



# Chapter 7: Entity-Relationship Model

**Database System Concepts, 7<sup>th</sup> Ed.**

©Silberschatz, Korth and Sudarshan

See [www.db-book.com](http://www.db-book.com) for conditions on re-use



# Chapter 7: Entity-Relationship Model

- Design Process
- Modeling
- Constraints
- E-R Diagram
- Design Issues
- Weak Entity Sets
- Extended E-R Features
- Design of the Bank Database
- Reduction to Relation Schemas
- Database Design
- UML



# Design Phases

- The initial phase of database design is to characterize fully the data needs of the prospective database users.
- Next, the designer chooses a data model and, by applying the concepts of the chosen data model, translates these requirements into a conceptual schema of the database.
- A fully developed conceptual schema also indicates the functional requirements of the enterprise. In a “specification of functional requirements”, users describe the kinds of operations (or transactions) that will be performed on the data.



# Design Phases (Cont.)

The process of moving from an abstract data model to the implementation of the database proceeds in two final design phases.

- Logical Design – Deciding on the database schema. Database design requires that we find a “good” collection of relation schemas.
  - Business decision – What attributes should we record in the database?
  - Computer Science decision – What relation schemas should we have and how should the attributes be distributed among the various relation schemas?
- Physical Design – Deciding on the physical layout of the database



# Design Approaches

- Entity Relationship Model (covered in this chapter)
  - 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 (Chapter 8)
  - Formalize what designs are bad, and test for them



# Outline of the ER Model



# ER model -- Database Modeling

- The ER data model was developed to facilitate database design by allowing specification of an **enterprise schema** that represents the overall logical structure of a database.
- The 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:
  - 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:  
 $instructor = (ID, name, street, city, salary)$   
 $course = (course\_id, title, credits)$
- A subset of the attributes form a **primary key** of the entity set; i.e., uniquely identifying each member of the set.





# 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*



# Relationship Sets

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

Example:

44553 (Peltier)	<u>advisor</u>	22222 ( <u>E</u> instein)
<i>student</i> entity	relationship set	<i>instructor</i> entity

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

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

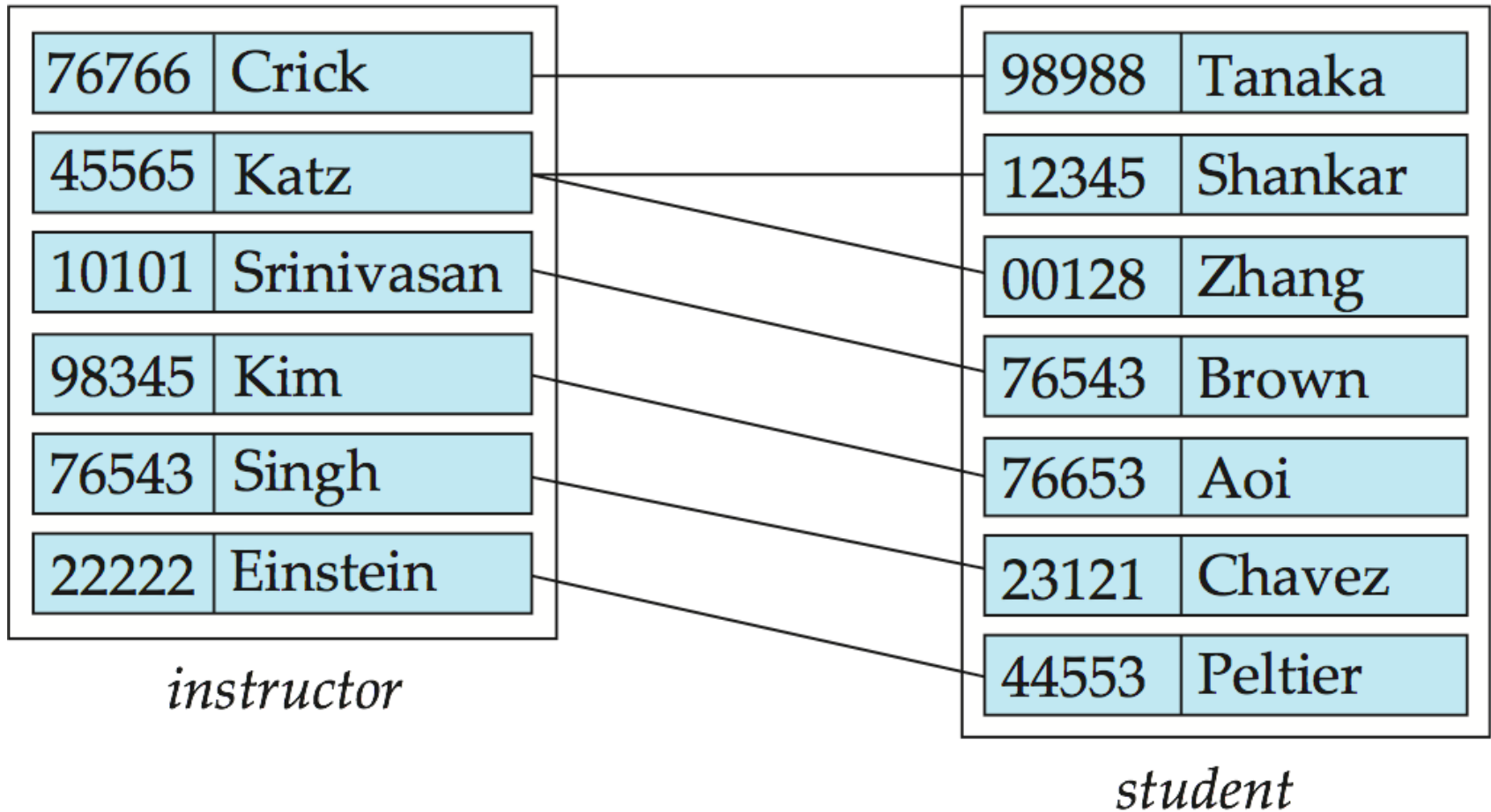
where  $(e_1, e_2, \dots, e_n)$  is a relationship

- Example:

$$(44553, 22222) \in \text{advisor}$$



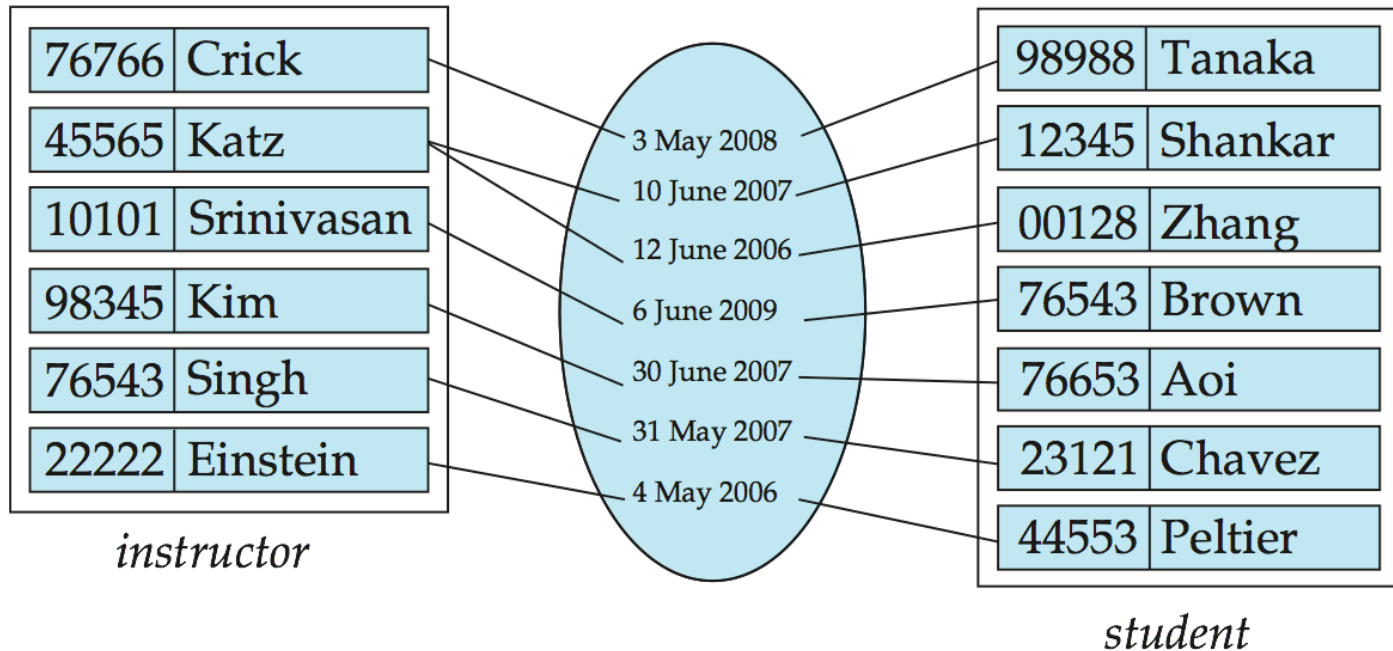
# Relationship Set *advisor*





# 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





# 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*

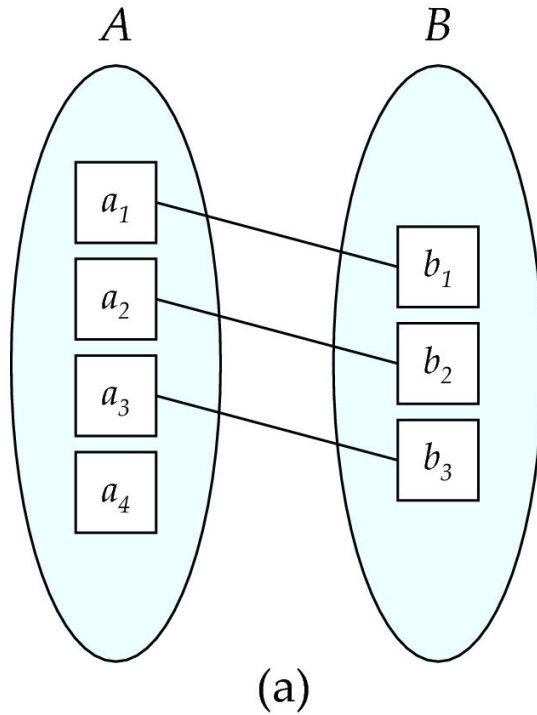


# Mapping Cardinality Constraints

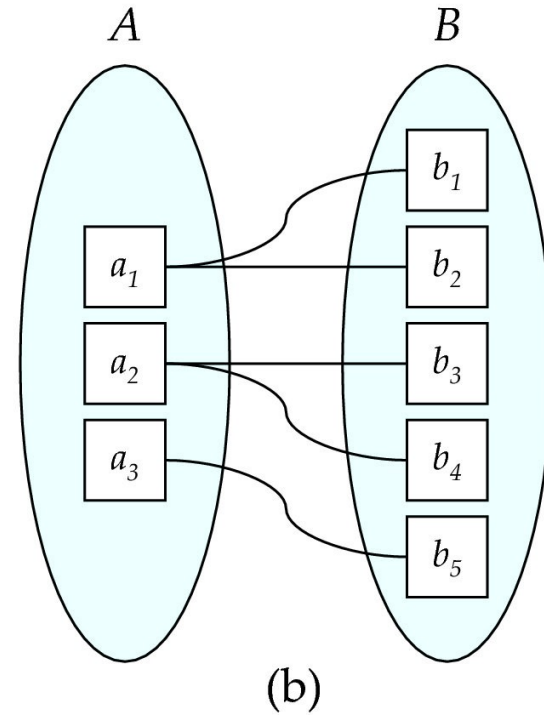
- Express the number of entities to which another entity can be associated via a relationship set.
- Most useful in describing binary relationship sets.
- For a binary relationship set the mapping cardinality must be one of the following types:
  - One to one
  - One to many
  - Many to one
  - Many to many



# Mapping Cardinalities



One to one

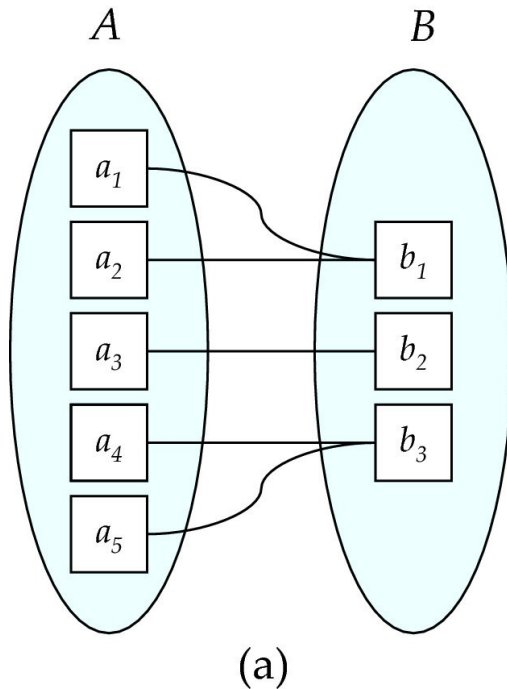


One to many

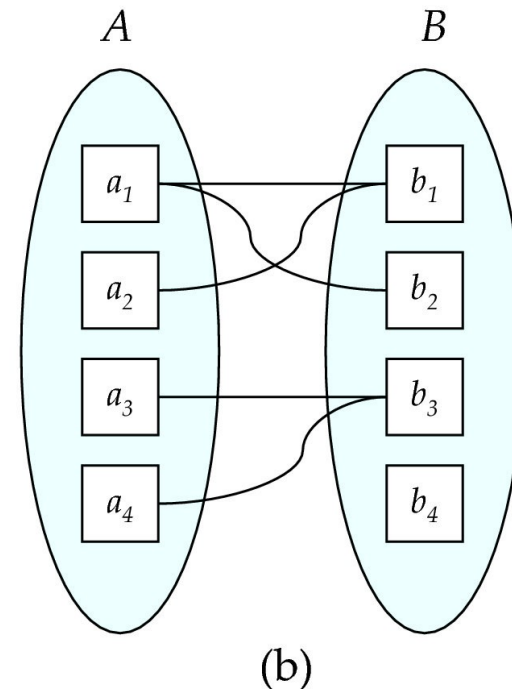
Note: Some elements in  $A$  and  $B$  may not be mapped to any elements in the other set



# Mapping Cardinalities



Many to one



Many to many

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





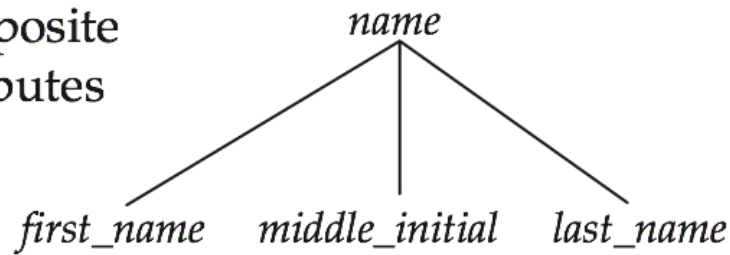
# Complex Attributes

- 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, given date\_of\_birth
- **Domain** – the set of permitted values for each attribute

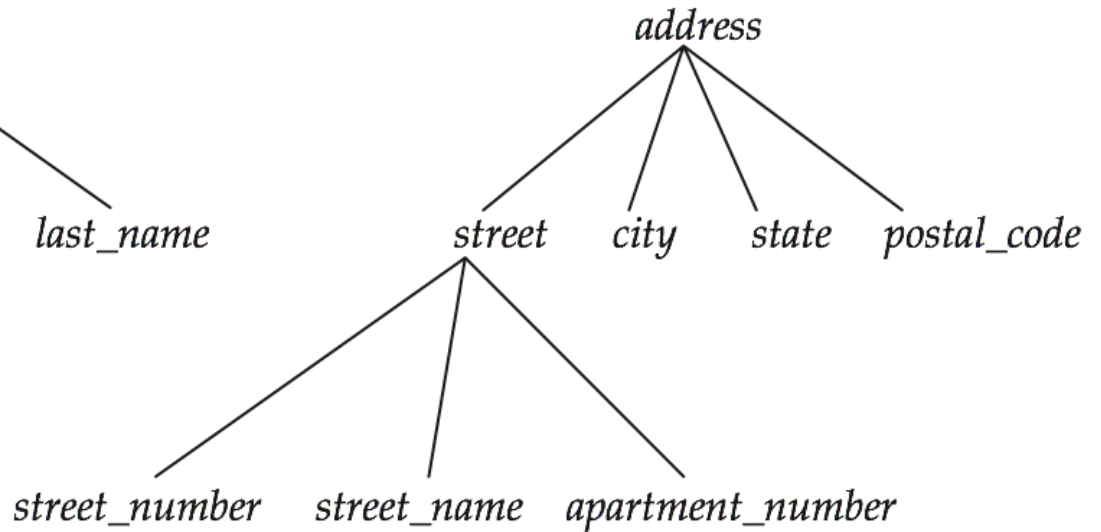


# Composite Attributes

composite  
attributes



component  
attributes





# 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.



# 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.



# 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; although each *section* entity is distinct, sections for different courses may share the same *section\_id*, *year*, and *semester*.
- 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.
- The notion of **weak entity set** formalizes the above intuition. 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. An entity set that is not a weak entity set is termed a **strong entity set**.



# Weak Entity Sets (Cont.)

- 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*.

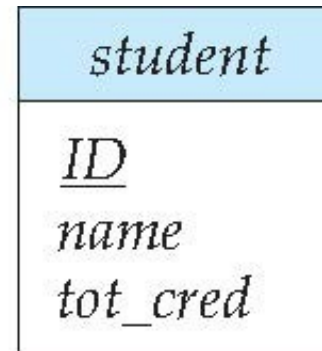
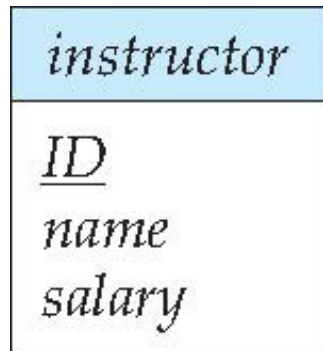


# E-R Diagrams



# Entity Sets

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

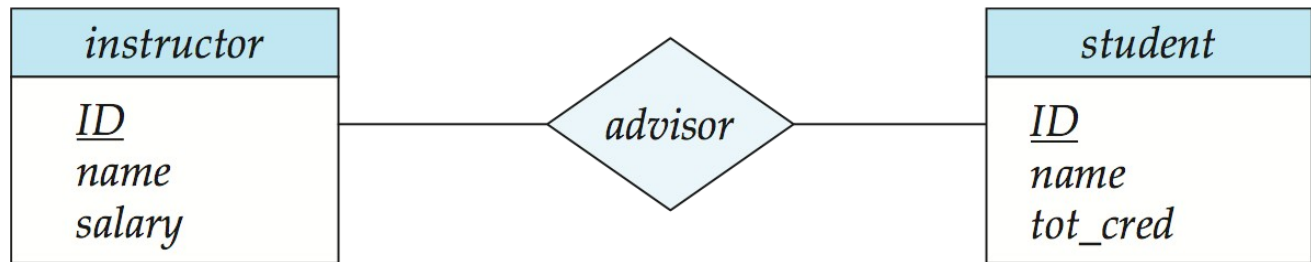






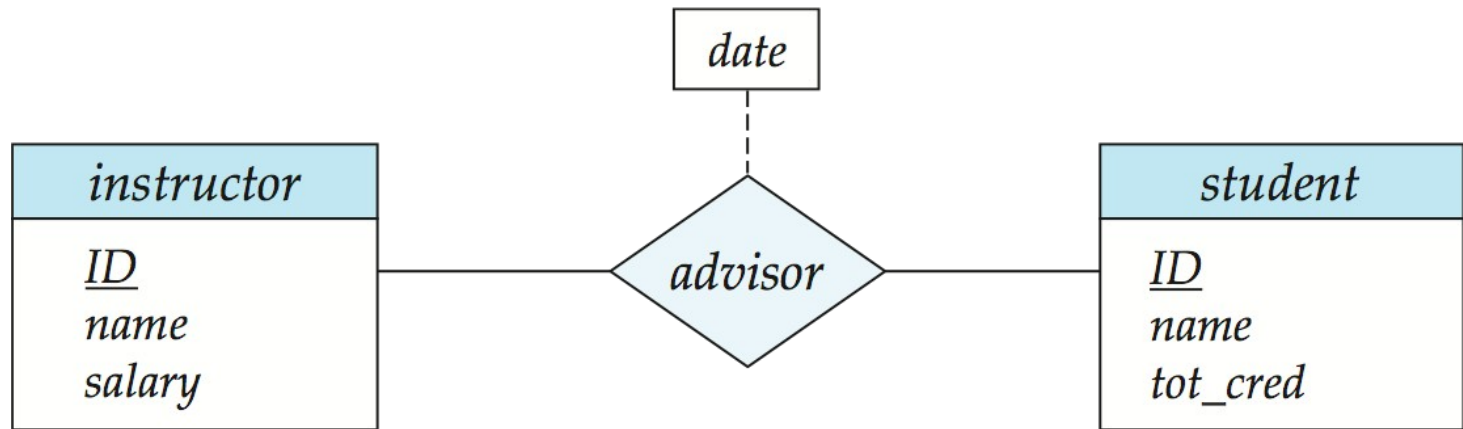
# Relationship Sets

- Diamonds represent relationship sets.





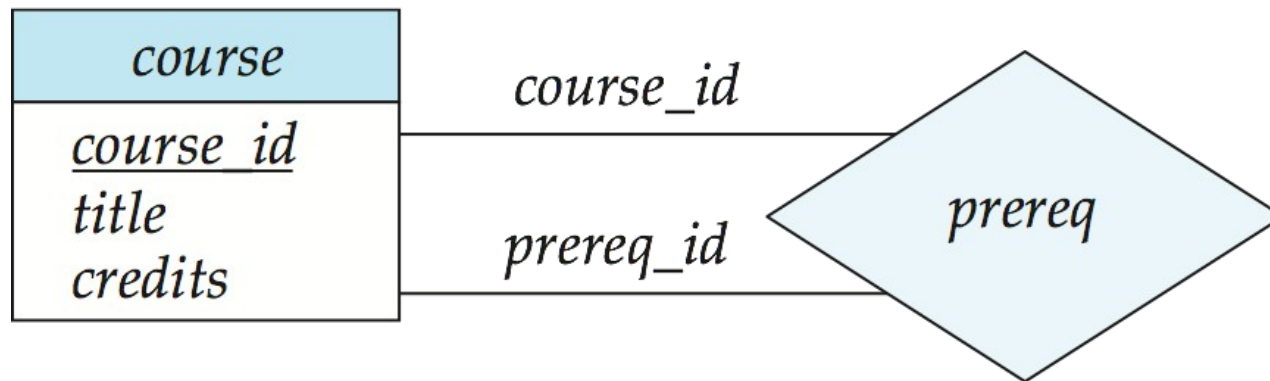
# Relationship Sets with Attributes





# Roles

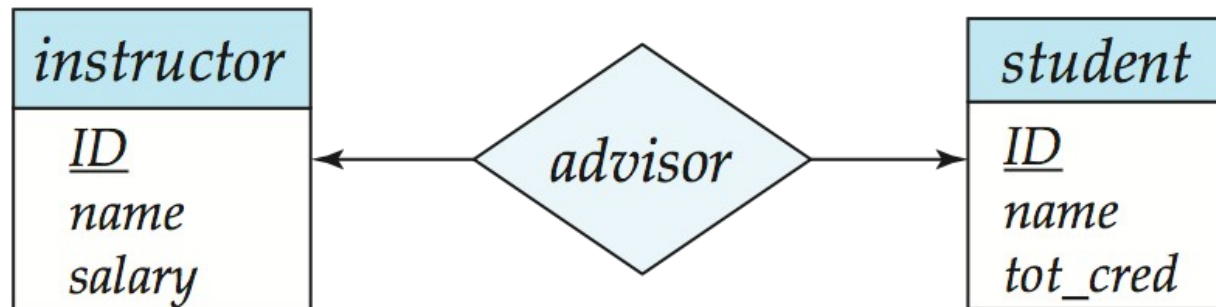
- Entity sets of a relationship need not be distinct
  - Each occurrence of an entity set plays a “role” in the relationship
- The labels “*course\_id*” and “*prereq\_id*” are called **roles**.





# Cardinality Constraints

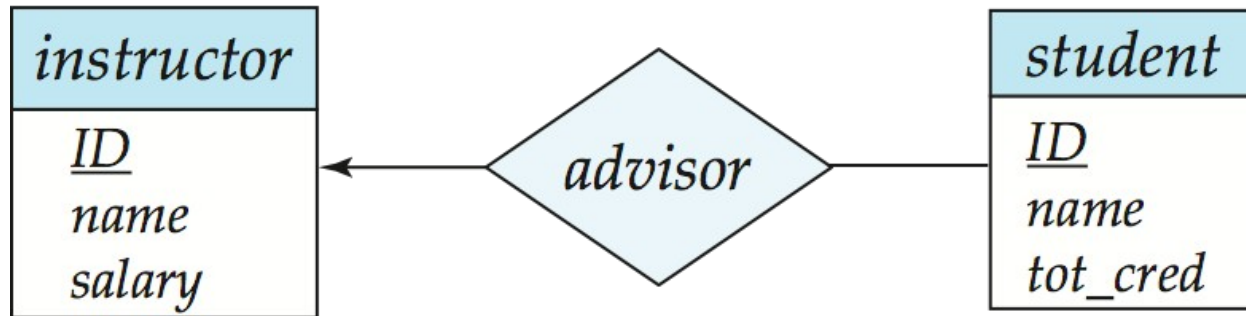
- We express cardinality constraints by drawing either a directed line ( $\rightarrow$ ), signifying “one,” or an undirected line ( $—$ ), signifying “many,” between the relationship set and the entity set.
- One-to-one relationship between an *instructor* and a *student* :
  - A student is associated 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

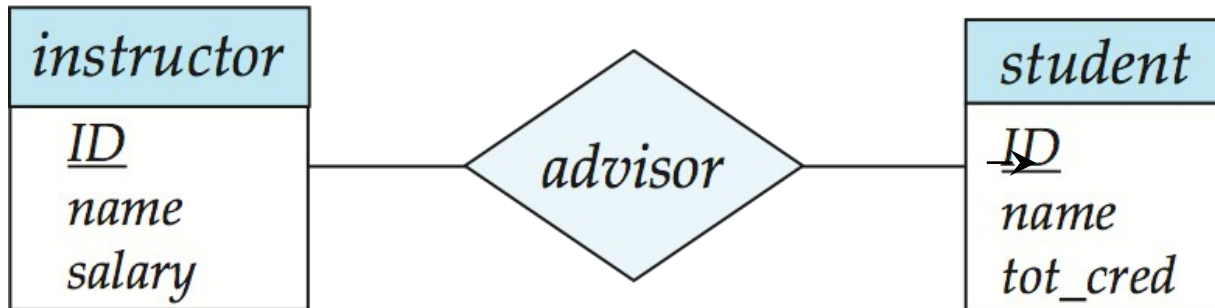
- one-to-many relationship between an *instructor* and a *student*
  - an instructor is associated with several (including 0) students via *advisor*
  - a student is associated with at most one instructor via *advisor*,





# Many-to-One Relationships

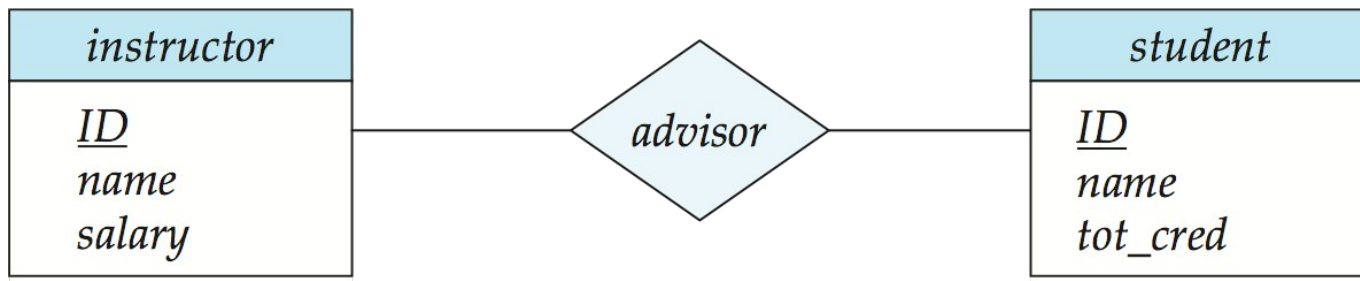
- In a many-to-one relationship between an *instructor* and a *student*,
  - an instructor is associated with at most one student via *advisor*,
  - and a student is associated with several (including 0) instructors via *advisor*





# Many-to-Many Relationship

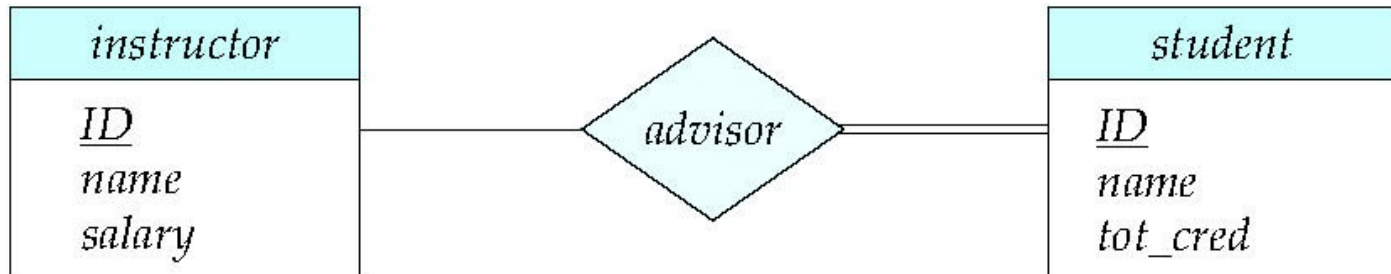
- An instructor is associated with several (possibly 0) students via *advisor*
- A student is associated with several (possibly 0) instructors via *advisor*





# 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

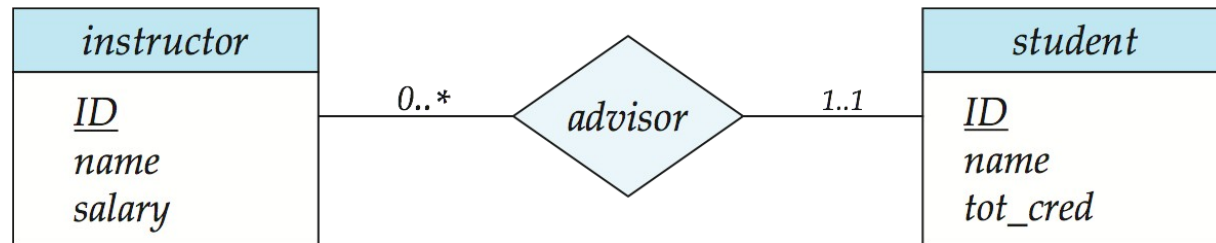
- ▶ every *student* must have an associated instructor
- Partial participation: some entities may not participate in any relationship in the relationship set
  - Example: participation of *instructor* in *advisor* is partial





# Notation for Expressing More Complex Constraints

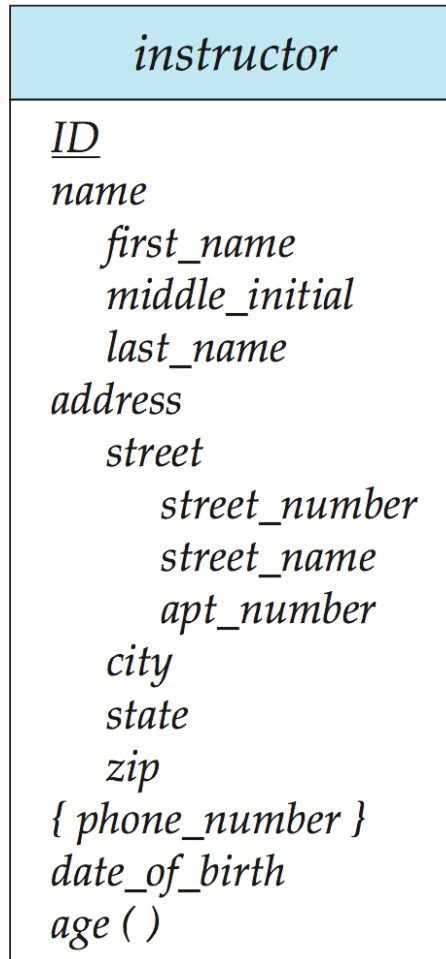
- A line may have an associated minimum and maximum cardinality, shown in the form  $l..h$ , where  $l$  is the minimum and  $h$  the maximum cardinality
  - A minimum value of 1 indicates total participation.
  - A maximum value of 1 indicates that the entity participates in at most one relationship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



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



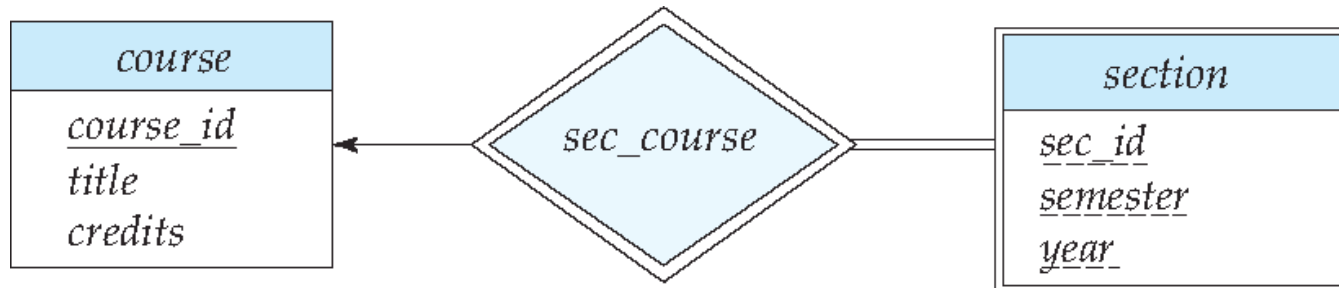
# Notation to Express Entity with Complex Attributes





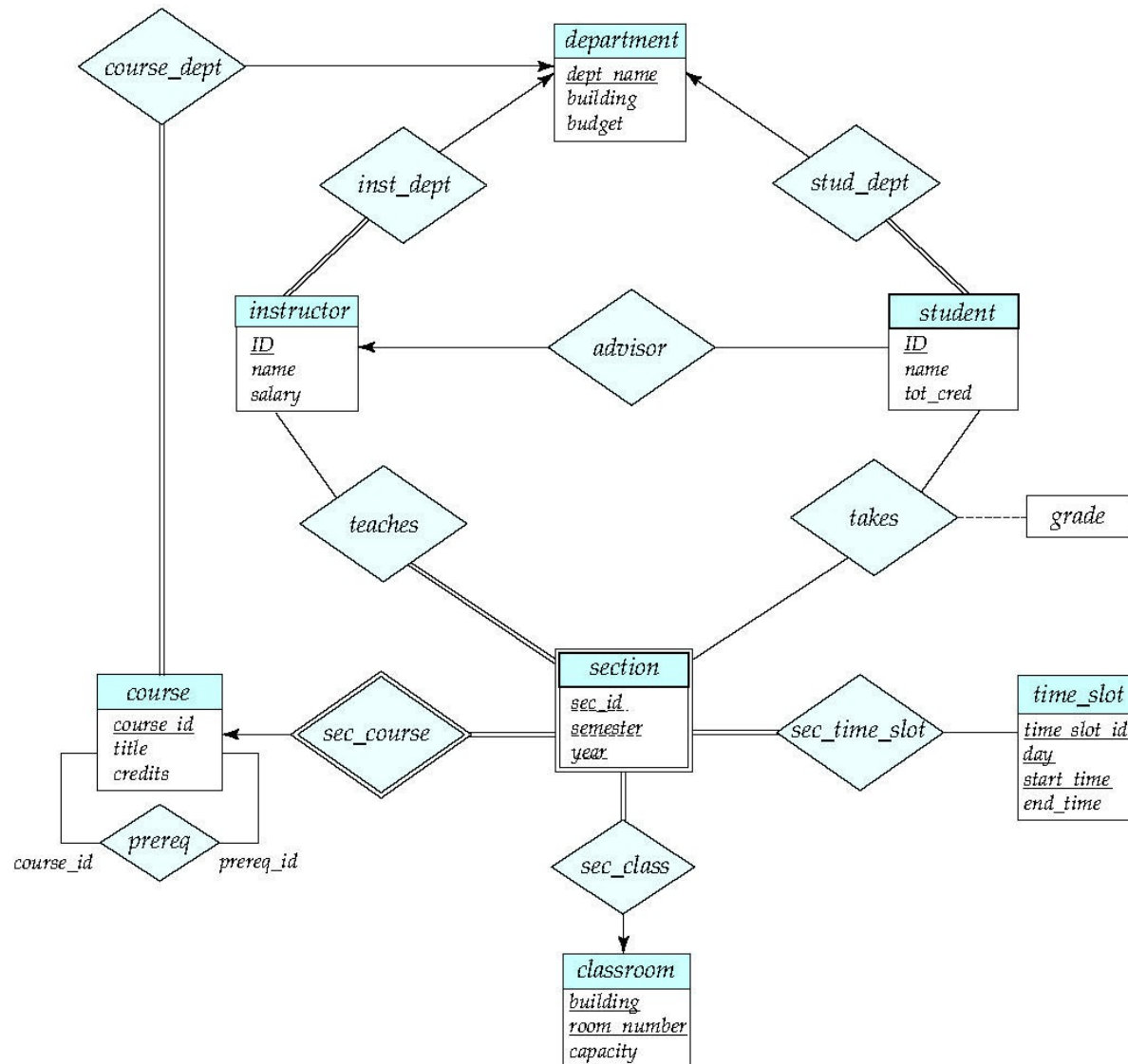
# Expressing Weak Entity Sets

- 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*)





# E-R Diagram for a University Enterprise





# Reduction to Relation Schemas



# 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.



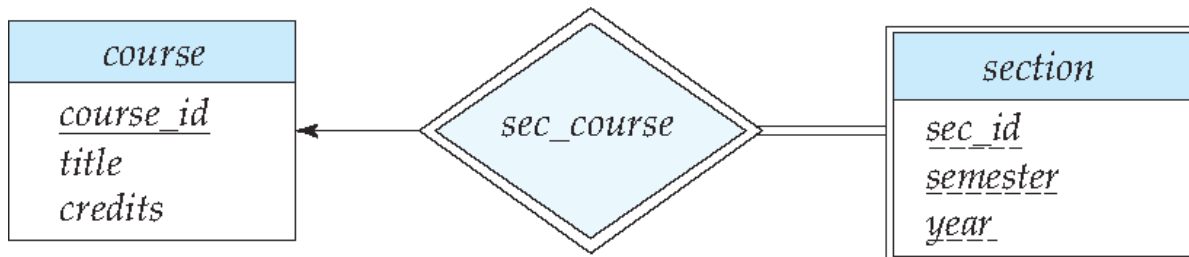
# Representing Entity Sets

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

*student(ID, 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 ( course\_id, sec\_id, sem, year )*

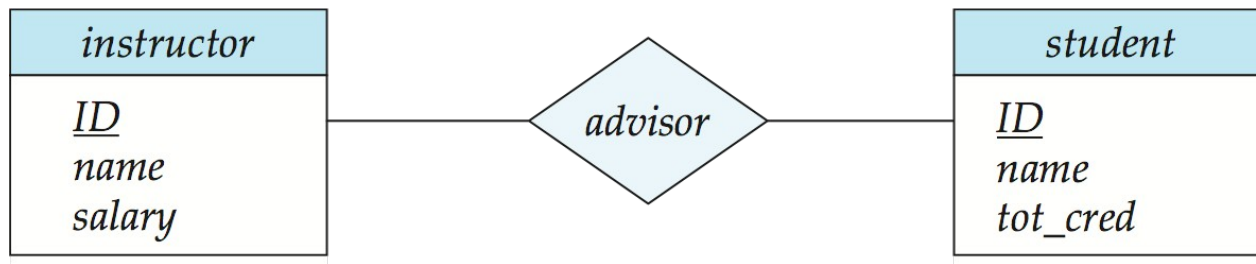




# Representing Relationship Sets

- 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*

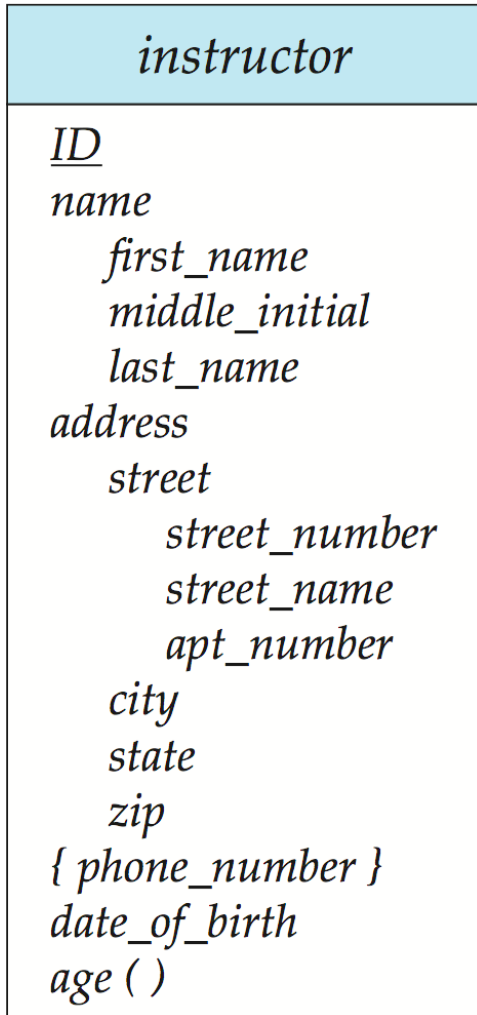
$advisor = (\underline{s\_id}, i\_id)$







# 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*)



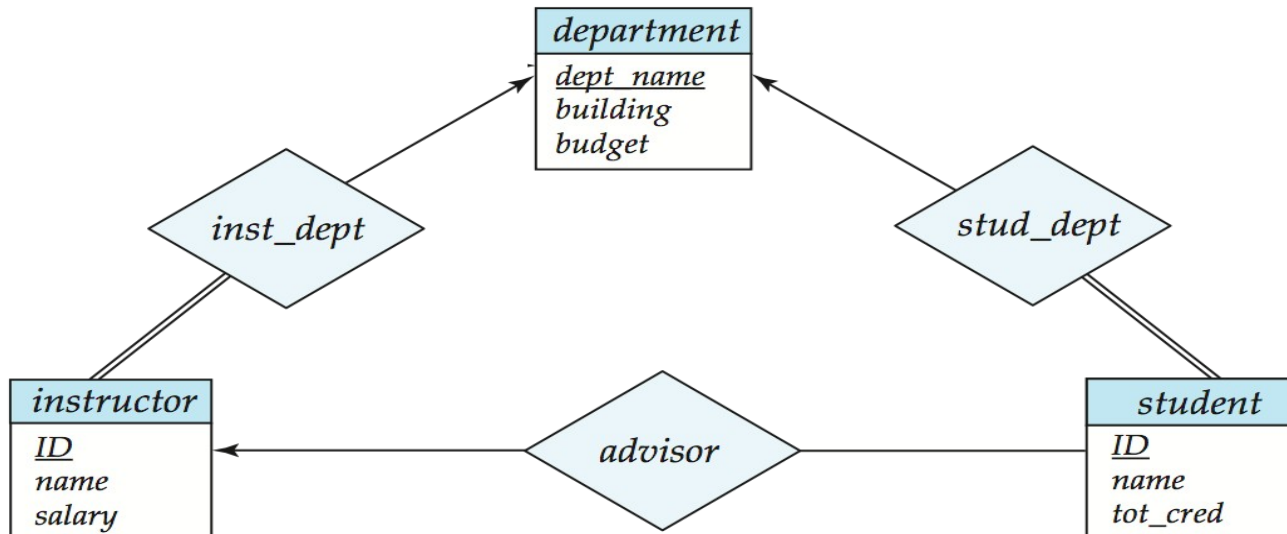
# Representation of Entity Sets with Multivalued Attributes

- 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:  
 $inst\_phone = ( \underline{ID}, \underline{phone\_number} )$
- 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)



# 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*





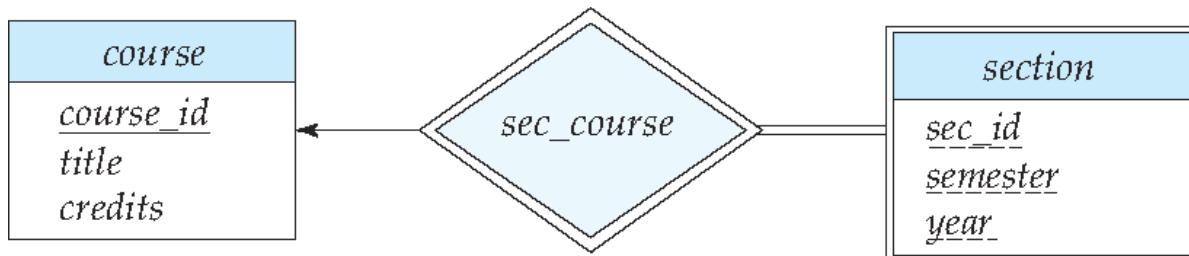
# Redundancy of Schemas (Cont.)

- 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



# 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



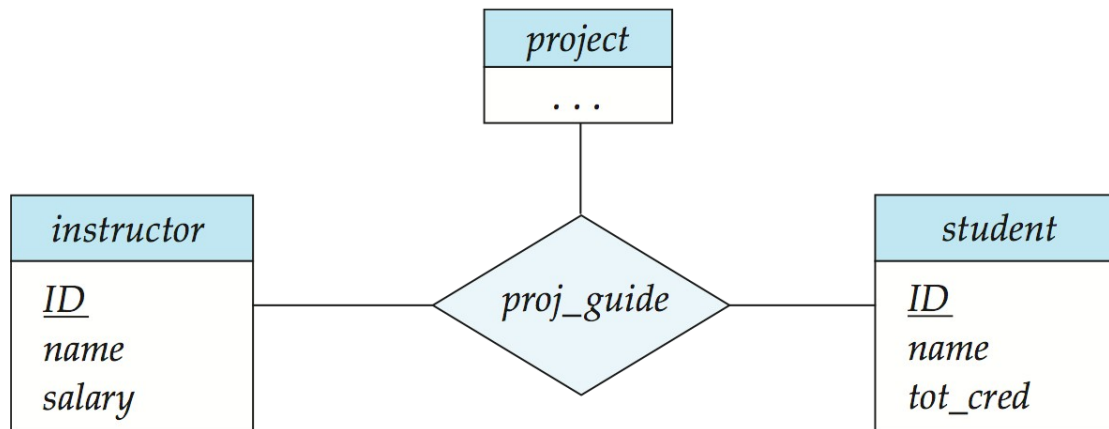


# Advanced Topics



# Non-binary Relationship Sets

- Most relationship sets are binary
- There are occasions when it is more convenient to represent relationships as non-binary.
- E-R Diagram with a Ternary Relationship





# 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





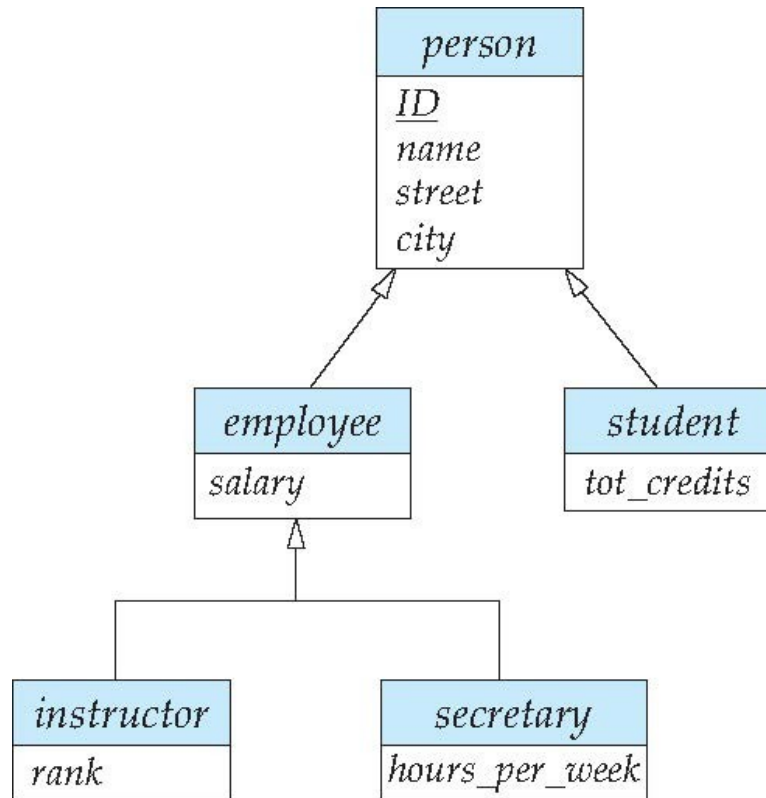
# Specialization

- 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.



# Specialization Example

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





# 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

<u>schema</u>	<u>attributes</u>
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

<u>schema</u>	<u>attributes</u>
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



# 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.



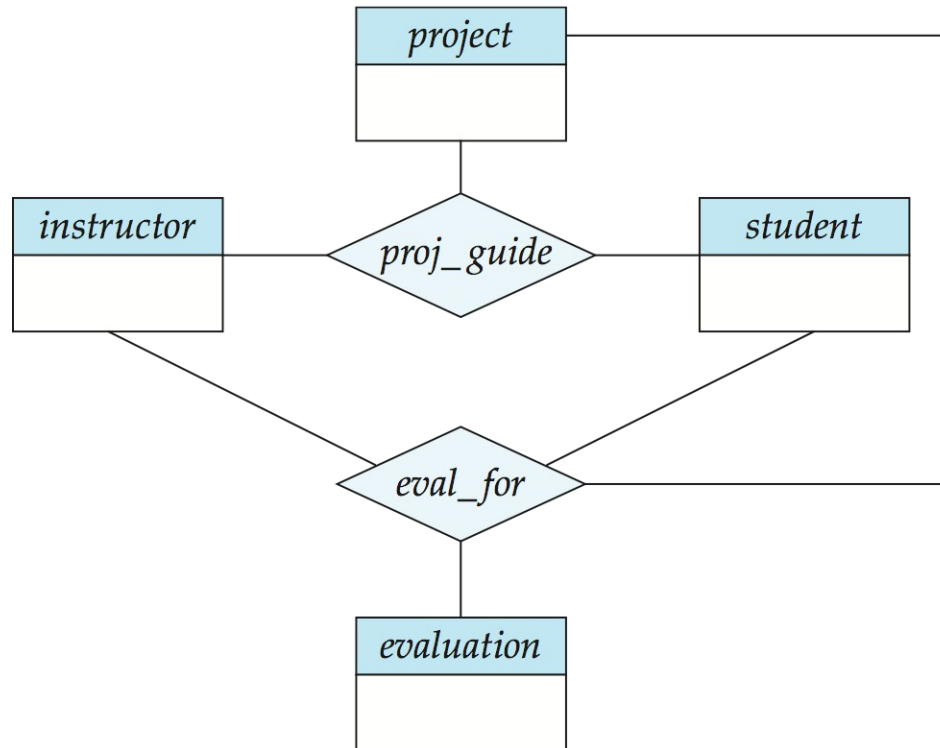
# Design Constraints on a Specialization/Generalization

- **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
- 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



# 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.)

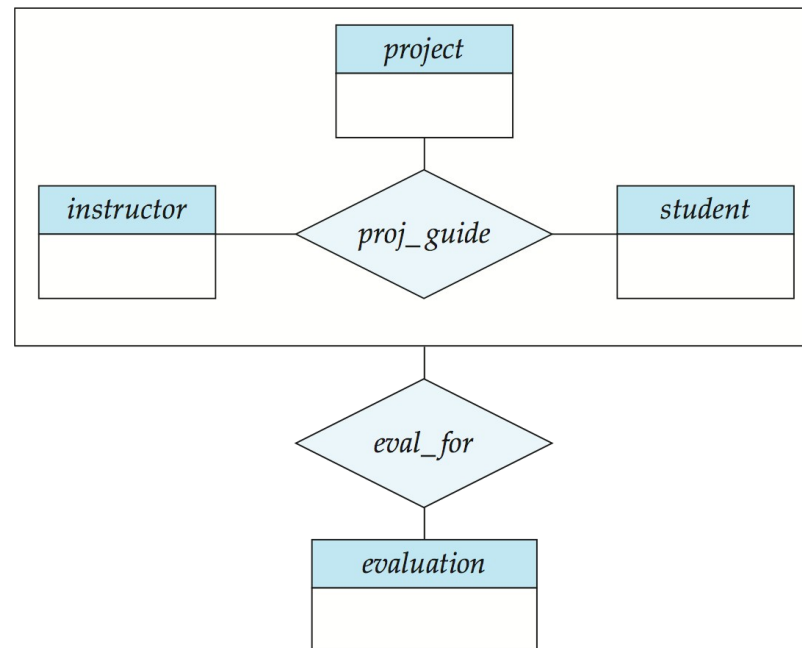
- Relationship sets *eval\_for* and *proj\_guide* represent overlapping information
  - Every *eval\_for* relationship corresponds to a *proj\_guide* relationship
  - However, some *proj\_guide* relationships may not correspond to any *eval\_for* relationships
    - ▶ So we can't discard the *proj\_guide*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





# 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





# Representing Aggregation via 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.

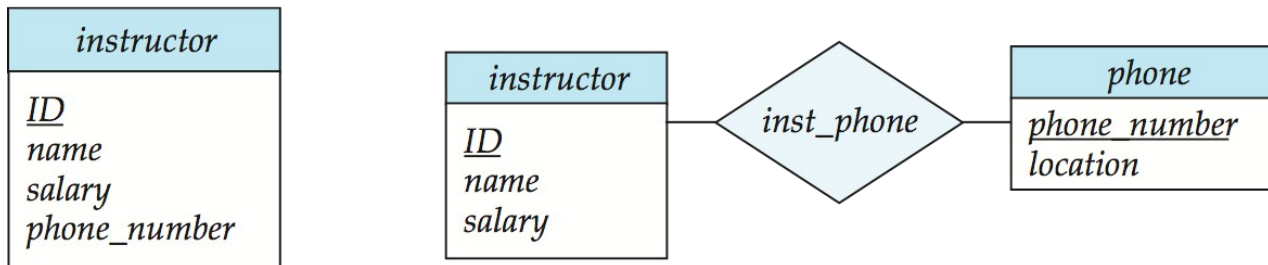


# Design Issues



# 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)

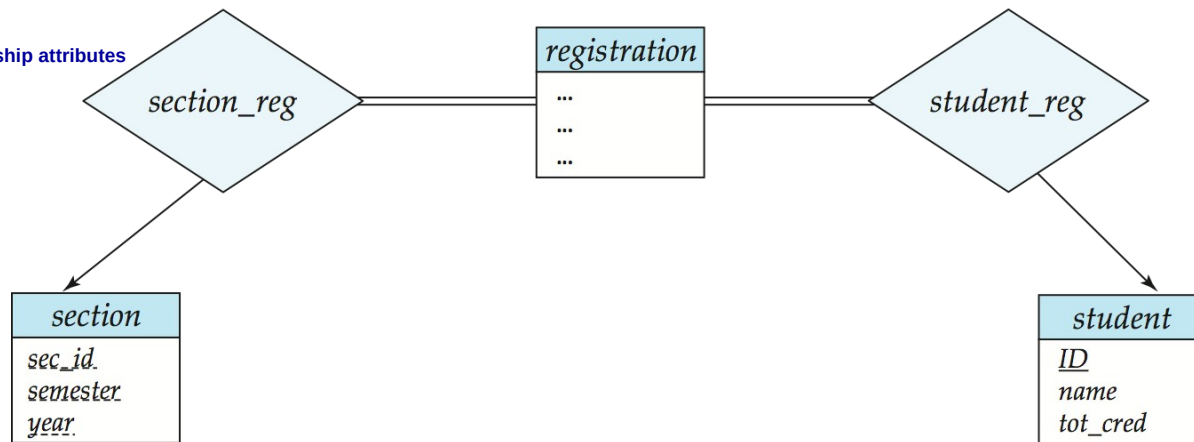


# Entities vs. Relationship sets

## ■ Use of entity sets vs. relationship sets

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

## ■ Placement of relationship attributes



For example, attribute date as attribute of advisor or as attribute of student



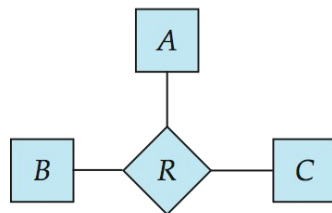
# Binary Vs. Non-Binary Relationships

- 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*

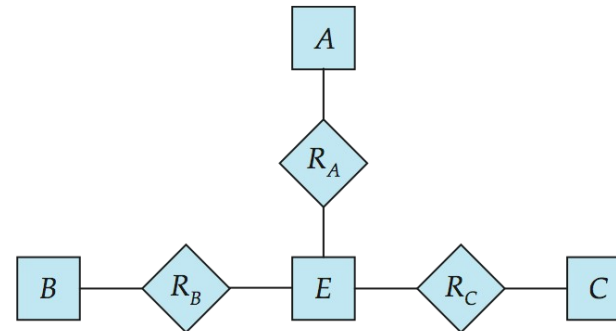


# Converting Non-Binary Relationships to Binary Form

- In general, any non-binary relationship can be represented using binary relationships by creating an artificial entity set.
  - Replace  $R$  between entity sets  $A$ ,  $B$  and  $C$  by an entity set  $E$ , and three relationship sets:
    1.  $R_A$ , relating  $E$  and  $A$
    2.  $R_B$ , relating  $E$  and  $B$
    3.  $R_C$ , relating  $E$  and  $C$
  - Create an identifying attribute for  $E$  and add any attributes of  $R$  to  $E$
  - For each relationship  $(a_i, b_i, c_i)$  in  $R$ , create
    1. a new entity  $e_i$  in the entity set  $E$
    2. add  $(e_i, a_i)$  to  $R_A$
    3. add  $(e_i, b_i)$  to  $R_B$
    4. add  $(e_i, c_i)$  to  $R_C$



(a)



(b)



# Converting Non-Binary Relationships (Cont.)

- Also need to translate constraints
  - Translating all constraints may not be possible
  - There may be instances in the translated schema that cannot correspond to any instance of  $R$ 
    - ▶ Exercise: *add constraints to the relationships  $R_A$ ,  $R_B$  and  $R_C$  to ensure that a newly created entity corresponds to exactly one entity in each of entity sets  $A$ ,  $B$  and  $C$*
  - We can avoid creating an identifying attribute by making  $E$  a weak entity set (described shortly) identified by the three relationship sets



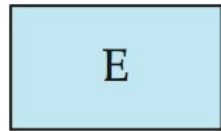


# E-R Design Decisions

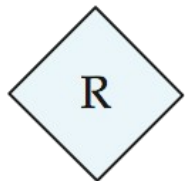
- The use of an attribute or entity set to represent an object.
- Whether a real-world concept is best expressed by an entity set or a relationship set.
- The use of a ternary relationship versus a pair of binary relationships.
- The use of a strong or weak entity set.
- The use of specialization/generalization – 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 E-R Notation



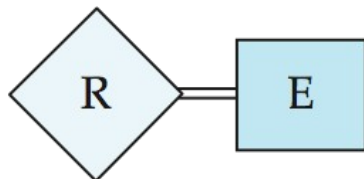
entity set



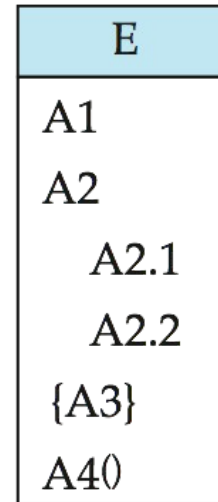
relationship set



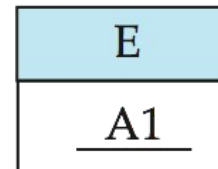
identifying  
relationship set  
for weak entity set



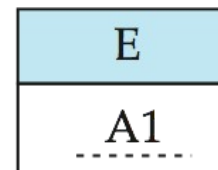
total participation  
of entity set in  
relationship



attributes:  
simple (A1),  
composite (A2) and  
multivalued (A3)  
derived (A4)



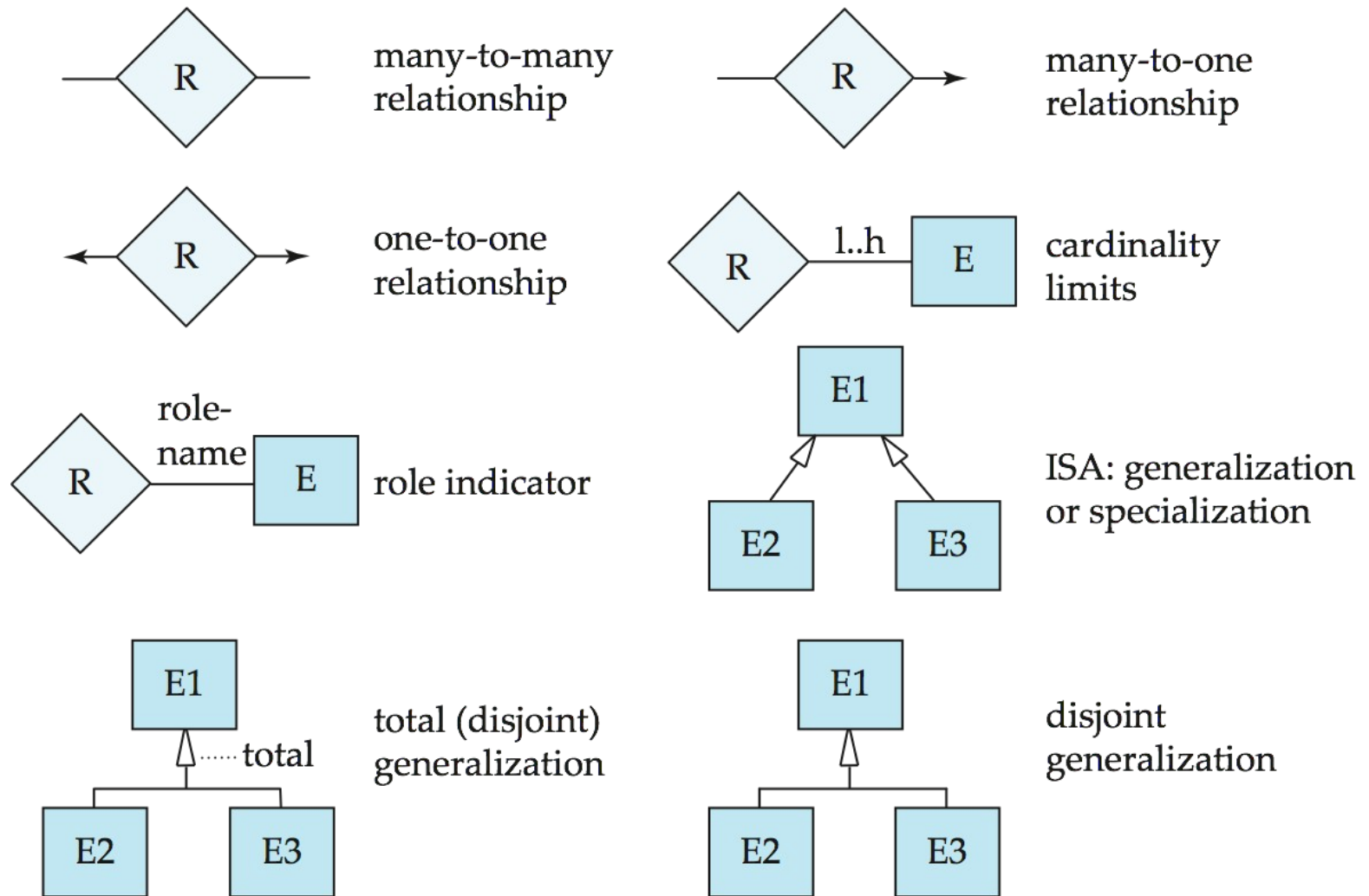
primary key



discriminating  
attribute of  
weak entity set



# Symbols Used in E-R Notation (Cont.)

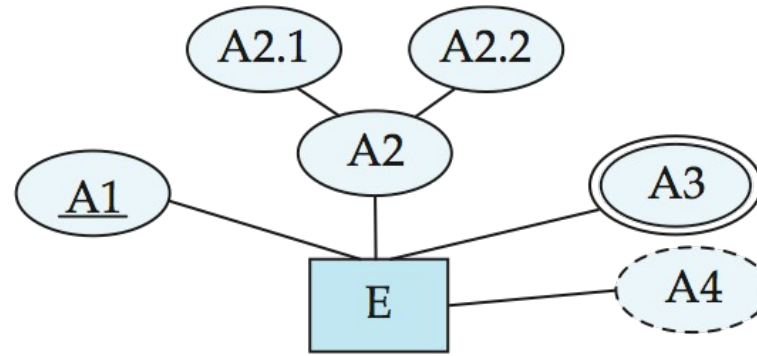




# Alternative ER Notations

## ■ Chen, IDE1FX, ...

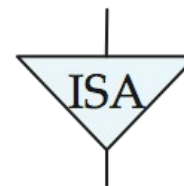
entity set E with  
simple attribute A1,  
composite attribute A2,  
multivalued attribute A3,  
derived attribute A4,  
and primary key A1



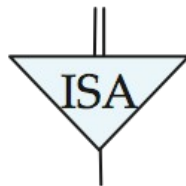
weak entity set



generalization



total  
generalization



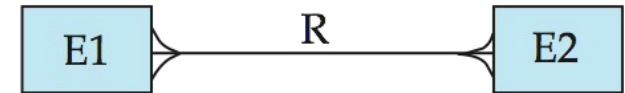
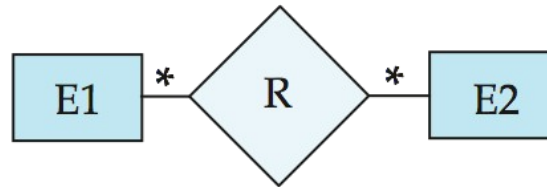


# Alternative ER Notations

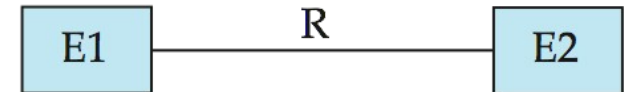
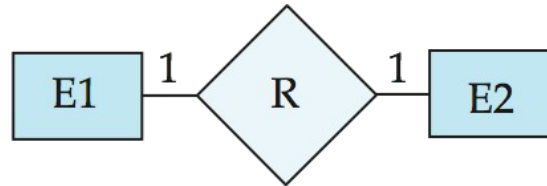
Chen

IDE1FX (Crows feet notation)

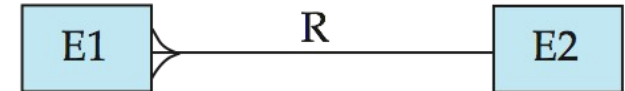
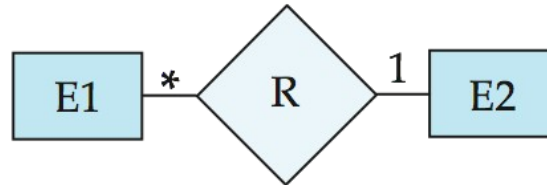
many-to-many  
relationship



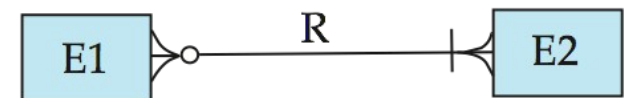
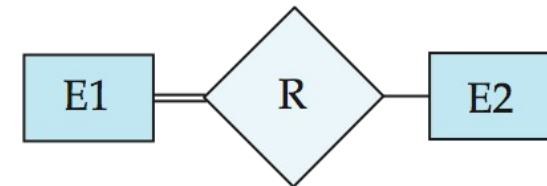
one-to-one  
relationship



many-to-one  
relationship



participation  
in R: total (E1)  
and partial (E2)





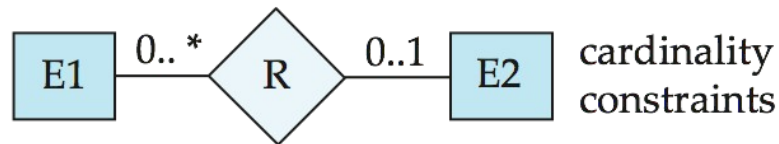
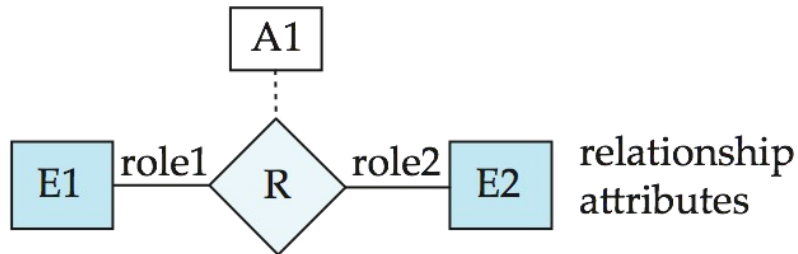
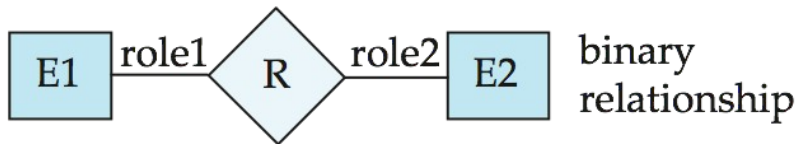
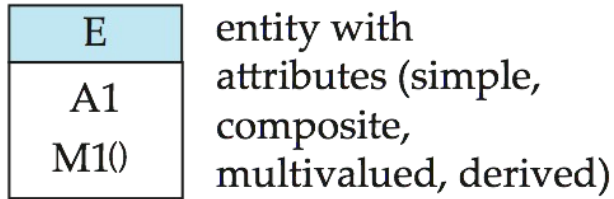
# UML

- **UML**: Unified Modeling Language
- UML has many components to graphically model different aspects of an entire software system
- UML Class Diagrams correspond to E-R Diagram, but several differences.

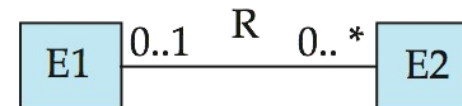
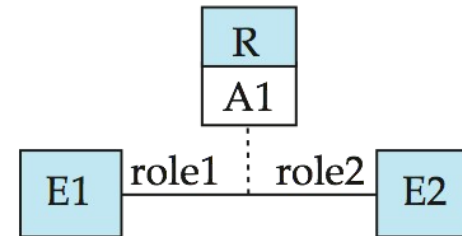
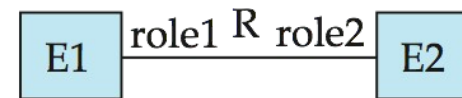
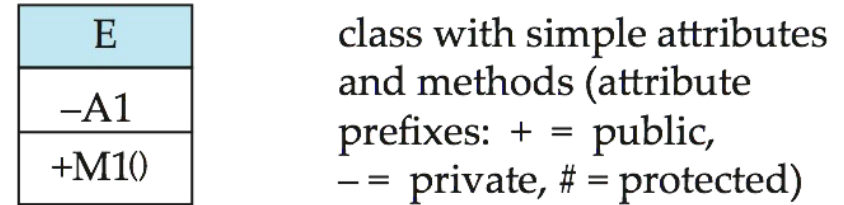


# ER vs. UML Class Diagrams

## ER Diagram Notation



## Equivalent in UML

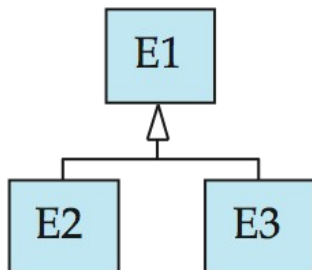
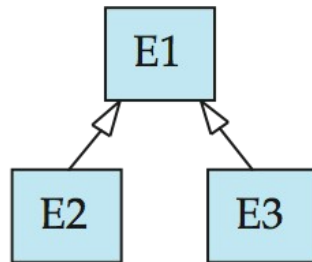
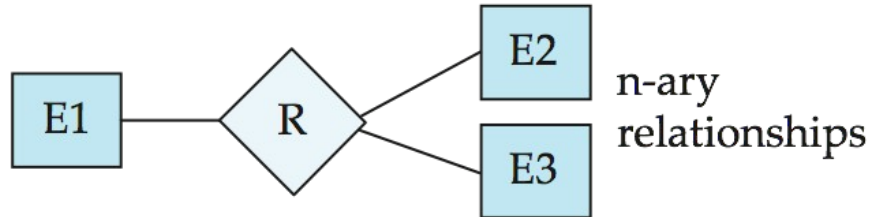


\*Note reversal of position in cardinality constraint depiction

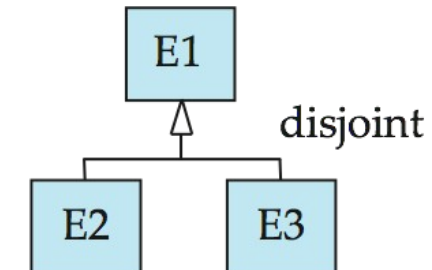
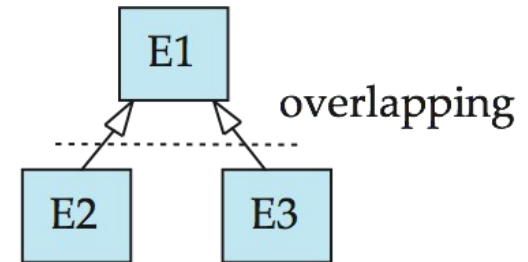
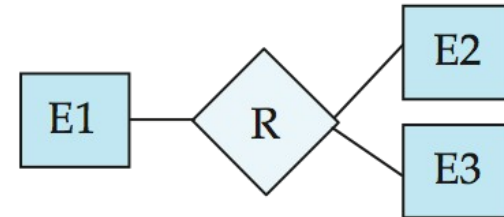


# ER vs. UML Class Diagrams

## ER Diagram Notation



## Equivalent in UML



\*Generalization can use merged or separate arrows independent of disjoint/overlapping





# UML Class Diagrams (Cont.)

- Binary relationship sets are represented in UML by just drawing a line connecting the entity sets. The relationship set name is written adjacent to the line.
- The role played by an entity set in a relationship set may also be specified by writing the role name on the line, adjacent to the entity set.
- The relationship set name may alternatively be written in a box, along with attributes of the relationship set, and the box is connected, using a dotted line, to the line depicting the relationship set.



# End of Chapter 7

**Database System Concepts, 7<sup>th</sup> Ed.**

©Silberschatz, Korth and Sudarshan

See [www.db-book.com](http://www.db-book.com) for conditions on re-use