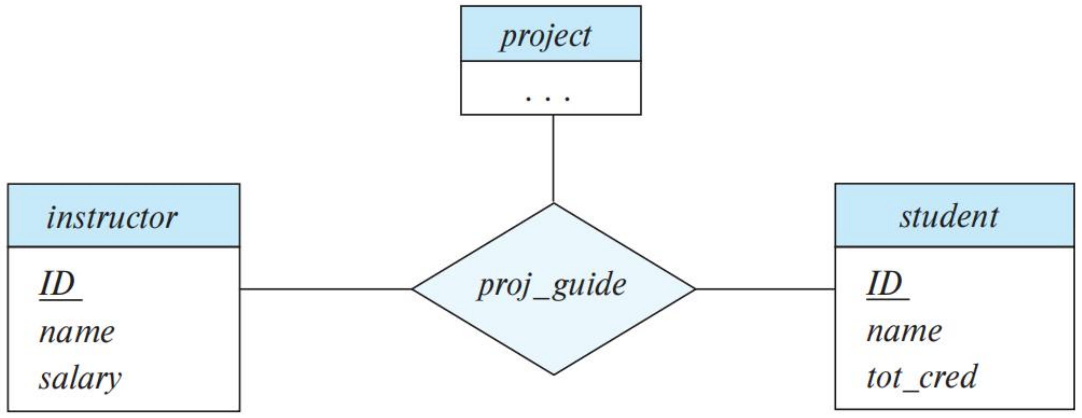
Aggregation

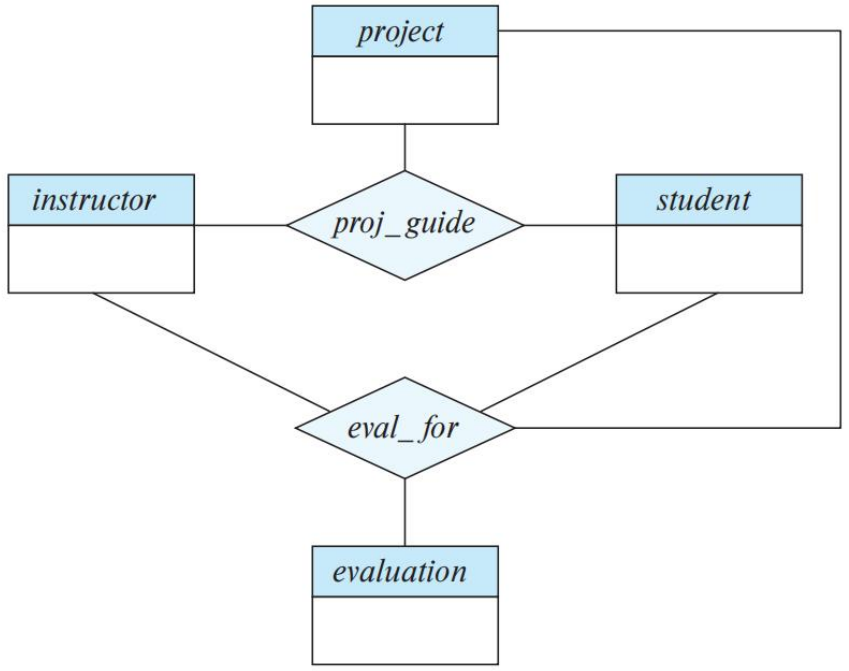
- One *limitation* of the E-R model is that it cannot express relationships among relationships.
- Consider the ternary relationship *proj_guide*, which we saw earlier
- Suppose we want to record evaluation report of a student by a guide on a project, once a month
  - Model the evaluation report as an entity *evaluation*



Aggregation

- One *limitation* of the E-R model is that it cannot express relationships among relationships.
- Consider the ternary relationship *proj_guide*, which we saw earlier
- Suppose we want to record evaluation report of a student by a guide on a project, once a month
  - Model the evaluation report as an entity *evaluation*
  - Create a quaternary (4-way) relationship set *eval_for*

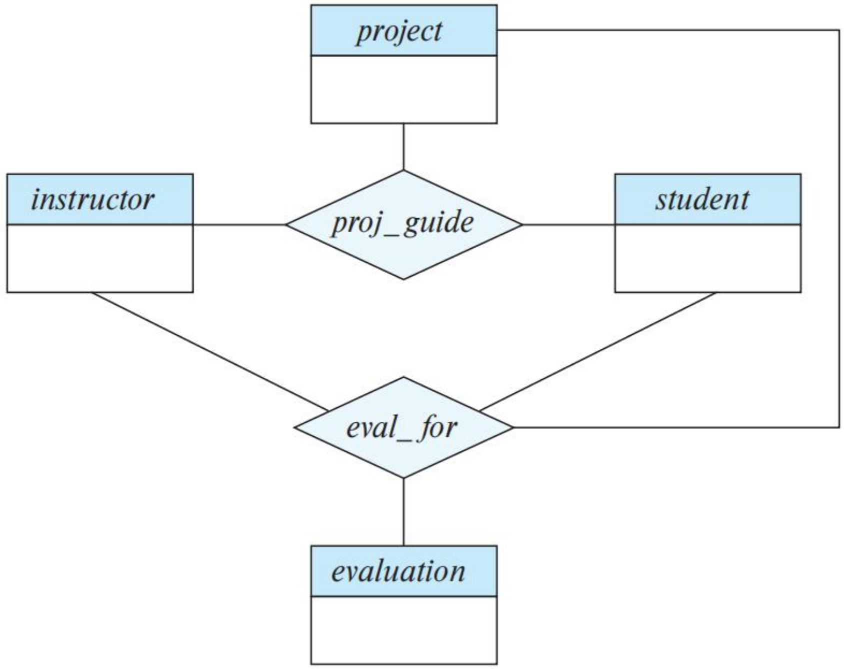




Aggregation (Cont.)

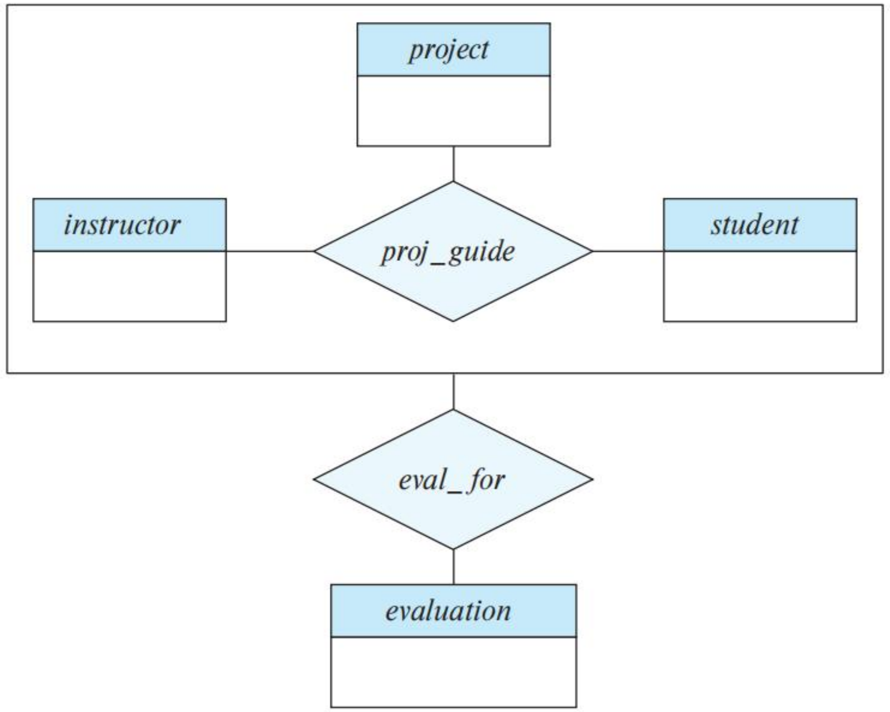
- Relationship sets *eval_for* and *proj_guide* represent overlapping information
  - Every *eval_for* relationship corresponds to a *proj_guide* relationship
  - However, some *proj_guide* relationships may not correspond to any *eval_for* relationships
    - So we can't discard the *proj_guide* relationship

- Eliminate this redundancy via ***aggregation***
  - Treat relationship as an abstract entity
  - Allows relationships between relationships
  - Abstraction of relationship into new entity



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 report

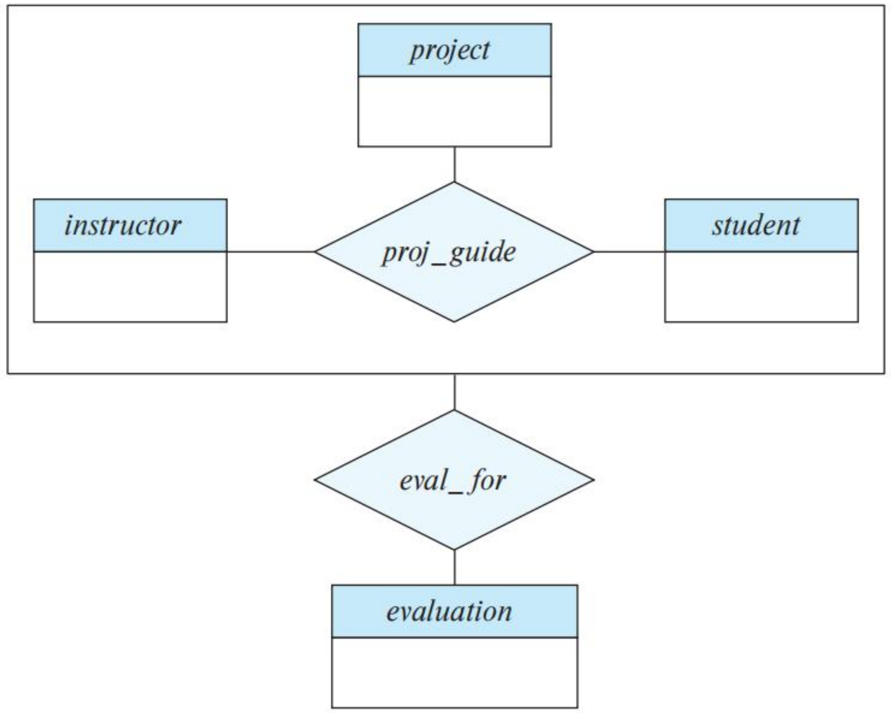


Reduction to Relational Schemas

- To represent aggregation, create a schema containing
  - Primary key of the aggregated relationship set
  - The primary key of the associated entity set
  - Any descriptive attributes

- In our example:
  - The schema *eval_for* is:

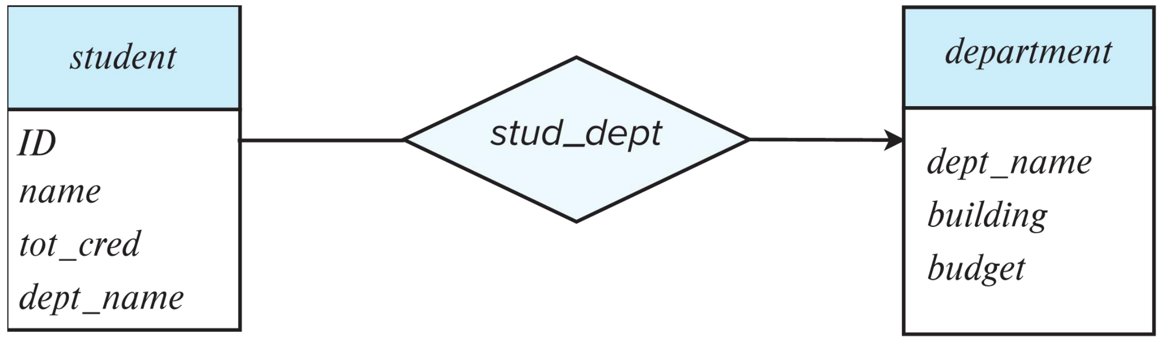
    eval_for (*s_ID, project_id, i_ID, evaluation_id*)



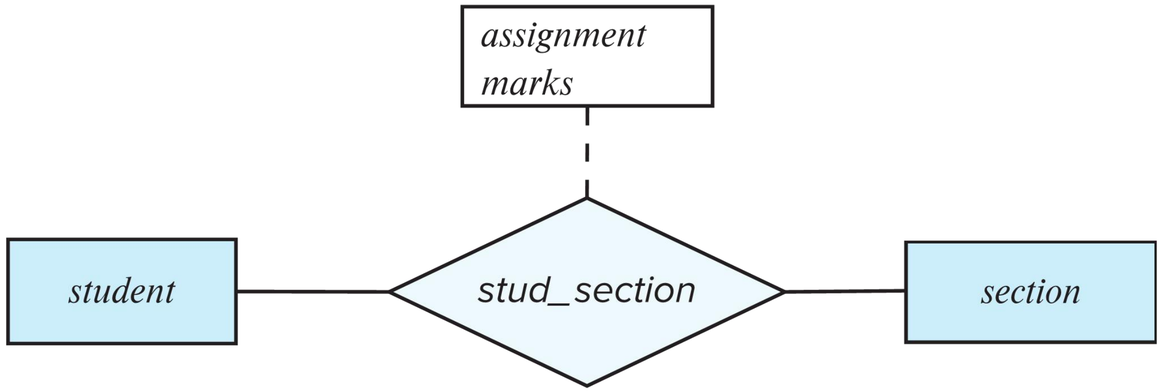
Design Issues

Common Mistakes in ER Diagrams

- Example of erroneous ER diagrams



(a) Incorrect use of attribute

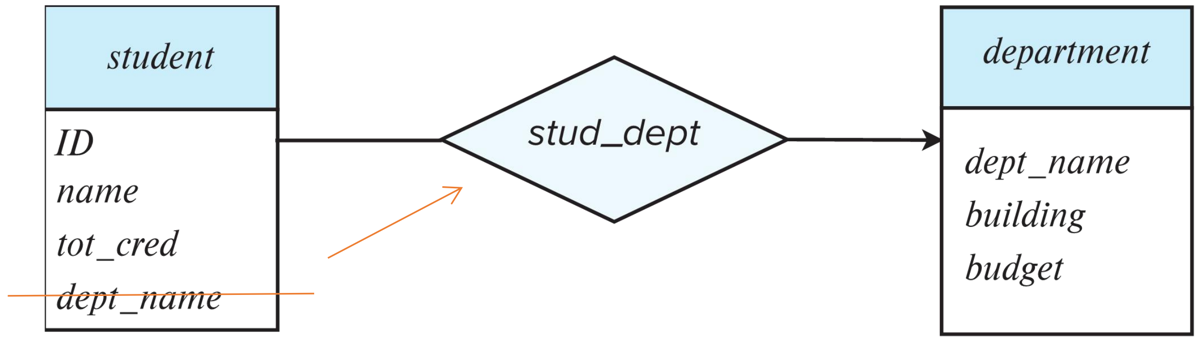


(b) Erroneous use of relationship attributes

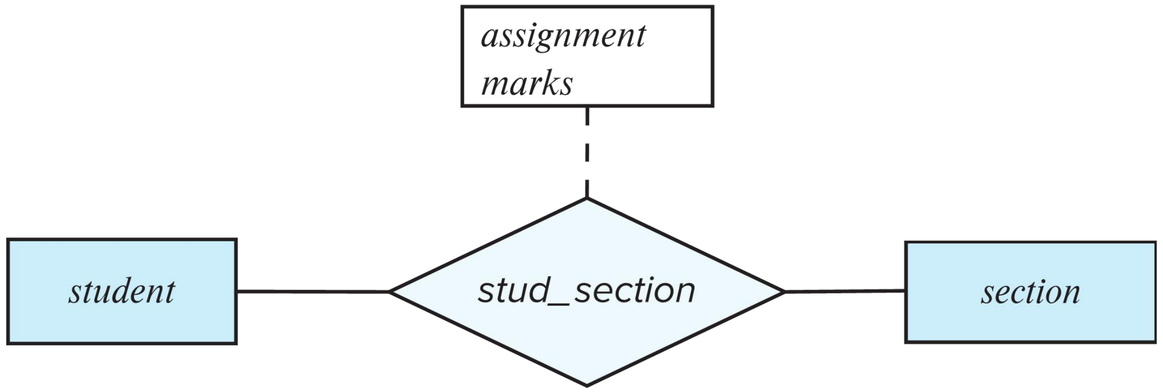
Common Mistakes in ER Diagrams

- Example of erroneous ER diagrams

Duplicate information



(a) Incorrect use of attribute



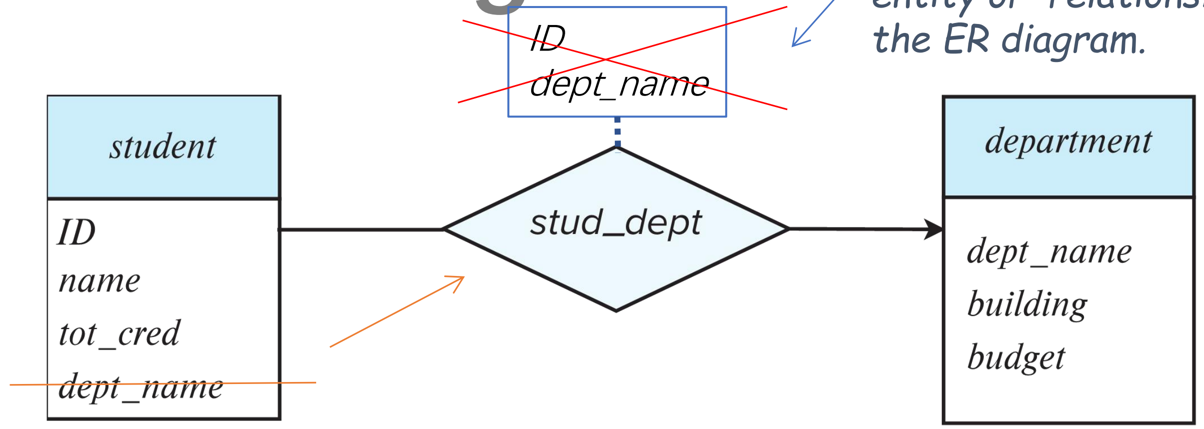
(b) Erroneous use of relationship attributes

Common Mistakes in ER Diagrams

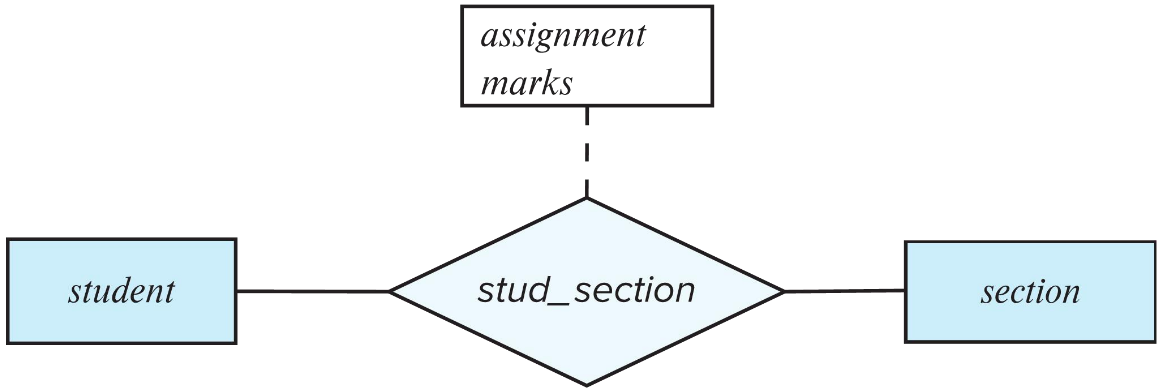
- Example of erroneous ER diagrams

Duplicate information

These attributes may appear in the **relation schema** created from the relationship set, but they should not appear in the entity or relationship set in the ER diagram.

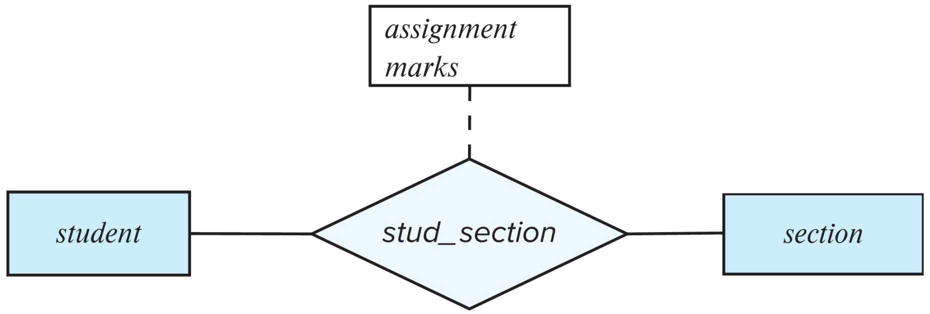


(a) Incorrect use of attribute



(b) Erroneous use of relationship attributes

Common Mistakes in ER Diagrams (Cont.)



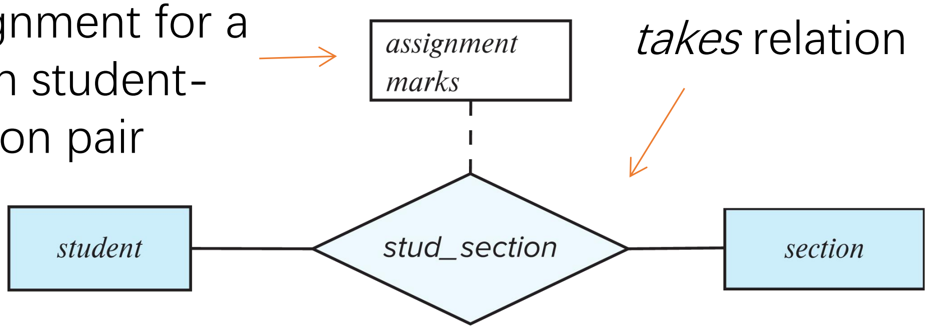
(b) Erroneous use of relationship attributes

E.g., We need to represent the marks a student gets in different assignments of a course offering (section).

Common Mistakes in ER Diagrams (Cont.)

we can only represent a single assignment for a given student-section pair

takes relation



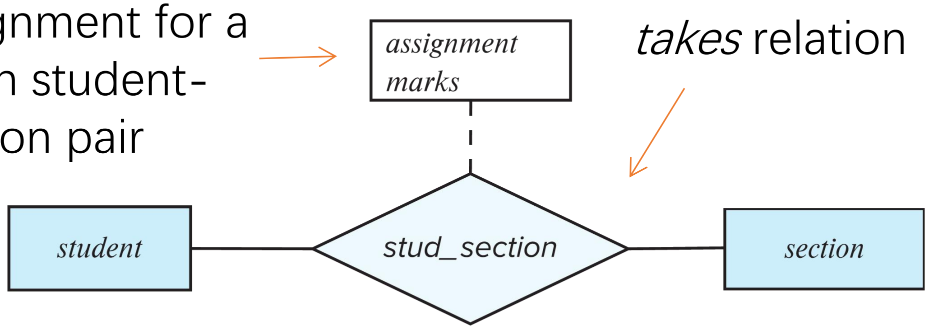
(b) Erroneous use of relationship attributes

E.g., We need to represent the marks a student gets in different assignments of a course offering (section).

Common Mistakes in ER Diagrams (Cont.)

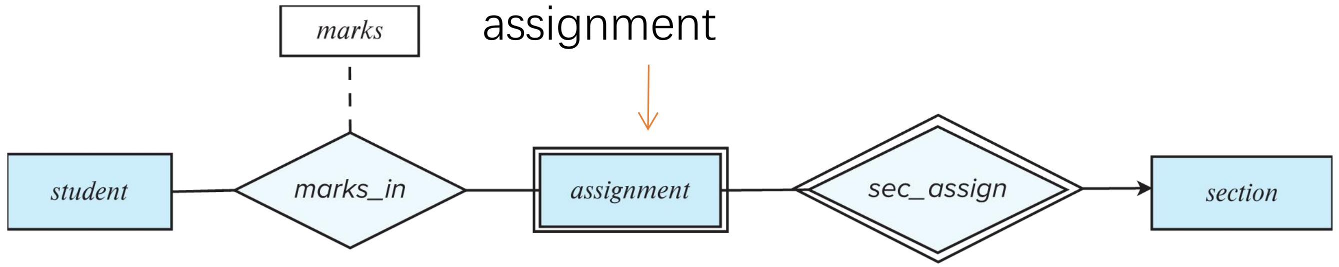
we can only represent a single assignment for a given student-section pair

takes relation



(b) Erroneous use of relationship attributes

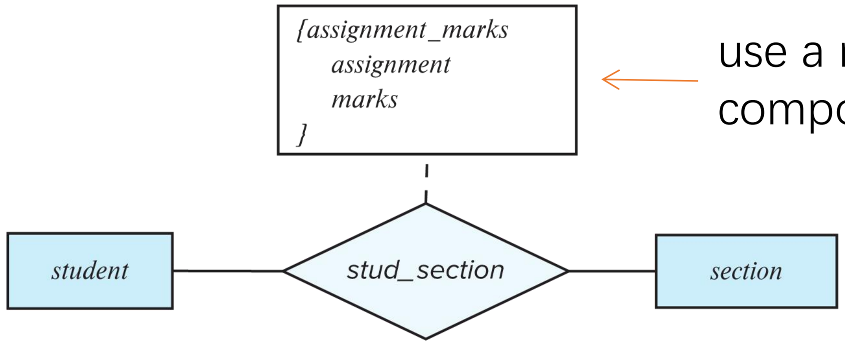
Introduce weak entity set assignment



(c) Correct alternative to erroneous E-R diagram (b)

E.g., We need to represent the marks a student gets in different assignments of a course offering (section).

use a multivalued composite attribute

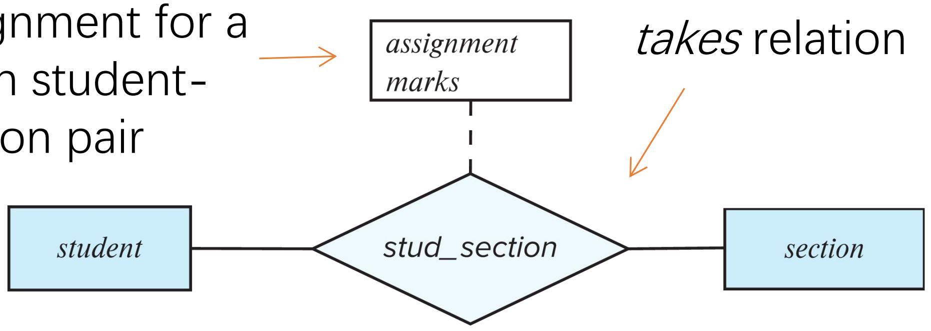


(d) Correct alternative to erroneous E-R diagram (b)

Common Mistakes in ER Diagrams (Cont.)

we can only represent a single assignment for a given student-section pair

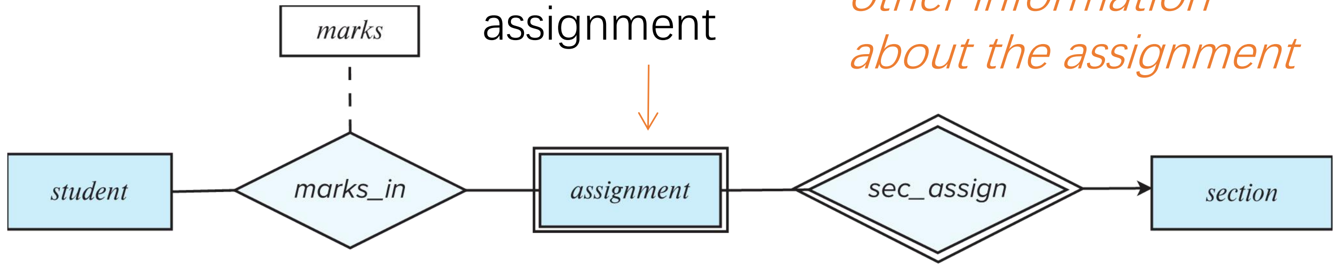
takes relation



(b) Erroneous use of relationship attributes

Introduce weak entity set assignment

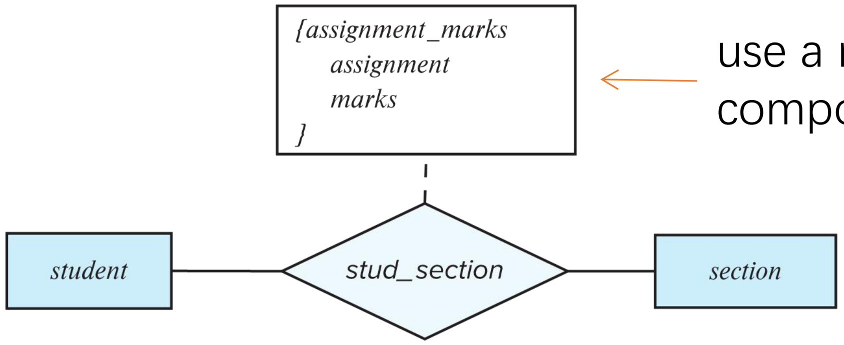
(c) is better than (d) as it allows recording other information about the assignment



(c) Correct alternative to erroneous E-R diagram (b)

E.g., We need to represent the marks a student gets in different assignments of a course offering (section).

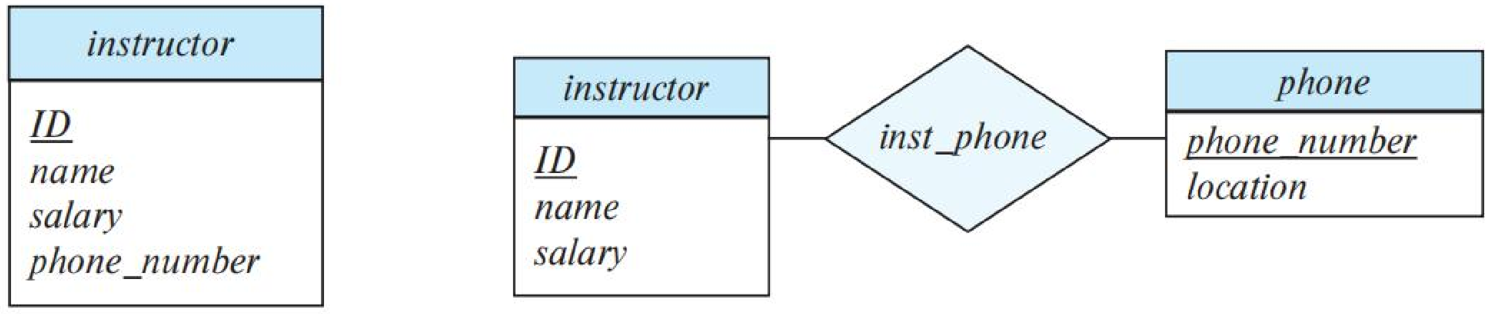
use a multivalued composite attribute



(d) Correct alternative to erroneous E-R diagram (b)

Entities vs. Attributes

- Use of entity sets vs. attributes

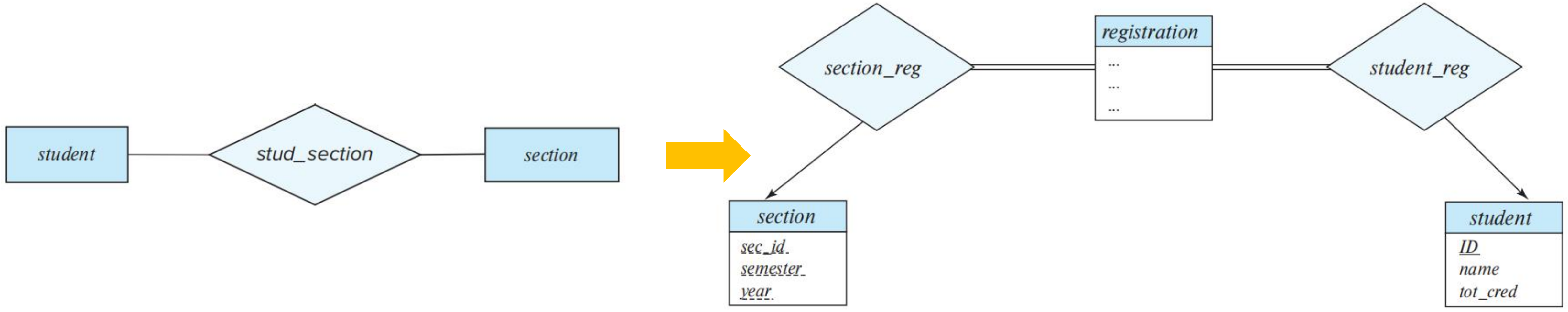


- Use of *phone* as an entity allows extra information about phone numbers (plus multiple phone numbers)
  - It would not be appropriate to treat the attribute name as an entity

- *What constitutes an attribute, and what constitutes an entity set?*

Entities vs. Relationship sets

- **Use of entity sets vs. relationship sets**



- The use of *takes* (*stud_section*) is more compact and probably preferable
- If the registrar's office associates other information with a course-registration record, it might be best to make it an entity in its own right

Possible guideline is to designate a relationship set to describe an action that occurs between entities

Binary Vs. Non-Binary Relationships

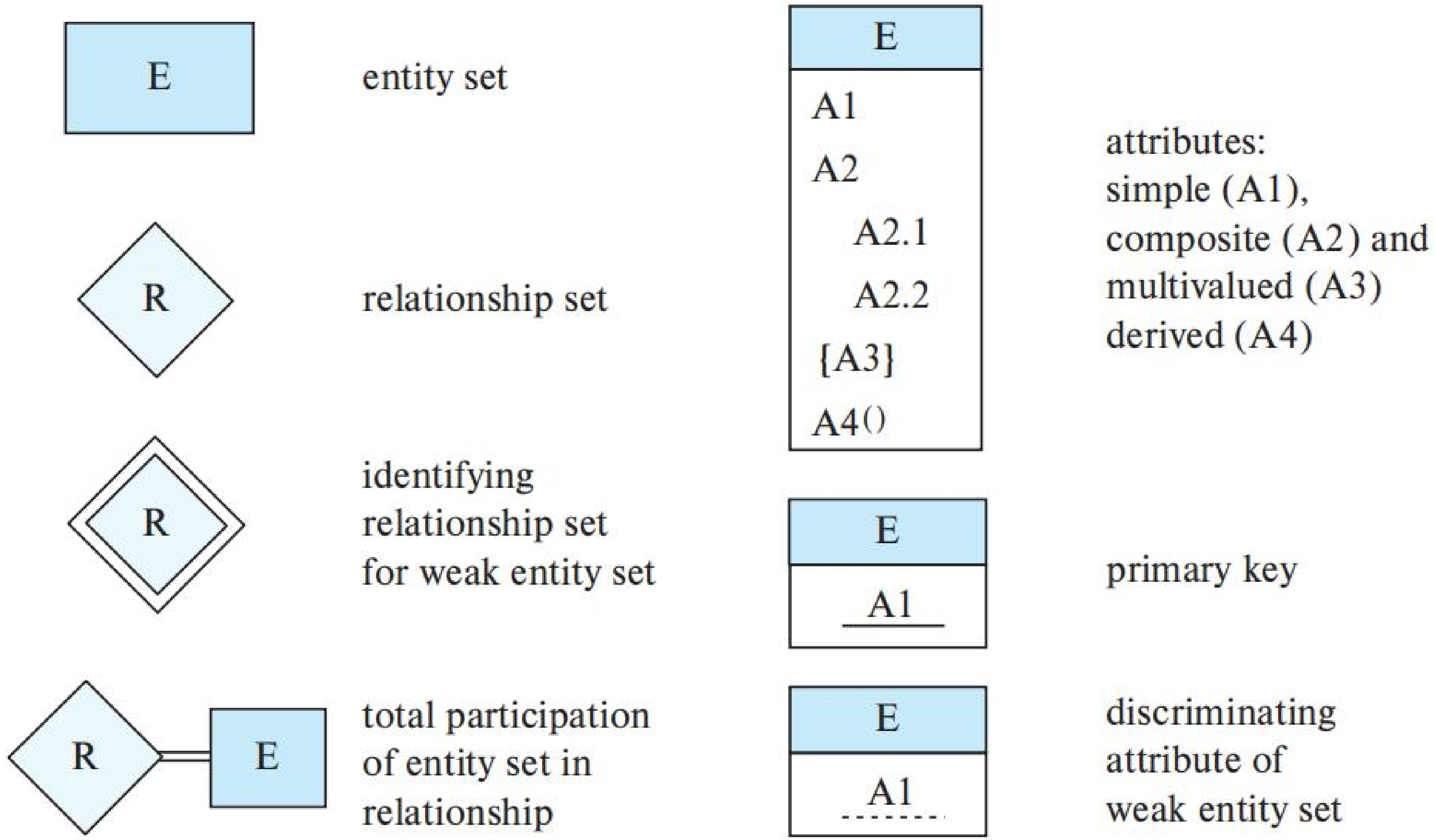
- Although it is possible to replace any non-binary (*n*-ary, for *n* > 2) relationship set by a number of distinct binary relationship sets, a *n*-ary relationship set shows more clearly that several entities participate in a single relationship.

- Some relationships that appear to be non-binary may be better represented using binary relationships
    - For example, a ternary relationship *parents*, relating a child to his/her father and mother, is best replaced by two binary relationships, *father* and *mother*
        - Using two binary relationships allows partial information (e.g., only mother being known)
    - But there are some relationships that are naturally non-binary
        - Example: *proj_guide*

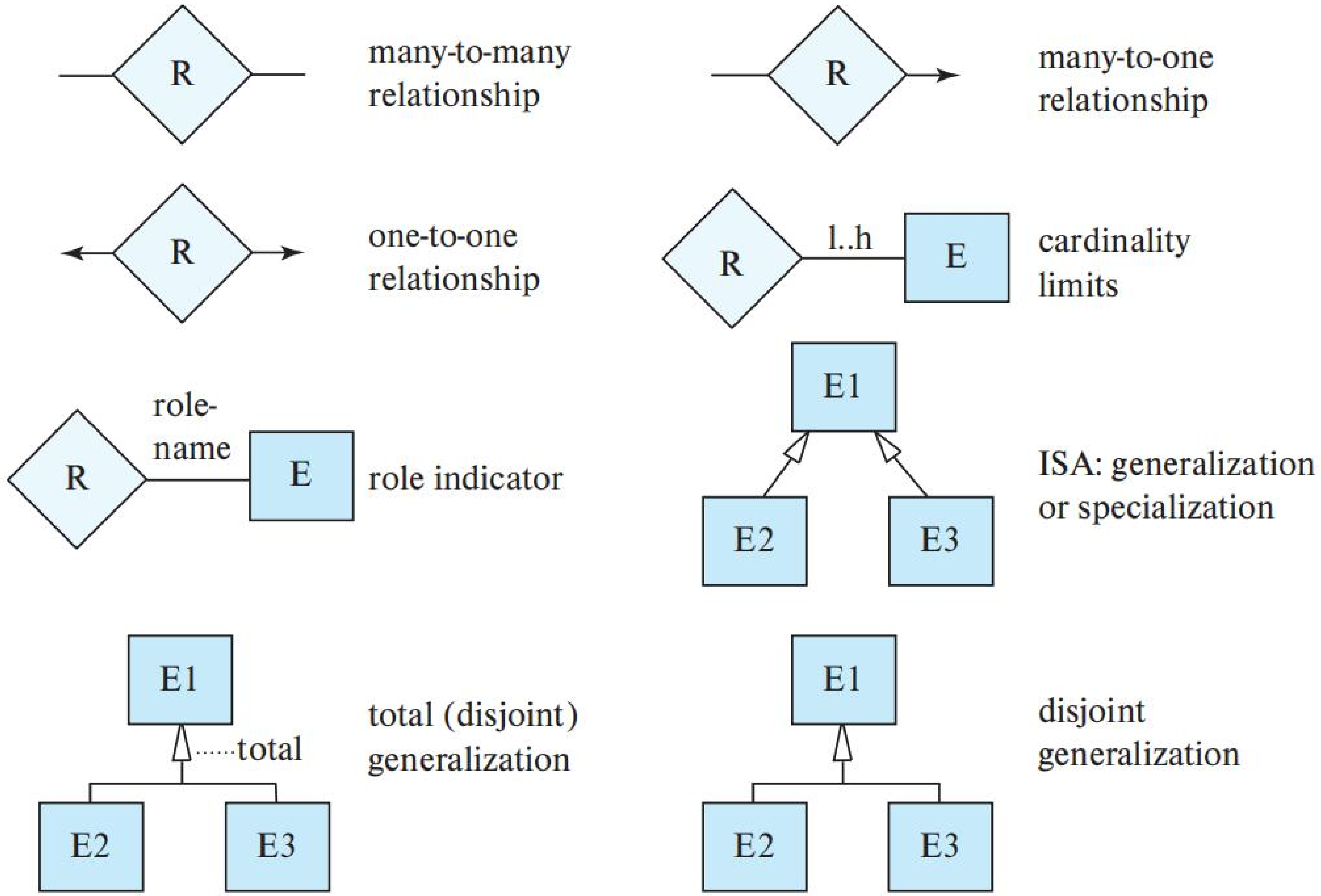
ER Design Decisions

- The use of an attribute or entity set to represent an object.
- Whether a real-world concept is best expressed by an entity set or a relationship set.
- The use of a ternary relationship versus a pair of binary relationships.
- The use of a strong or weak entity set.
- The use of specialization/generalization – contributes to modularity in the design.
- The use of aggregation – can treat the aggregate entity set as a single unit without concern for the details of its internal structure.

Summary of Symbols Used in ER Notation


E
entity set
R
relationship set
R
identifying relationship set for weak entity set
R
E
total participation of entity set in relationship
E
A1
A2
A2.1
A2.2
{A3}
A4()
attributes: simple (A1), composite (A2) and multivalued (A3) derived (A4)
E
A1
primary key
E
A1
discriminating attribute of weak entity set

Symbols Used in ER Notation (Cont.)



Steps in Database Design

- **Requirements Analysis**
  - User needs; what must database do?
- **Conceptual Design**
  - High level description (often done w/ER model)
- **Logical Design**
  - Translate ER into DBMS data model
- **Schema Refinement**
  - Consistency, normalization
- **Physical Design** - Indexes, disk layout
- **Security Design** - Who accesses what, and how

Schema Refinement

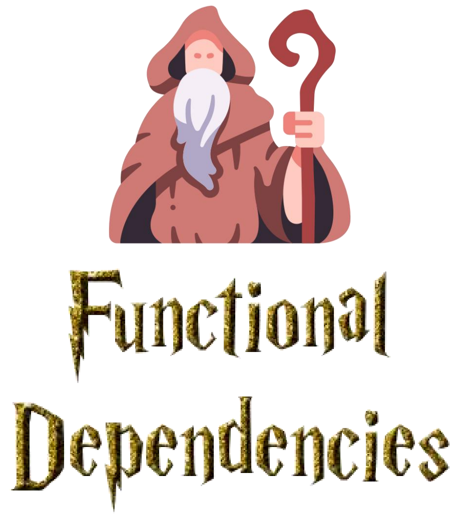
Overview of Schema Refinement

- The Evil to Avoid: Redundancy in your schema
  - i.e., replicated values
  - Leads to wasted storage
  - More important: insert/delete/update *anomalies*
    - *Replicated data + change = Trouble.*
    - *We'll see examples shortly*

Overview of Schema Refinement

- Solution: *Functional Dependencies*
    - A form of integrity constraints
    - Help identify *redundancy* in schemas
    - Help suggest refinements

- Main refinement technique: *Decomposition*
    - Split the columns of one table into two tables
    - Often good, but need to do this judiciously



Functional Dependencies (FDs)

- Idea: X →Y means
  - (Read "→" as "determines")
  - Given any two tuples in table r,
    if their X values are the same,
    then their Y values must be the same.
    (but not vice versa)

X	Y	Z
2	8	1
2	?	2
3	8	1

Functional Dependencies (FDs)

- Idea: X →Y means
    - (Read "→" as "determines")
    - Given any two tuples in table r,
      if their X values are the same,
      then their Y values must be the same.
      (but not vice versa)

X	Y	Z
2	8	1
2	8	2
3	8	1

Functional Dependencies (Cont.)

- Formally: An FD **X → Y** holds over relation schema R if, for every allowable instance r of R:

$$t1 \in r,\ t2 \in r,\ \pi_X(t1) = \pi_X(t2) \Rightarrow \pi_Y(t1) = \pi_Y(t2)$$

  - t1, t2 are tuples
  - X, Y are sets of attributes

- An FD is w.r.t. all allowable instances.
  - Declared based on application semantics
  - Not learned from data
    - (though you might learn suggestions for FDs)

Key Terminology (Important)

- Question: How are FDs related to keys?
  - Keys are special cases of FDs
  - $K \to$ {all attributes}
- Superkey: a set of columns in a table that determine all the columns in that table
  - $K \to$ {all attributes}
- Candidate Key: a **minimal** set of columns in a table that determine all the columns in the table
  - $K \to$ {all attributes}
  - For any $L \subset K$, $L \,!\!\to$ {all attributes} (minimal)
- Primary Key: a single chosen candidate key

- Notice: Index/sort "key" – columns used in an index or sort.
  - Unrelated to FDs, dependencies.

Example - Constraints on Entity Set

- Consider relation *Hourly_Emps*:

  Hourly_Emps(ssn, name, lot, rating, wage_per_hr, hrs_per_wk)
- We can denote a relation schema by listing its attributes:
  - e.g., SNLRWH
  - This is really the set of attributes {S,N,L,R,W,H}.
- And we can use relation name to refer to the set of all its attributes
  - e.g., "Hourly_Emps" for SNLRWH

- What are some FDs on Hourly_Emps?
  - **ssn** is the primary key: $S \rightarrow SNLRWH$
  - **rating** determines wage_per_hr: $R \rightarrow W$
  - **lot** determines lot: $L \rightarrow L$ ("trivial" dependency)

Problem 1 - Due to R → W

- *Update anomaly*: Can we modify *W* in only the 1st tuple of SNLRWH?

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Hourly_Emps

Then that tuple will be inconsistent with Smiley and Madayan!

Problem 2 - Due to R → W

- *Insertion anomaly*: What if we want to insert an employee and don’t know the hourly wage for his or her rating? (or we get it wrong?) e.g., insert a new employee with rating 6

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Hourly_Emps

Then you have to invent a value without reference to established truth!

Problem 3 - Due to R → W

- *Deletion anomaly*: If we delete all employees with rating 5, we lose the information about the wage for rating 5!

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Hourly_Emps

Then you will forget established truth!

Detecting Redundancy

- Q: Why is R → W problematic, but S →W not?

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Hourly_Emps

Detecting Redundancy

- Q: Why is R → W problematic, but S →W not?
- A: R is not a key, so any pair, e.g.,(8,10), appears multiple times.
  S is a **candidate key**, so each pair like (123-22-3666,10) is stored exactly once.

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

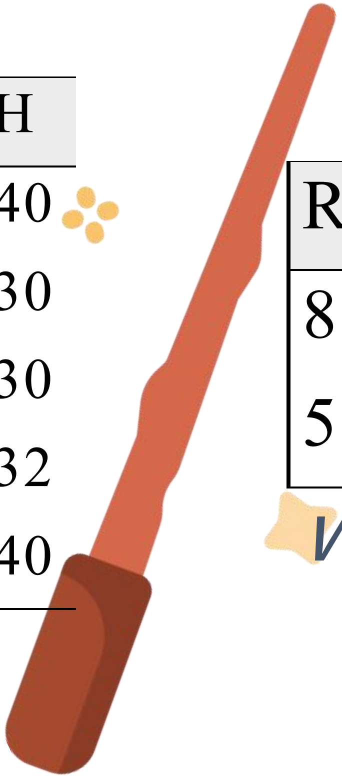
Hourly_Emps

Decomposing a Relation

- Redundancy can be removed by "chopping" the relation into pieces.
- FD's are used to drive this process.
  - R → W is causing the problems, so decompose SNLRWH into what relations?

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

Hourly_Emps2



R	W
8	10
5	7

Wages

Reasoning About FDs

- Given some FDs, we can usually infer additional FDs:

bookID → (publisher, author)	implies	bookID → publisher
	and	bookID → author
bookID → publisher and bookID → author	implies	bookID → (publisher,author)
bookID → author and author → publisher	implies	bookID → publisher

Reasoning About FDs

- Given some FDs, we can usually infer additional FDs:

bookID → (publisher, author)	implies	bookID → publisher
	and	bookID → author
bookID → publisher and bookID → author	implies	bookID → (publisher, author)
bookID → author and author → publisher	implies	bookID → publisher

- But,

  (title, author) → publisher does **NOT** necessarily imply that
  title → publisher **NOR** that author → publisher

Reasoning About FDs - General

- Generally, an FD g is *implied by* a set of FDs F if g holds whenever all FDs in F hold.

- F+ = closure of F
  - The set of all FDs that are implied by F.
  - Includes "trivial dependencies"

  - How do we get F+ given F?

Rules of Inference

- Armstrong's Axioms (X, Y, Z are sets of attributes):
  - *Reflexivity*: If $X \supseteq Y$, then $X \rightarrow Y$
  - *Augmentation*: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
  - *Transitivity*: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

William Ward Armstrong

Rules of Inference

- Armstrong's Axioms (X, Y, Z are sets of attributes):
  - *Reflexivity*: If $X \supseteq Y$, then $X \rightarrow Y$
  - *Augmentation*: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
  - *Transitivity*: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

William Ward Armstrong

- S*ound* and *complete* inference rules for FDs!
  - using AA you get *only* the FDs in F+ and *all* these FDs.

- Some additional rules (that follow from AA):
  - *Union*: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
  - *Decomposition*: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

Example

- Contracts(*<u>cid</u>, sid, jid, did, pid, qty, value*), and:
  - C is the key: C → CSJDPQV
  - Proj (J) purchases each part (P) using single contract (C): JP → C
  - Dept (D) purchases at most 1 part (P) from a supplier (S): SD → P

Given F

Attribute	Meaning
cid	contract
sid	supplier
jid	project
did	department
pid	part

Example

Given F

- Contracts(*<u>cid</u>, sid, jid, did, pid, qty, value*), and:
  - C is the key: C → CSJDPQV
  - Proj (J) purchases each part (P) using single contract (C): JP → C
  - Dept (D) purchases at most 1 part (P) from a supplier (S): SD → P

- **Problem: Prove that SDJ is a key for Contracts**

Attribute	Meaning
cid	contract
sid	supplier
jid	project
did	department
pid	part

Example

- Contracts(*cid, sid, jid, did, pid, qty, value*), and:
  - C is the key: C → CSJDPQV
  - Proj (J) purchases each part (P) using single contract (C): JP → C
  - Dept (D) purchases at most 1 part (P) from a supplier (S): SD → P

Given F

Attribute	Meaning
cid	contract
sid	supplier
jid	project
did	department
pid	part

- **Problem: Prove that SDJ is a key for Contracts**
  - JP → C, C → CSJDPQV
    - Imply JP → CSJDPQV
    - By transitivity (shows that JP is a key)

Example

Given F

- Contracts(*cid, sid, jid, did, pid, qty, value*), and:
  - C is the key: C → CSJDPQV
  - Proj (J) purchases each part (P) using single contract (C): JP → C
  - Dept (D) purchases at most 1 part (P) from a supplier (S): SD → P
- **Problem: Prove that SDJ is a key for Contracts**
  - JP → C, C → CSJDPQV
    - Imply JP → CSJDPQV
    - By transitivity (shows that JP is a key)
  - SD → P
    - Implies SDJ → JP (by augmentation)

Attribute	Meaning
cid	contract
sid	supplier
jid	project
did	department
pid	part

Example

Given F

- Contracts(*<u>cid</u>, sid, jid, did, pid, qty, value*), and:
  - C is the key: C → CSJDPQV
  - Proj (J) purchases each part (P) using single contract (C): JP → C
  - Dept (D) purchases at most 1 part (P) from a supplier (S): SD → P

Attribute	Meaning
cid	contract
sid	supplier
jid	project
did	department
pid	part

- **Problem: Prove that SDJ is a key for Contracts**
  - JP → C, C → CSJDPQV
    - Imply JP → CSJDPQV
    - By transitivity (shows that JP is a key)
  - SD → P
    - Implies SDJ → JP (by augmentation)
  - SDJ → JP, JP → CSJDPQV
    - Imply SDJ → CSJDPQV
    - By transitivity (shows that SDJ is a key)

Example

Given F

- Contracts(*cid*, *sid*, *jid*, *did*, *pid*, *qty*, *value*), and:
  - C is the key: C → CSJDPQV
  - Proj (J) purchases each part (P) using single contract (C): JP → C
  - Dept (D) purchases at most 1 part (P) from a supplier (S): SD → P

Attribute	Meaning
cid	contract
sid	supplier
jid	project
did	department
pid	part

- **Problem: Prove that SDJ is a key for Contracts**
  - JP → C, C → CSJDPQV
    - Imply JP → CSJDPQV
    - By transitivity (shows that JP is a key)
  - SD → P
    - Implies SDJ → JP (by augmentation)
  - SDJ → JP, JP → CSJDPQV
    - Imply SDJ → CSJDPQV
    - By transitivity (shows that SDJ is a key)

Q: Can you now infer that SD → CSDPQV (i.e., drop J on both sides)?

Example

Given F

- Contracts(*cid, sid, jid, did, pid, qty, value*), and:
  - C is the key: C → CSJDPQV
  - Proj (J) purchases each part (P) using single contract (C): JP → C
  - Dept (D) purchases at most 1 part (P) from a supplier (S): SD → P

Attribute	Meaning
cid	contract
sid	supplier
jid	project
did	department
pid	part

- **Problem: Prove that SDJ is a key for Contracts**
  - JP → C, C → CSJDPQV
    - Imply JP → CSJDPQV
    - By transitivity (shows that JP is a key)
  - SD → P
    - Implies SDJ → JP (by augmentation)
  - SDJ → JP, JP → CSJDPQV
    - Imply SDJ → CSJDPQV
    - By transitivity (shows that SDJ is a key)

Q: Can you now infer that SD → CSDPQV (i.e., drop J on both sides)?
A: NO!

Attribute Closure

- Computing closure F^+ of a set of FDs F is hard:
  - exponential in # attrs!

Attribute Closure

- Computing closure F^+ of a set of FDs F is hard:
  - exponential in # attrs!

- Typically, just check if $X \rightarrow Y$ is in F+. Efficient!

Attribute Closure

- Computing closure F^+ of a set of FDs F is hard:
  - exponential in # attrs!

- Typically, just check if $X \rightarrow Y$ is in $F+$. Efficient!
  - Compute *attribute closure* of X (denoted X^+) wrt F.
    X^+ = Set of all attributes A such that $X \rightarrow A$ is in F^+
    - $X^+ := X$
    - Repeat until no change (fixpoint):

      ```
      for $U \rightarrow V \subseteq F$,
          if $U \subseteq X^+$, then add $V$ to $X^+$
      ```
  - Check if Y is in X^+
  - If Y is in X^+, then $X \rightarrow Y$ is in $F+$

Attribute Closure (Cont.)

- Typically, just check if $X \to Y$ is in F+. Efficient!
  - Compute *attribute closure* of X (denoted X^+) wrt F.
    X^+ = Set of all attributes A such that $X \to A$ is in F^+
    - $X^+ := X$
    - Repeat until no change (fixpoint):

      ```
              for $U \to V \subseteq F$,
                  if $U \subseteq X^+$, then add $V$ to $X^+$
      ```
  - Check if Y is in X^+
  - If Y is in X+, then $X \to Y$ is in F+

- The above approach can also be used to check for keys of a relation R.
  - If $X^+ = R$, then X is a superkey for R. $X^+ = R$ *means* X^+ = {all attributes of R}
  - Q: How to check if X is a "candidate key" (minimal)?
  - A: For each attribute A in X, check if $(X - A)^+ = R$. If $(X - A)^+ != R$ for every A in X, then X is a minimal superkey, i.e., a candidate key of R.

Attribute Closure - Example

- $R = \{A, B, C, D, E\}$
- $F = \{B \rightarrow CD, D \rightarrow E, B \rightarrow A, E \rightarrow C, AD \rightarrow B\}$

Attribute Closure - Example

- $R = \{A, B, C, D, E\}$
- $F = \{B \rightarrow CD, D \rightarrow E, B \rightarrow A, E \rightarrow C, AD \rightarrow B\}$

- **Is B → E in F+ ?**
  B+ = …

Attribute Closure - Example

- $R = \{A, B, C, D, E\}$
- $F = \{B \rightarrow CD, D \rightarrow E, B \rightarrow A, E \rightarrow C, AD \rightarrow B\}$

- **Is $B \rightarrow E$ in F+ ?**
  B+ = {B, C, D, E, …}
  … Yep!

- **Is D a key for R?**
  D+ = …

Attribute Closure - Example

- R = {A, B, C, D, E}
- F = {B → CD, D → E, B → A, E → C, AD → B}

- **Is B → E in F+ ?**
  B+ = {B, C, D, E, …}
  … Yep!

- **Is D a key for R?**
  D+ = {D, E, C}
  … Nope!

- **Is AD a key for R?**
  AD^+ = …

Attribute Closure - Example

- R = {A, B, C, D, E}
- F = {B $\rightarrow$ CD, D $\rightarrow$ E, B $\rightarrow$ A, E $\rightarrow$ C, AD $\rightarrow$ B}

- **Is B $\rightarrow$ E in F+ ?**
  B+ = {B, C, D, E, …}
  … Yep!

- **Is D a key for R?**
  D+ = {D, E, C}
  … Nope!

- **Is AD a key for R?**
  AD^+ = {A, D, E, C, B}
  …Yep!

- **Is AD a candidate key for R?**
  A^+ = … D^+ = …

Attribute Closure - Example

- R = {A, B, C, D, E}
- F = {B → CD, D → E, B → A, E → C, AD → B}

- **Is B → E in F+ ?**
  B+ = {B, C, D, E, ...}
  ... Yep!

- **Is D a key for R?**
  D+ = {D, E, C}
  ... Nope!

- **Is AD a key for R?**
  AD^+ = {A, D, E, C, B}
  ...Yep!

- **Is AD a candidate key for R?**
  A^+ = {A} D^+ = {D, E, C}
  ...Yes!

- **Is ADE a candidate key for R?**

Attribute Closure - Example

- $R = \{A, B, C, D, E\}$
- $F = \{B \rightarrow CD, D \rightarrow E, B \rightarrow A, E \rightarrow C, AD \rightarrow B\}$

- **Is $B \rightarrow E$ in F+ ?**
  $B+ = \{B, C, D, E, \ldots\}$
  ... Yep!

- **Is D a key for R?**
  $D+ = \{D, E, C\}$
  ... Nope!

- **Is AD a key for R?**
  $AD^{+} = \{A, D, E, C, B\}$
  ...Yep!

- **Is AD a candidate key for R?**
  $A^{+} = \{A\}$ $D^{+} = \{D, E, C\}$
  ...Yes!

- **Is ADE a candidate key for R?**
  No!

Thanks for that…

- So we know a lot about FDs
- Can they help with schema refinement?

The Notion of Normal Forms

- Q1: given a schema and some FDs, is there any refinement needed?

The Notion of Normal Forms

- Q1: given a schema and some FDs, is there any refinement needed?

- If relation is in a *normal form* (e.g. BCNF):
  - We know certain problems are avoided/minimized.
  - Helps decide whether decomposing relation is useful.

The Notion of Normal Forms

- Q1: given a schema and some FDs, is there any refinement needed?

- If relation is in a *normal form* (e.g. BCNF):
  - We know certain problems are avoided/minimized.
  - Helps decide whether decomposing relation is useful.

- Consider a relation R with 3 attributes, ABC.
  - No (non-trivial) FDs hold: No redundancy here.
  - Given A → B: If A is not a key, then several tuples could have the same A value, and if so, they'll all have the same B value!

Basic Normal Forms

- 1st Normal Form – all attributes atomic
  - I.e. relational model
  - Violated by many common data models
    - Including XML, JSON, various OO models
  - Some of these “non-first-normal form” (NFNF) quite useful in various settings
    - Especially in update-never settings – e.g., data tranfer
    - If you never “unnest”, then who cares!
      - Basically relational collection of structured objects

- 1st ⊃ 2nd (of historical interest)
  ⊃ 3rd
  ⊃ Boyce-Codd …

Boyce-Codd Normal Form (BCNF)

- Relation R with FDs F is in BCNF if, for all X → A in F+
  - A ⊆ X (called a trivial FD), or
  - X is a superkey for R.

- In other words: “R is in BCNF if the only non-trivial FDs over R are key constraints.”

Boyce-Codd Normal Form (BCNF)

- Relation R with FDs F is in BCNF if, for all X → A in F+
  - A ⊆ X (called a trivial FD), or
  - X is a superkey for R.

- In other words: “R is in BCNF if the only non-trivial FDs over R are key constraints.”

- Q: How to know if a set of attributes X is superkey of R?

Why is BCNF Useful?

- If R is in BCNF, every field of every tuple stores useful info that cannot be inferred via FDs alone.
  - Say we know FD $X \rightarrow A$ holds for this example relation:
  - Can you guess the value of the missing attribute?
  - Yes, so relation is not in BCNF

X	Y	A
x	y1	a
x	y2	?

Decomposition of a Relation Scheme

- How to normalize a relation?
  - *Decompose* into multiple normalized relations

- Suppose R contains attributes $A_1 \ldots A_n$.

- A *decomposition* of R consists of replacing R by two or more relations such that:
  - Each new relation scheme contains a subset of the attributes of R, and
  - Every attribute of R appears as an attribute of at least one of the new relations.

Example

- SNLRWH has FDs S → SNLRWH and R → W
- Q: Is this relation in BCNF?
  - No. The second FD causes a violation; R is not a superkey.
  - W values repeatedly associated with R values.

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Hourly_Emps

Decomposing a Relation

- Easiest fix is to create a relation RW to store these associations, and to remove W from the main schema:

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

Hourly_Emps2

R	W
8	10
5	7

Wages

- Q: Are both of these relations are now in BCNF?

Decomposing a Relation

- Easiest fix is to create a relation RW to store these associations, and to remove W from the main schema:

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

Hourly_Emps2

R	W
8	10
5	7

Wages

- Q: Are both of these relations are now in BCNF?
- A: Yes. S → SNLRH is ok, as is R → W.

Problems with Decompositions

- There are three potential problems to consider:

  1) May be ***impossible*** to reconstruct the original relation! (Lossiness)
     - Fortunately, not in the SNLRWH example.

  2) Dependency checking may require joins.
     - Fortunately, not in the SNLRWH example.

  3) Some queries become more expensive.
     - e.g., How much does Guldu earn?

Tradeoff: Must consider these 3 problems vs. redundancy.