

CS34800  
Information Systems

*Entity-Relationship Model*

Prof. Walid Aref

22 September 2017





# Main Categories of Data Models



- **Logical Models:**
  - Describe, organize, and access data in DBMS
  - Application programs refer to these models
  - Are independent from the physical data structures that store the data
  - Examples: Relational data model, Hierarchical data model, Network data model, Object-oriented data model, Object-relational data model
- **Conceptual Models:**
  - Support the representation of data independently from specific DBMS
  - Provide representations of the real-world entities, their properties and relationships.
  - Mainly used for the conceptual design of databases
  - The **Entity-Relationship** is the most well-known model in this category

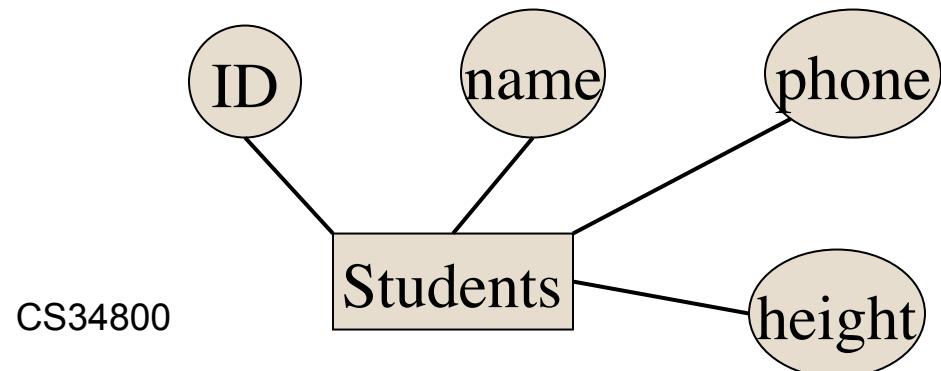


# Entity/Relationship Model



Diagrams to represent designs

- *Entity*:
  - An object, e.g., student, book, classroom
- *Entity set*:
  - A class or collection of objects, e.g., set of students
  - Set of “similar” entities/objects
- *Attribute*:
  - Property of entities in an entity set that describe an entity
  - Similar to fields in a struct.
- In diagrams:
  - Entity set → rectangle
  - Attribute → oval





# 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
- Attributes forming the key are underlined



# 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*



# Entity Sets

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

<i>instructor</i>
<u>ID</u>
<i>name</i>
<i>salary</i>

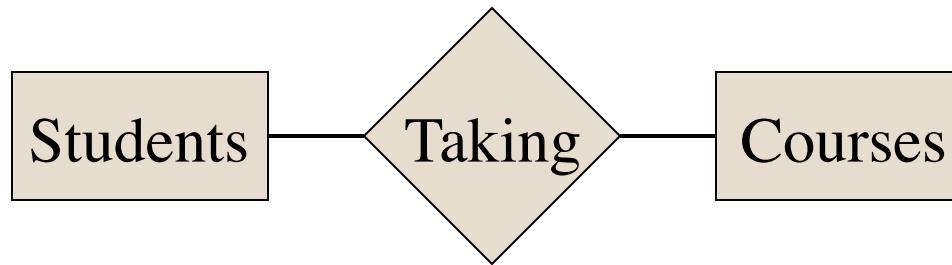
<i>student</i>
<u>ID</u>
<i>name</i>
<i>tot_cred</i>



# Relationships



- Connect two or more entity sets
- Represented by diamonds





# Relationship Set



Think of the “values” in a relationship set tuples in a table

- One column (attribute) from each of the connected entity sets
- One row (tuple) from each list of entities (one from each set) that are connected by the relationship

Students	Courses
Sally	CS348
Sally	CS355
Joe	CS348
...	





# Relationship Sets

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

Example:

44553 (Peltier)                  advisor                  22222 (Einstein)  
student entity                  relationship set                  instructor 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

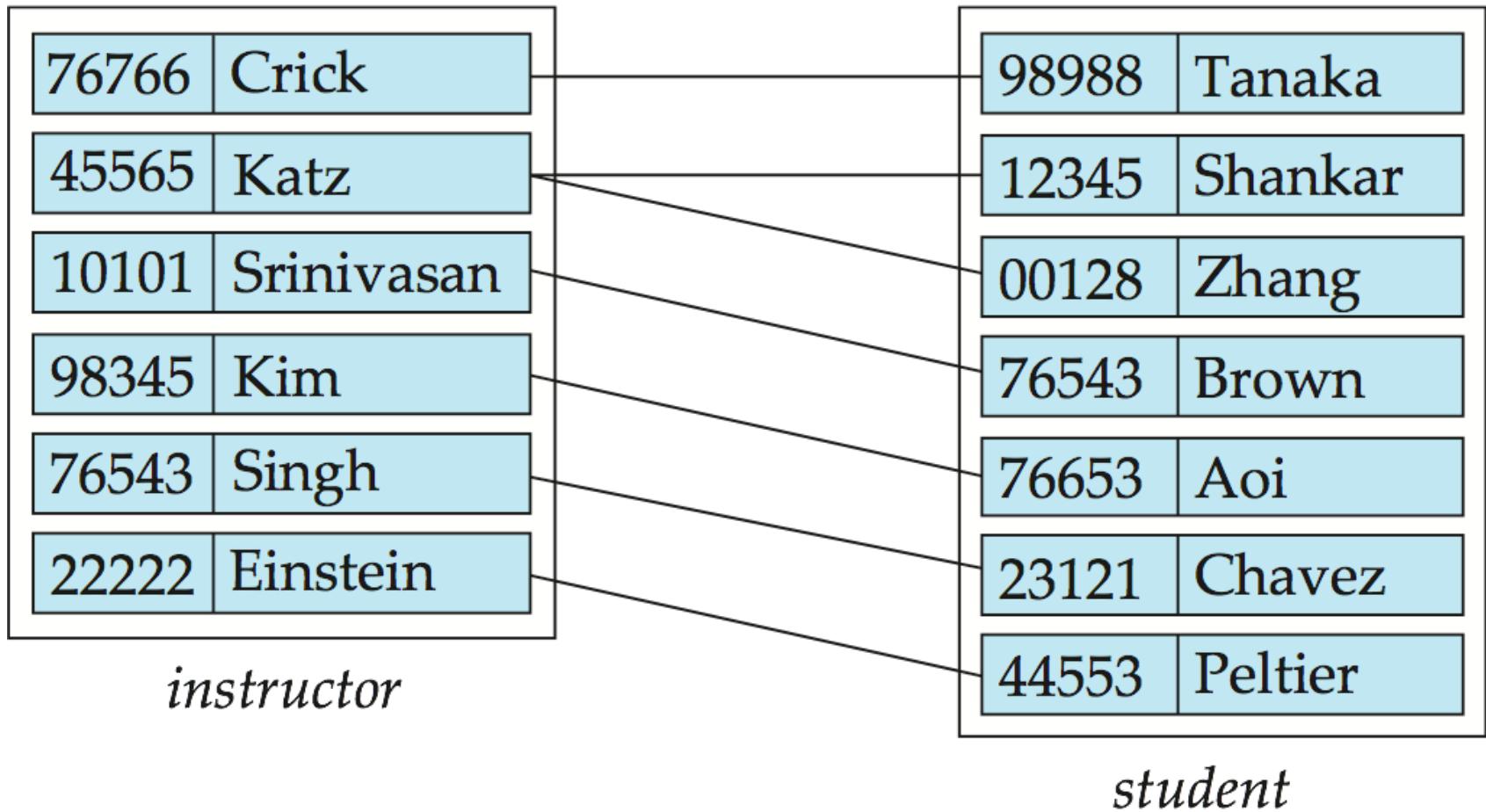
$e_i$  is key in  $E_i$

- Example:

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



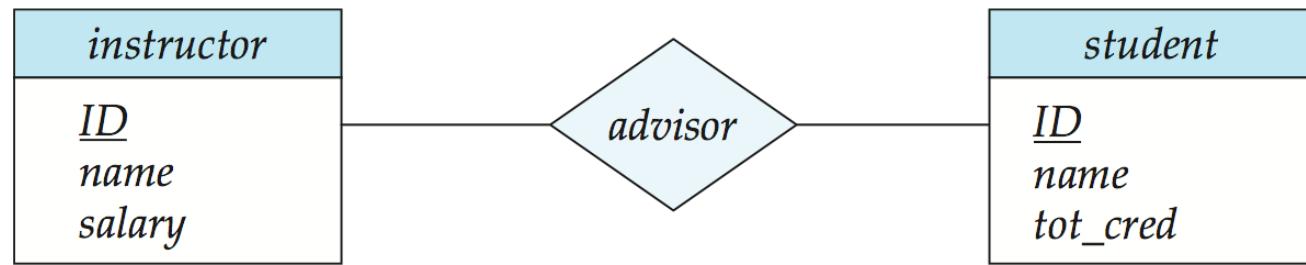
# Relationship Set Advisor





# Relationship Sets

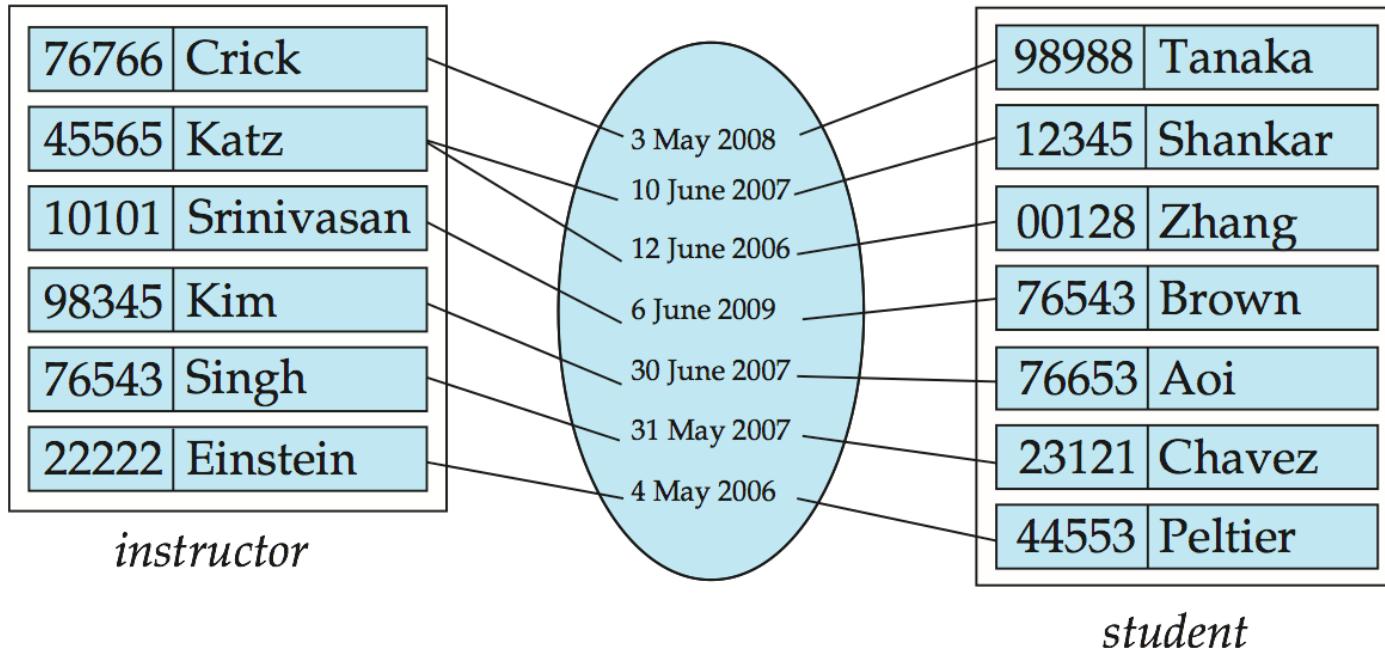
- Diamonds represent relationship sets





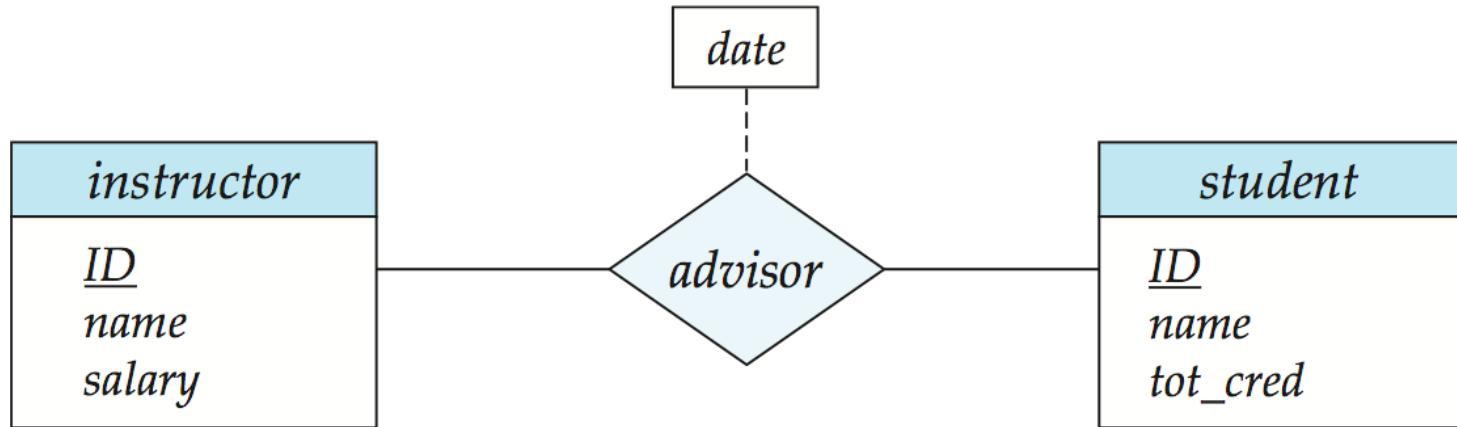
# Relationship Sets (Cont'd)

- Attributes can also be associated with a relationship set
- Example: The *advisor* relationship set between entity sets *instructor* and *student* has Attribute *date* that tracks when the student started being associated with the advisor





# Relationship Sets with Attributes





# Multiway Relationships



- Example:
  - Assignment of student to TA not captured in “course to student” relationship
  - Student taking this course with this TA
  - Ternary relationship among entity sets student, course, TA

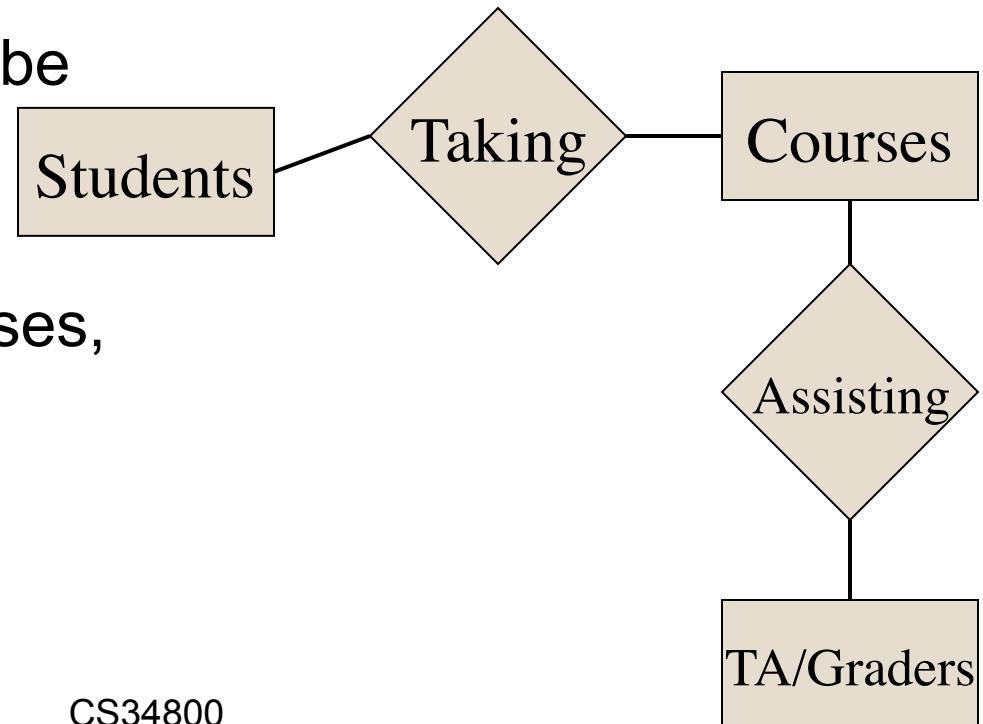


# Multiway Relationships

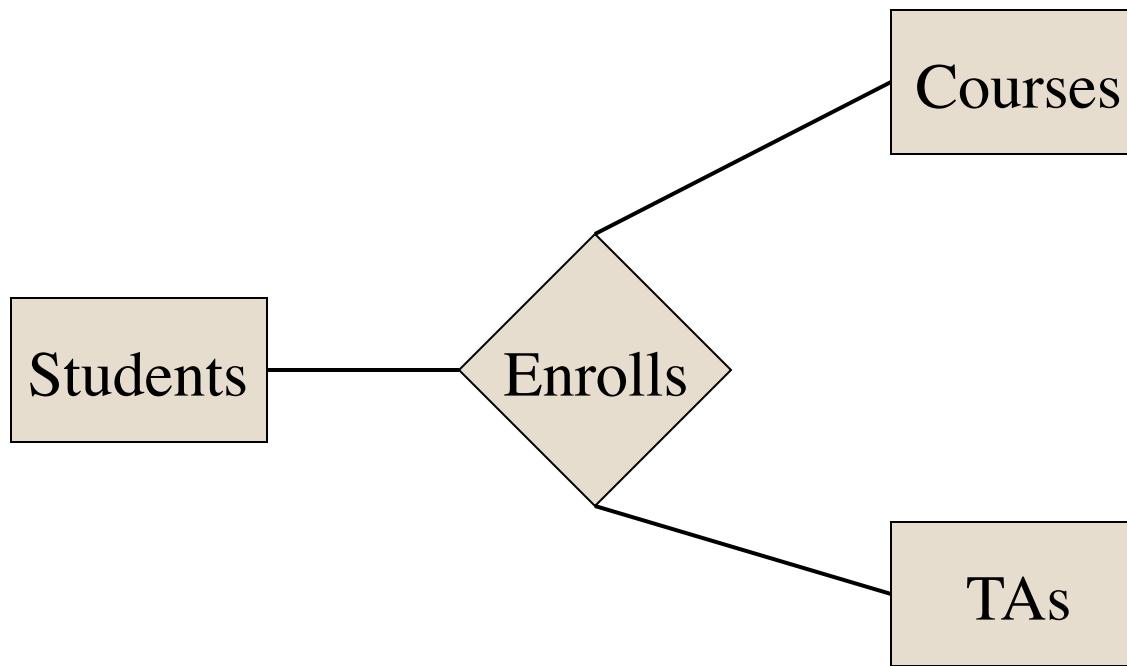


Usually, binary relationships  
(connecting two entity sets) suffice

- However, there are some cases where three or more entity sets must be connected by one relationship
- Example: relationship among students, courses, TA's (and graders)



Possibly, this E/R  
diagram is OK:

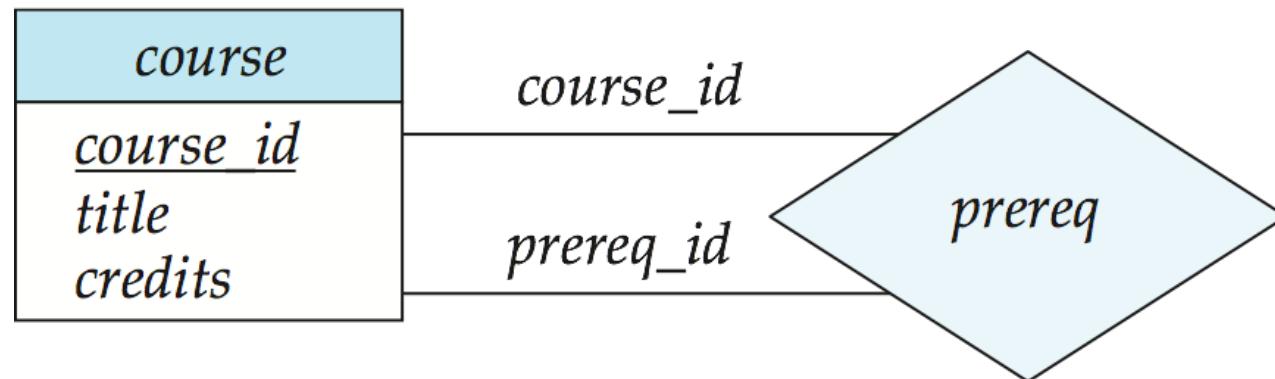


Students	Courses	TAs
Ann	CS348	Ahmed
Sue	CS348	Jaideep
Bob	CS348	Anshu
...	...	...



# Roles

- Consider the following relationship “is prerequisite of”:
  - Course 25100 is “prerequisite” of Course 34800



- 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**



# Degree of a Relationship Set

- Binary relationship
  - Involves 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*

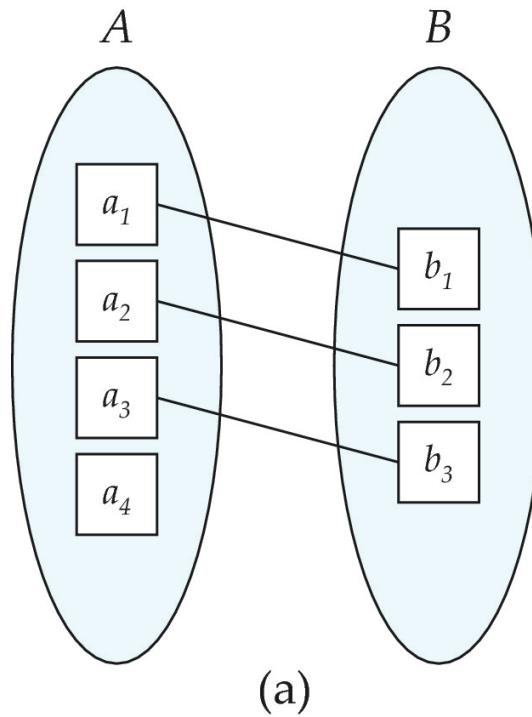


# Mapping Cardinality Constraints

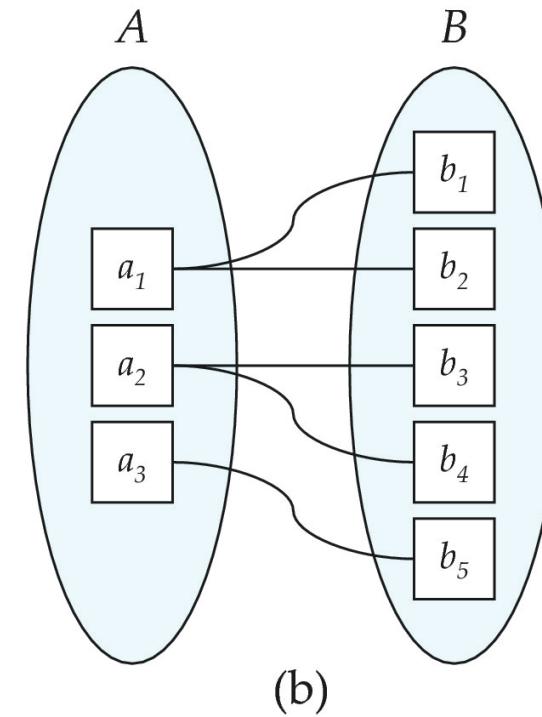
- Express the number of entities to which another entity can be associated via a relationship set
- Most useful in describing binary relationship sets
- For a binary relationship set, the mapping cardinality must be one of the following types:
  - One to one
  - One to many
  - Many to one
  - Many to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



# Mapping Cardinalities



One to one

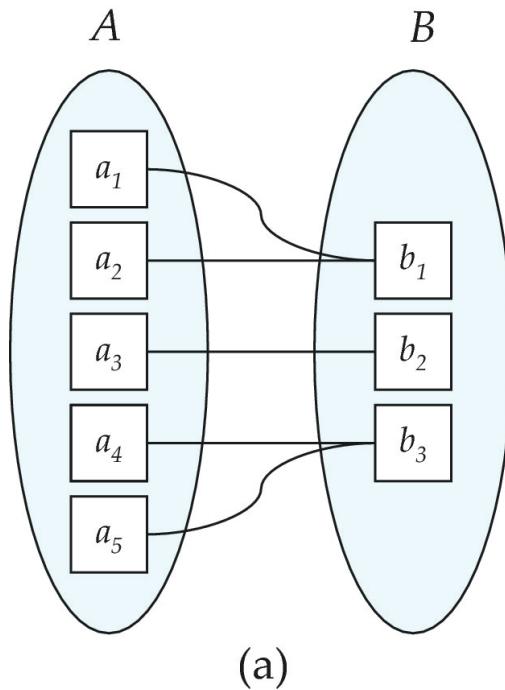


One to many

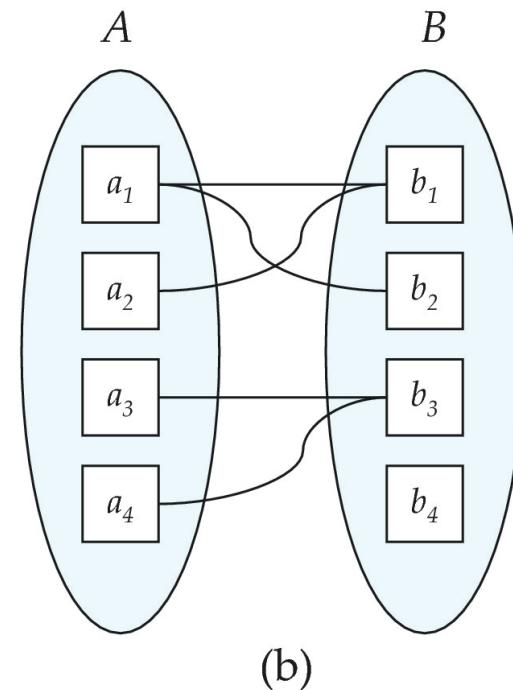
Note: Some elements in  $A$  and  $B$  may not be mapped to any elements in the other set



# Mapping Cardinalities



Many to one



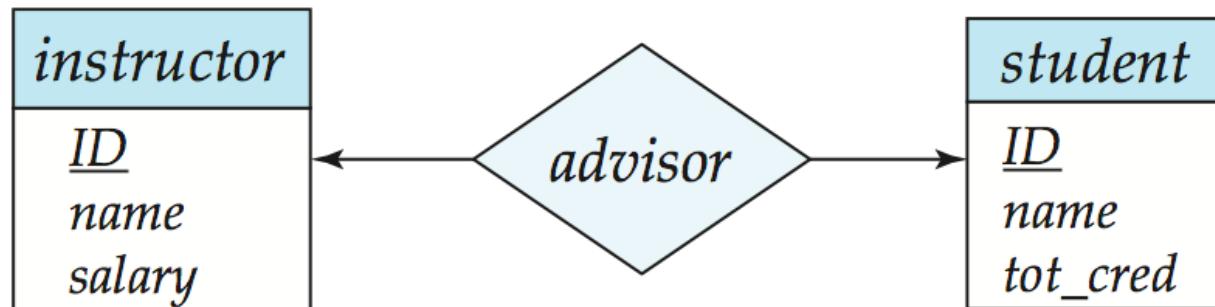
Many to many

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



# Cardinality Constraints

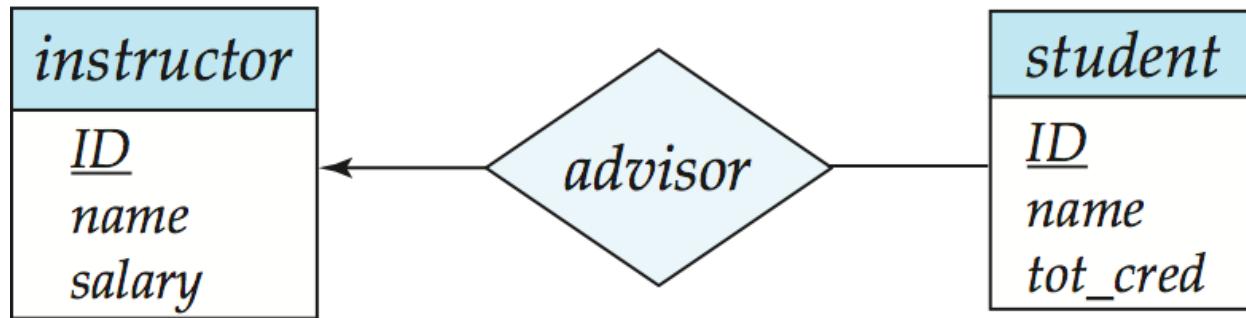
- We express cardinality constraints by drawing either a directed line ( $\rightarrow$ ), signifying “one,” or an undirected line ( $-$ ), signifying “many,” between the relationship set and the entity set.
- One-to-one relationship between an *instructor* and a *student* :
  - A student is associated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

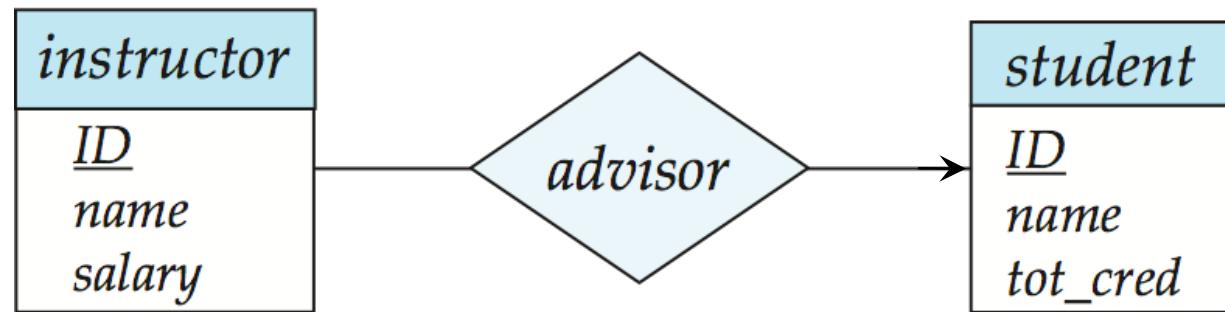
- One-to-many relationship between an *instructor* and a *student*
  - An instructor is associated with several (including 0) students via *advisor*
  - A student is associated with at most one instructor via advisor





# Many-to-One Relationships

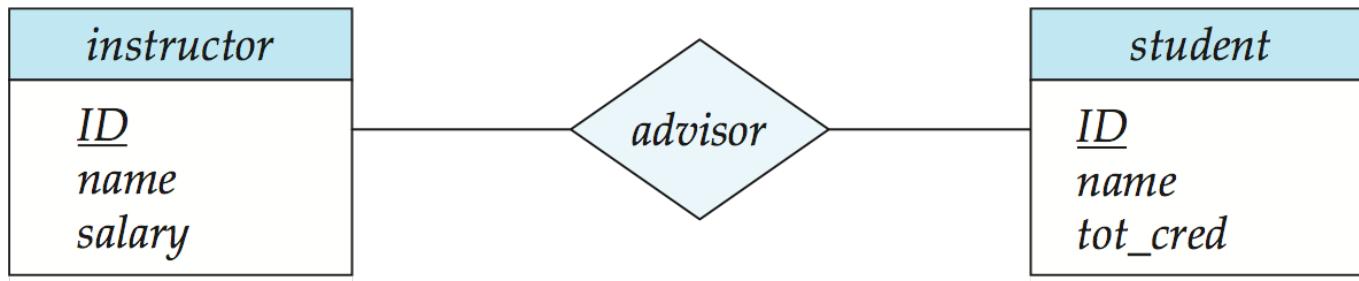
- In a many-to-one relationship between an *instructor* and a *student*,
  - An *instructor* is associated with at most one *student* via *advisor*
  - A *student* is associated with several (including 0) *instructors* via *advisor*





# 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*

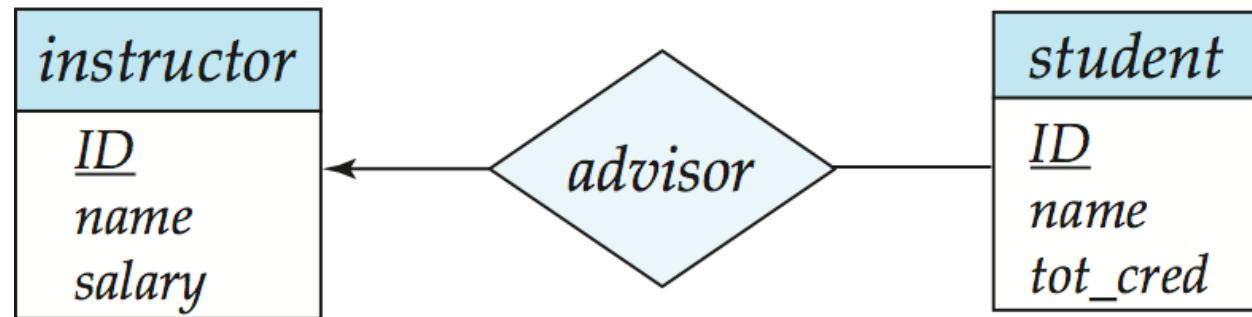




# ER Diagram: Review

## ■ Conceptual Database Design Model

- Entity Sets
- Relationships



## ■ In this diagram:

- advisor is a relationship
- A student can have at most one instructor as advisor
- Each *instructor* can advise 0 or more students
- ID* is key for *instructor*



# Weak Entity Sets

- Consider a ***section*** entity:
  - Is uniquely identified by a *course\_id*, *semester*, *year*, and *sec\_id*.
  - ***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**.



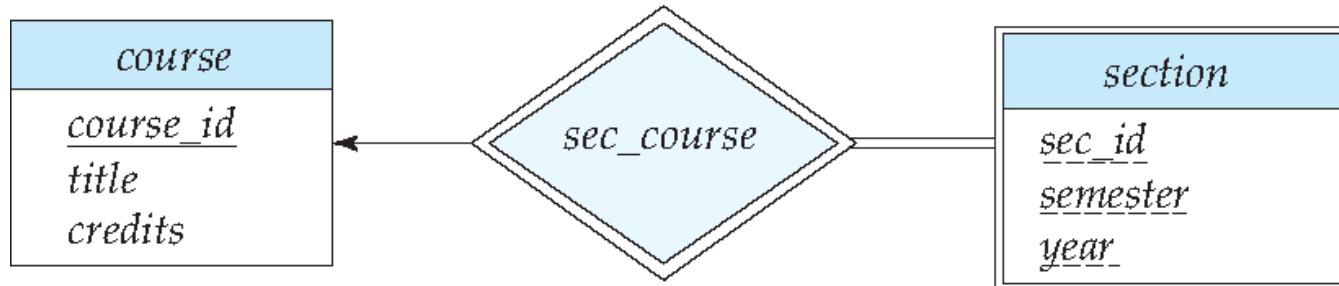
# Weak Entity Sets (Cont.)

- 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*.



# 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*)





# Don't Overuse Weak E.S.

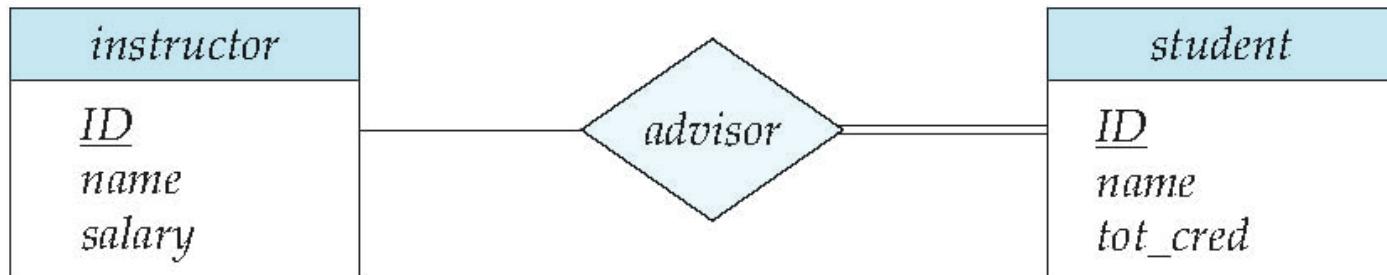


- There is a tendency to feel that no E.S. has its entities uniquely determined without following some relationships.
- However, in practice, we almost always create unique ID's to compensate: student ID numbers, VIN's, etc.
- The only times weak E.S.'s seem necessary are when:
  - a) We can't easily create such ID's; e.g., no one is going to accept a "species ID" as part of the standard nomenclature (species is a weak E.S. supported by membership in a genus).
  - b) There is no global authority to create them, e.g., assignments in a class.



# 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



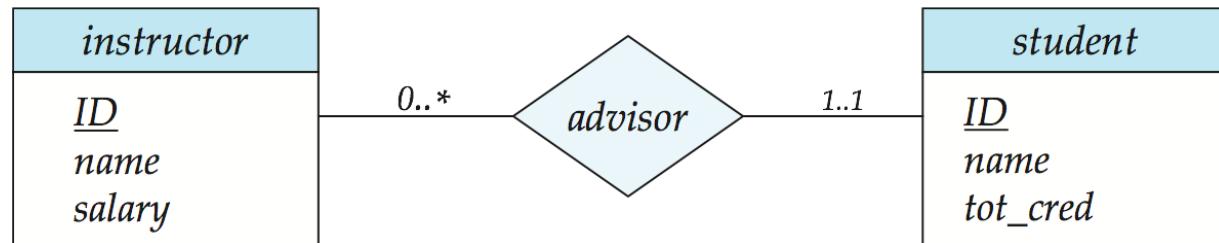
participation of *student* in *advisor* relation is total

- ▶ 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



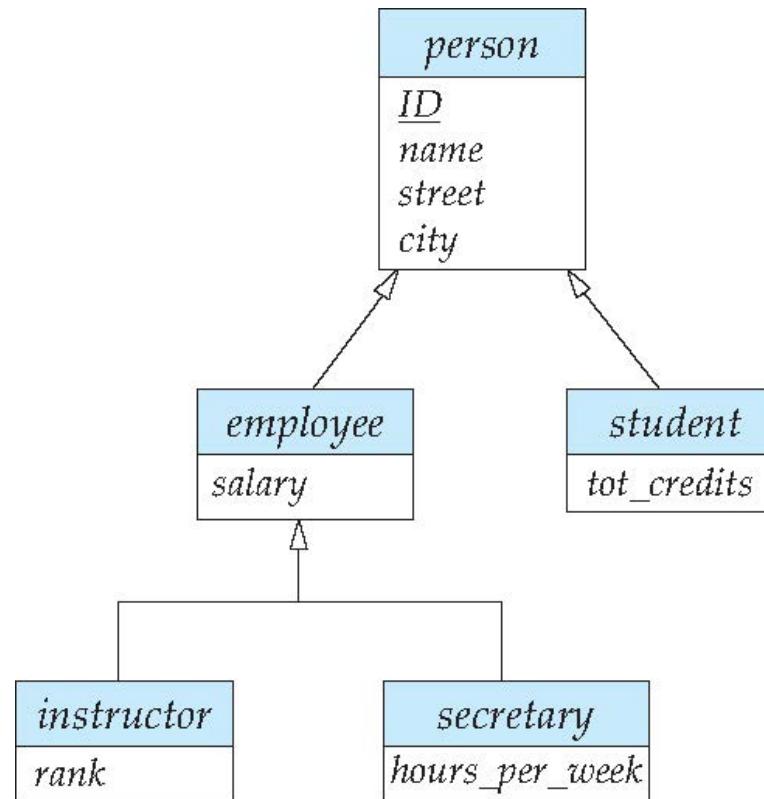
# Specialization

- 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

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



# 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



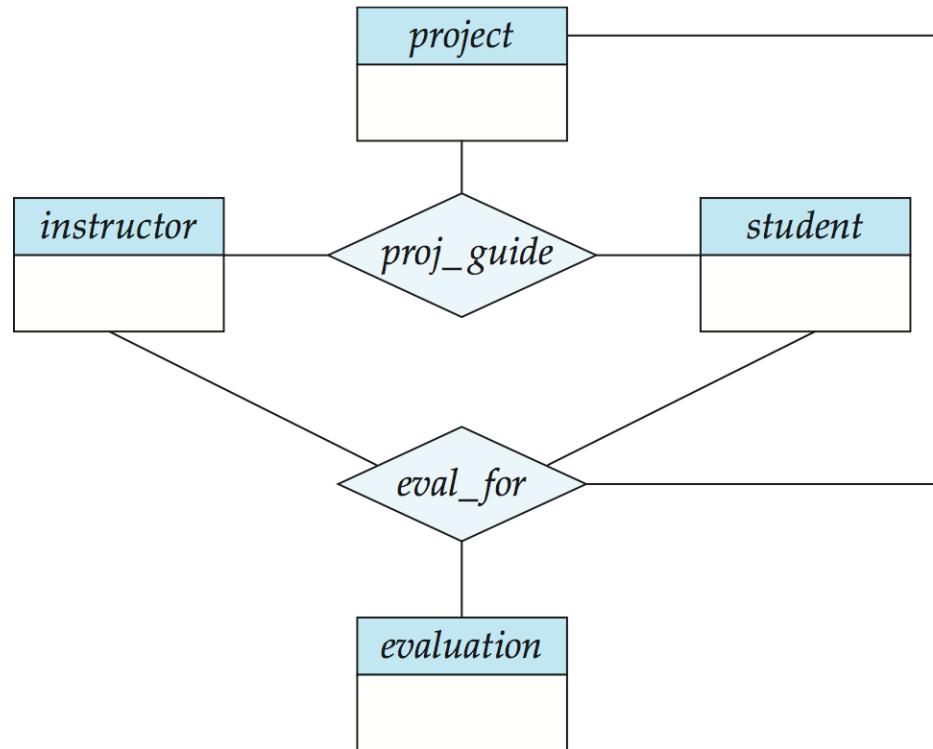
# Design Constraints on a Specialization/Generalization

- **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





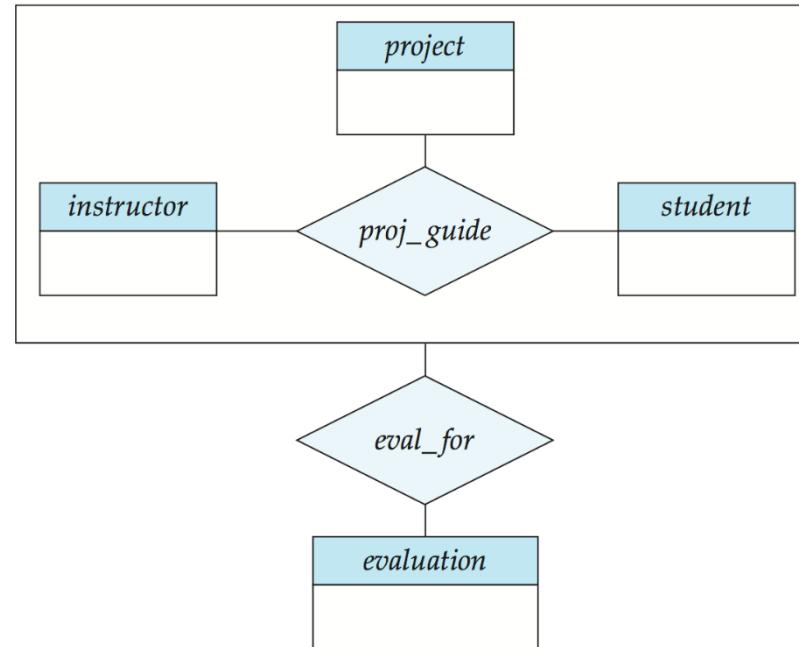
# Aggregation (Cont.)

- 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:  
$$\text{eval\_for} (s\_ID, project\_id, i\_ID, evaluation\_id)$$
  - The schema *proj\_guide* is redundant.



# 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



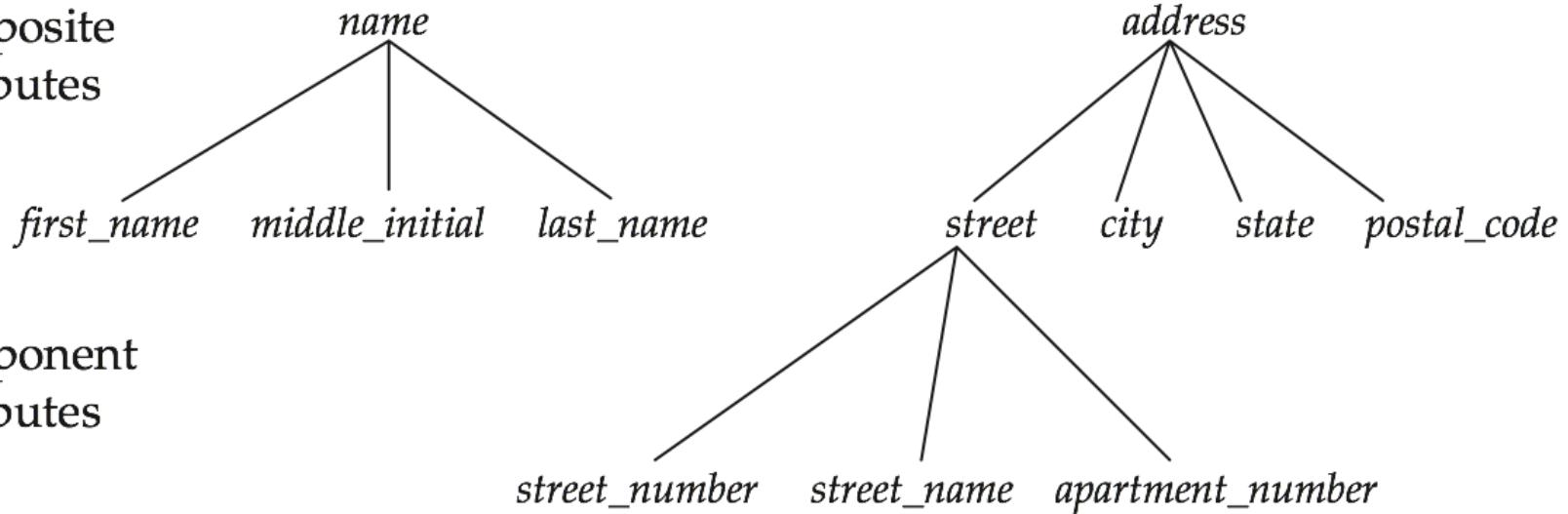
# Notation to Express Entity with Complex Attributes

<i>instructor</i>
<i>ID</i>
<i>name</i>
<i>first_name</i>
<i>middle_initial</i>
<i>last_name</i>
<i>address</i>
<i>street</i>
<i>street_number</i>
<i>street_name</i>
<i>apt_number</i>
<i>city</i>
<i>state</i>
<i>zip</i>
{ <i>phone_number</i> }
<i>date_of_birth</i>
<i>age</i> ( )



# Composite Attributes

composite  
attributes



component  
attributes

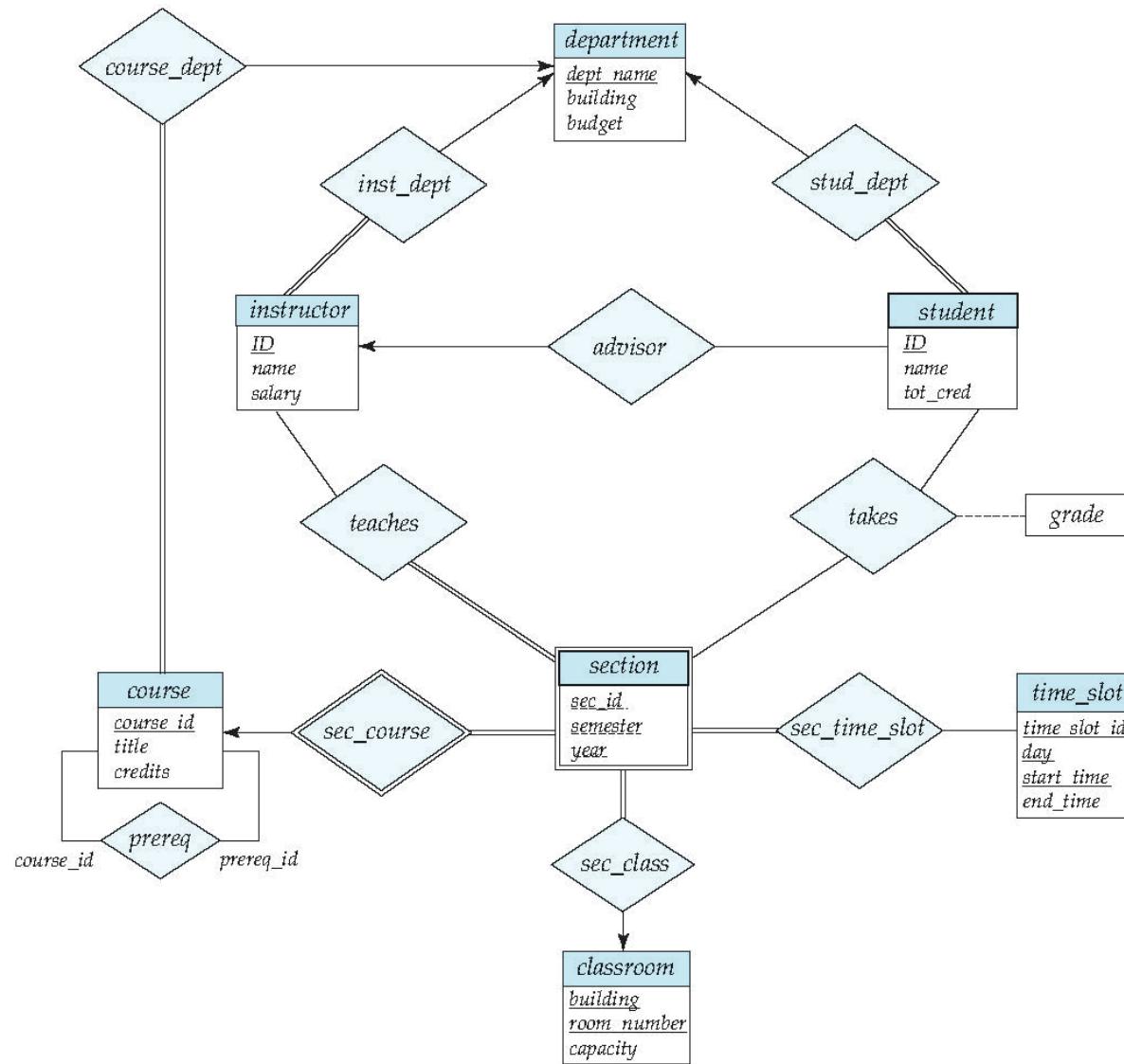


# Redundant Attributes

- 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



# E-R Diagram for a University Enterprise





# Reduction to Relation Schemas



# Relational Database Design



- Goals
  - Capture all the data
    - Nothing lost
  - Represent only the data
    - Prevent data not matching the real world
- How?
  - Division of data into relations
  - Key constraints



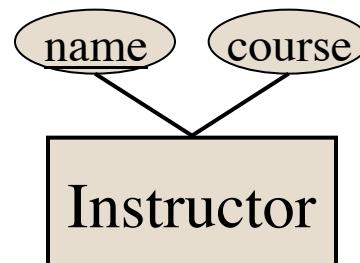
# Relational Design



Simplest approach (not always best): convert each E.S. to a relation and each relationship to a relation.

## Entity Set → Relation

E.S. attributes become relational attributes.



Becomes:

Instructor (name, course)



# Keys in Relations



An attribute or set of attributes  $K$  is a *key* for a relation  $R$  if we expect that in no instance of  $R$  will two different tuples agree on all the attributes of  $K$ .

- Indicate a key by underlining the key attributes
- Example: If name is a key for Instructor:  
Instructor (name, course)



# In Database Design, a Key:



- A. Is needed in order to access an entity
- B. Is the only way to select on an attribute
- C. Uniquely identifies each entity in an entity set
- D. Consists of a single attribute
- E. Is used to authenticate to the database



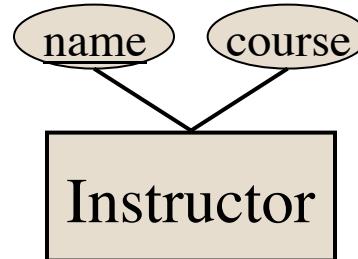
# Types of keys:



- Candidate key
  - Any valid key (guaranteed to be unique)
- Primary key
  - A candidate key chosen as the one to use
- Superkey
  - Key that can have attributes removed and still be a key



# Relational Design



Becomes:

Instructor(name, course)

SQL:

```
CREATE TABLE Instructor (
    name varchar2(40), PRIMARY KEY,
    course varchar2(9)
);
```



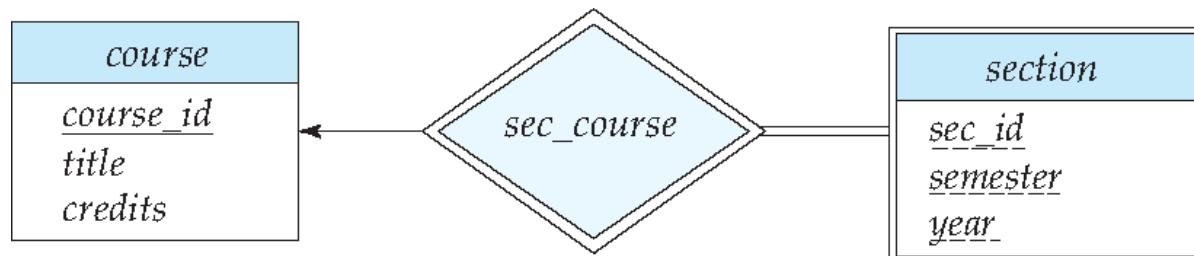
# Representing Entity Sets

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

*student( ID, name, tot\_cred )*

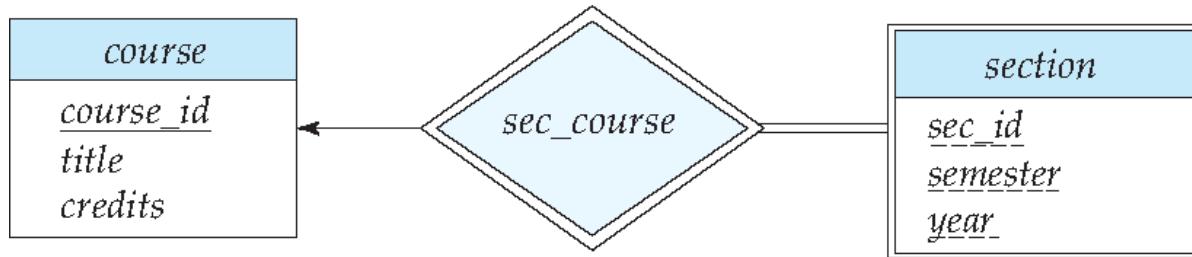
- A weak entity set becomes a table that includes a column for the primary key of the identifying strong entity set

*section ( course\_id, sec\_id, sem, year )*





# Weak Entity Sets in SQL



*section ( course\_id, sec\_id, sem, year )*

```
CREATE TABLE section (
    course_id varchar2(9),
    sec_id number(5),
    semester char(1),
    year number(4),
    PRIMARY KEY (course_id, sec_id, semester, year),
    FOREIGN KEY (course_id) REFERENCES course(course_id)
);
```



# E/R Relationships → Relations



Relation has attribute for *key* attributes of each E.S. that participates in the relationship

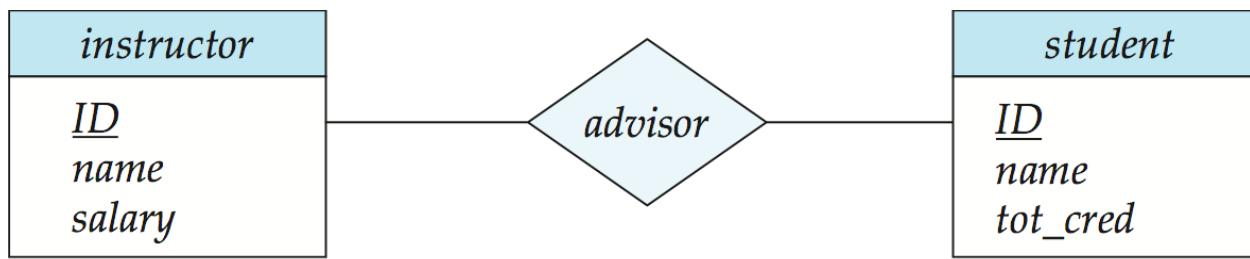
- Add any attributes that belong to the relationship itself
- Renaming attributes OK
  - Essential if multiple roles for an E.S.



# Representing Relationship Sets

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

*advisor* = (s\_id, i\_id)



```
CREATE TABLE advisor(  
    S_ID REFERENCES student(ID),  
    I_ID REFERENCES instructor(ID),  
    PRIMARY KEY (S_ID, I_ID),  
);FOREIGN KEY S_ID REFERENCES student(ID),  
    FOREIGN KEY I_ID REFERENCES instructor(ID)  
);
```

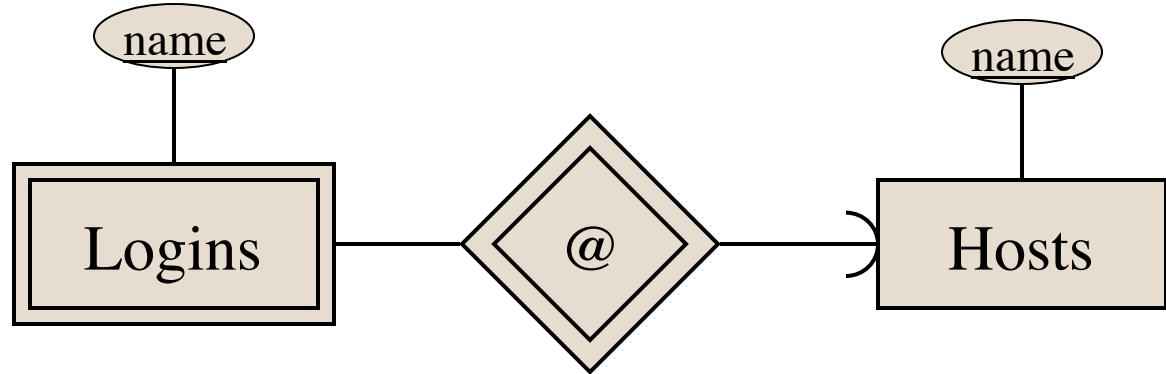


# Weak Entity Sets, Relationships → Relations



- Relation for a weak E.S. must include its full key (*i.e.*, attributes of related entity sets) as well as its own attributes.
- A supporting (double-diamond) relationship yields a relation that is actually redundant and should be deleted from the database schema.

# Example



Hosts (hostName)

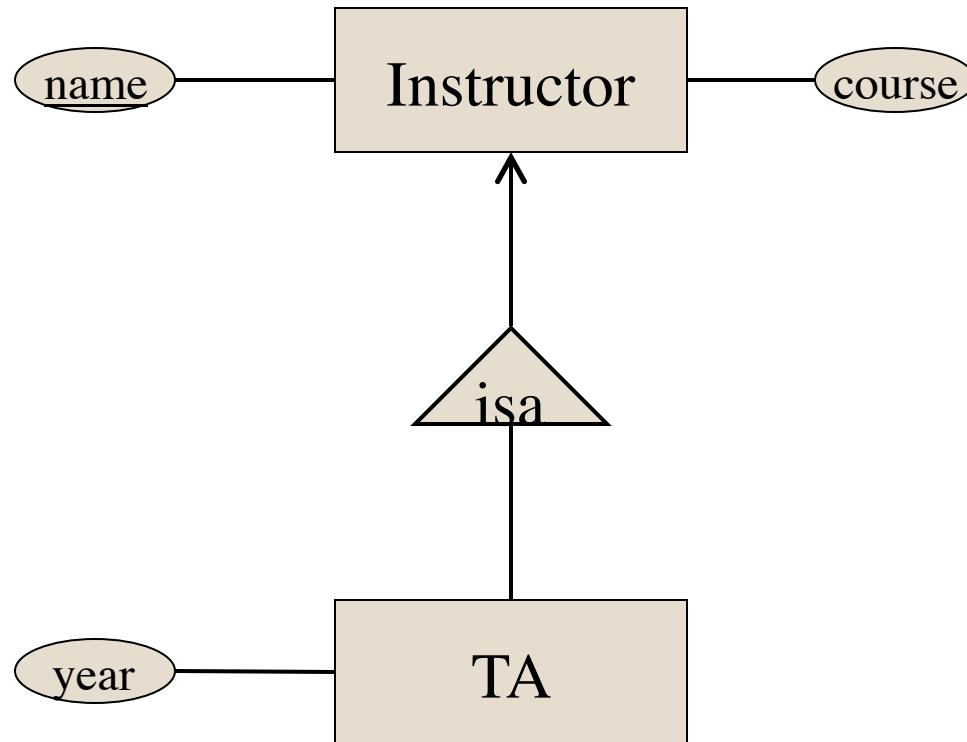
Logins (loginName, hostName)

At (loginName, hostName, hostName2)

- In At, hostName and hostName2 must be the same host, so delete one of them.
- Then, Logins and At become the same relation; delete one of them.



# Generalization / Specialization





# Representing Generalization



## A. OO style

Name	course
Clifton	CS34800

Name	Course	year
Pradhan	CS34800	4

## B. ER style

Name	course
Clifton	CS34800
Pradhan	CS34800

Name	year
Pradhan	4

## C. Using nulls

Name	Course	year
Clifton	CS34800	⊥
Pradhan	CS34800	4



# Subclasses → Relations



Three approaches:

1. Object-oriented: each entity is in one class. Create a relation for each class, with all the attributes for that class.
  - Don't forget inherited attributes.
2. E/R style: an entity is in a network of classes related by *isa*. Create one relation for each E.S.
  - An entity is represented in the relation for each subclass to which it belongs.
  - Relation has only the attributes attached to that E.S. + key.
3. Use nulls. Create one relation for the root class or root E.S., with all attributes found anywhere in its network of subclasses.
  - Put NULL in attributes not relevant to a given entity.



# Representation of Entity Sets with Composite Attributes

<i>instructor</i>
<i>ID</i>
<i>name</i>
<i>first_name</i>
<i>middle_initial</i>
<i>last_name</i>
<i>address</i>
<i>street</i>
<i>street_number</i>
<i>street_name</i>
<i>apt_number</i>
<i>city</i>
<i>state</i>
<i>zip</i>
{ <i>phone_number</i> }
<i>date_of_birth</i>
<i>age ()</i>

- 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)*



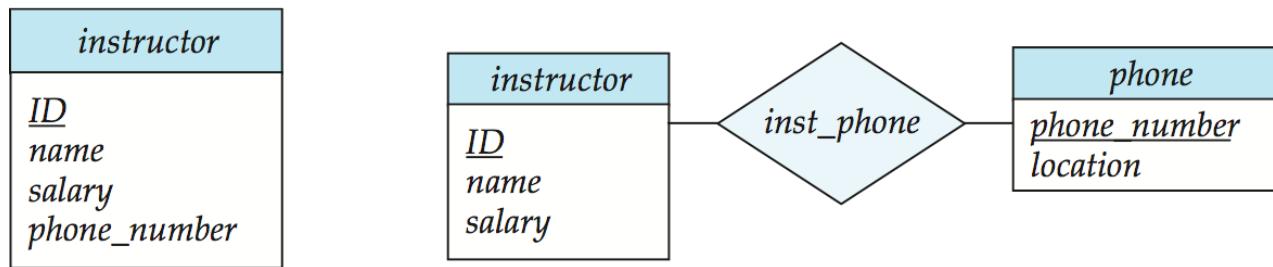
# Representation of Entity Sets with Multivalued Attributes

- 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)



# Entities vs. Attributes

- Use of entity sets vs. attributes



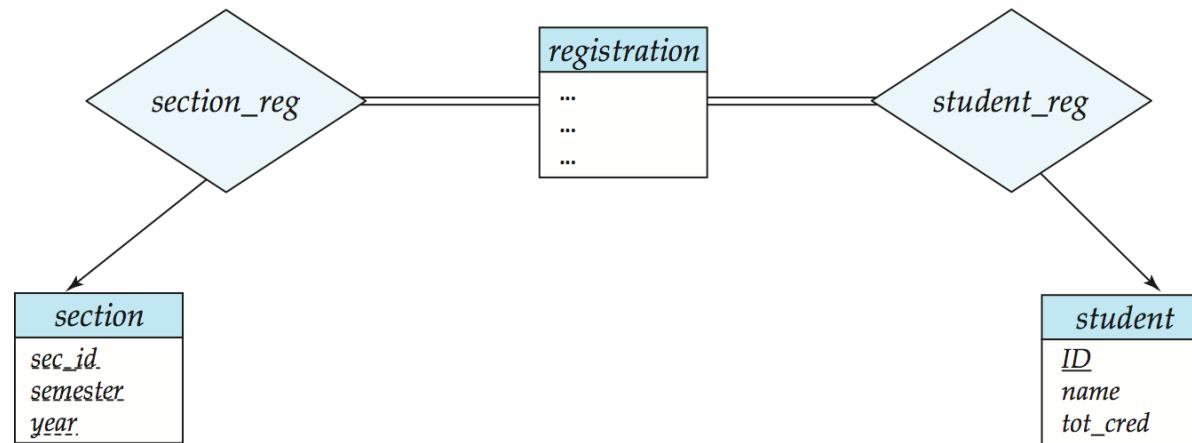
- Use of phone as an entity allows extra information about phone numbers (plus multiple phone numbers)



# 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



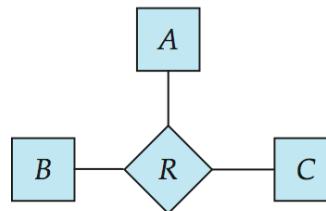
# Binary Vs. Non-Binary Relationships

- 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*

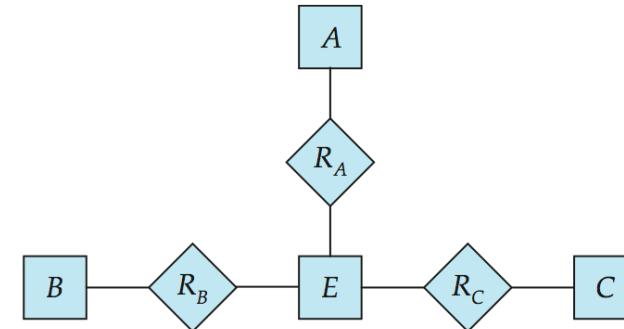


# Converting Non-Binary Relationships to Binary Form

- 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:
    - $R_A$ , relating  $E$  and A
    - $R_B$ , relating  $E$  and B
    - $R_C$ , 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
    - a new entity  $e_i$  in the entity set  $E$
    - add  $(e_i, a_i)$  to  $R_A$
    - add  $(e_i, b_i)$  to  $R_B$
    - 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

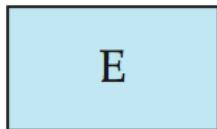


# E-R Design Decisions

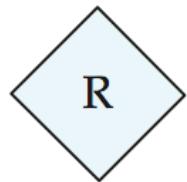
- The use of an attribute or entity set to represent an object.
- Whether a real-world concept is best expressed by an entity set or a relationship set.
- The use of a ternary relationship versus a pair of binary relationships.
- The use of a strong or weak entity set.
- The use of specialization/generalization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



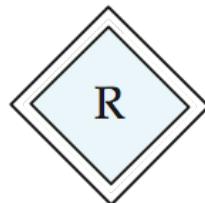
# Summary of Symbols Used in E-R Notation



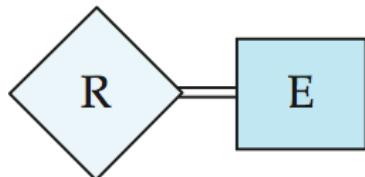
entity set



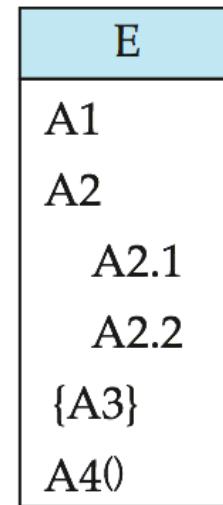
relationship set



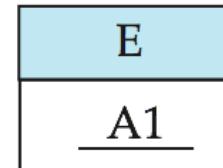
identifying  
relationship set  
for weak entity set



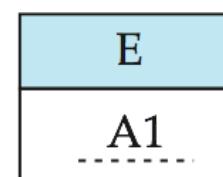
total participation  
of entity set in  
relationship



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



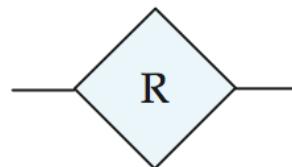
primary key



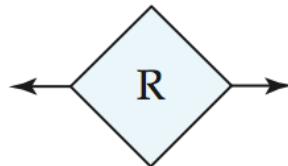
discriminating  
attribute of  
weak entity set



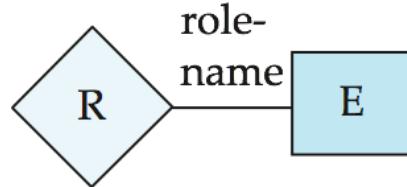
# Symbols Used in E-R Notation (Cont.)



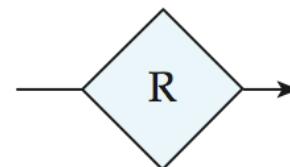
many-to-many  
relationship



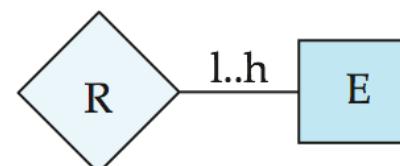
one-to-one  
relationship



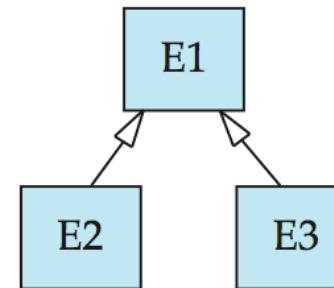
role indicator



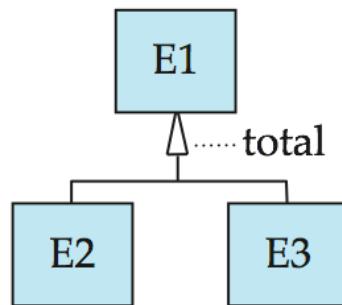
many-to-one  
relationship



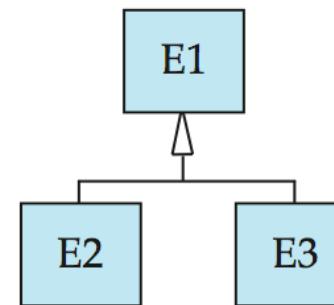
cardinality  
limits



ISA: generalization  
or specialization



total (disjoint)  
generalization



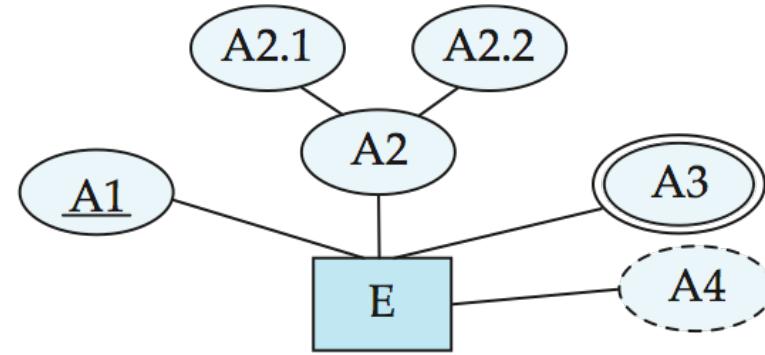
disjoint  
generalization



# Alternative ER Notations

- Chen, IDE1FX, ...

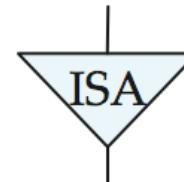
entity set E with  
simple attribute A1,  
composite attribute A2,  
multivalued attribute A3,  
derived attribute A4,  
and primary key A1



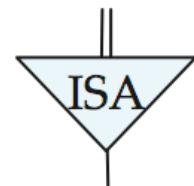
weak entity set



generalization



total  
generalization

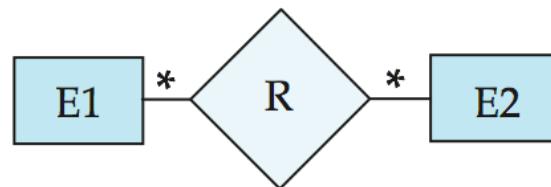




# Alternative ER Notations

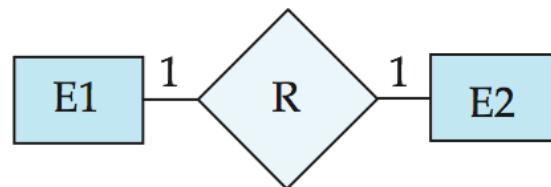
Chen

many-to-many  
relationship

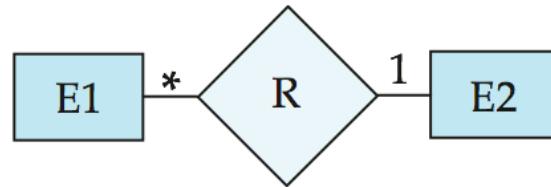


IDE1FX (Crows feet notation)

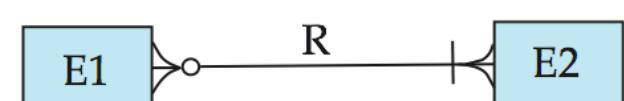
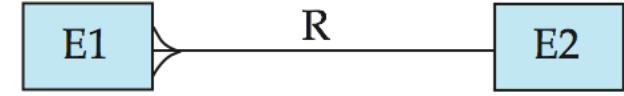
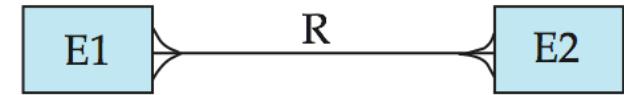
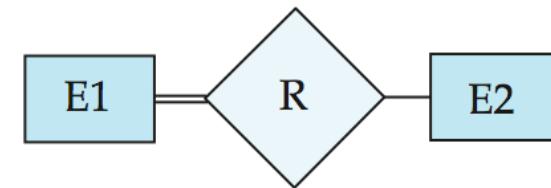
one-to-one  
relationship



many-to-one  
relationship



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





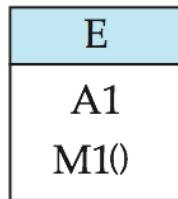
# UML

- **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.

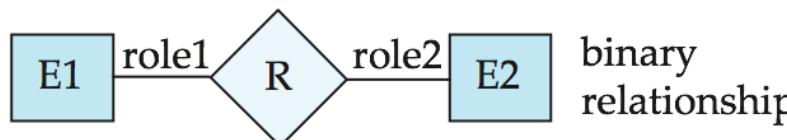


# ER vs. UML Class Diagrams

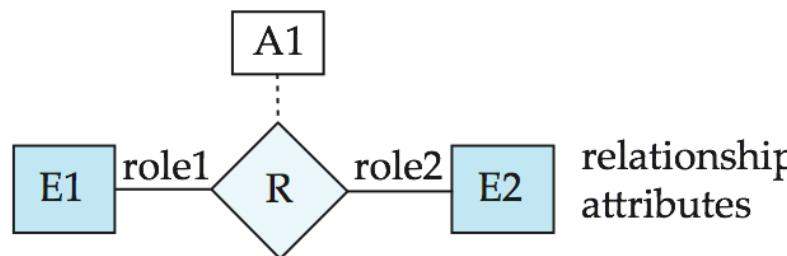
## ER Diagram Notation



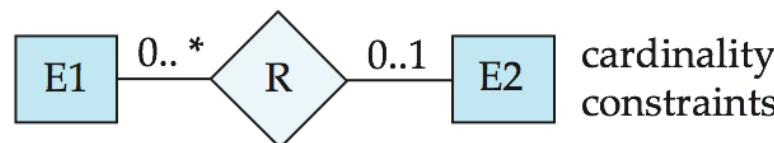
entity with attributes (simple, composite, multivalued, derived)



binary relationship

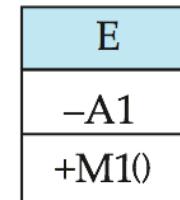


relationship attributes

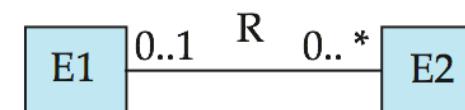
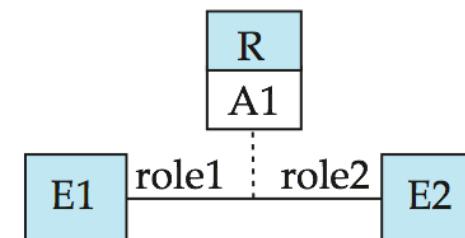
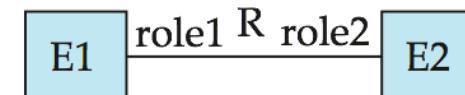


cardinality constraints

## Equivalent in UML



class with simple attributes and methods (attribute prefixes: + = public, - = private, # = protected)

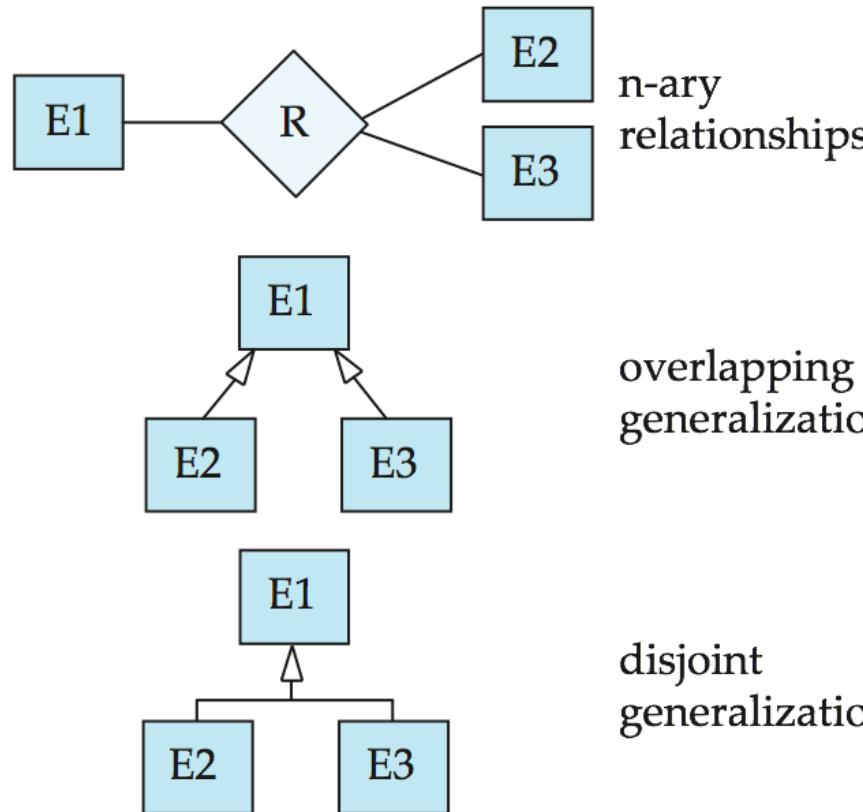


\*Note reversal of position in cardinality constraint depiction

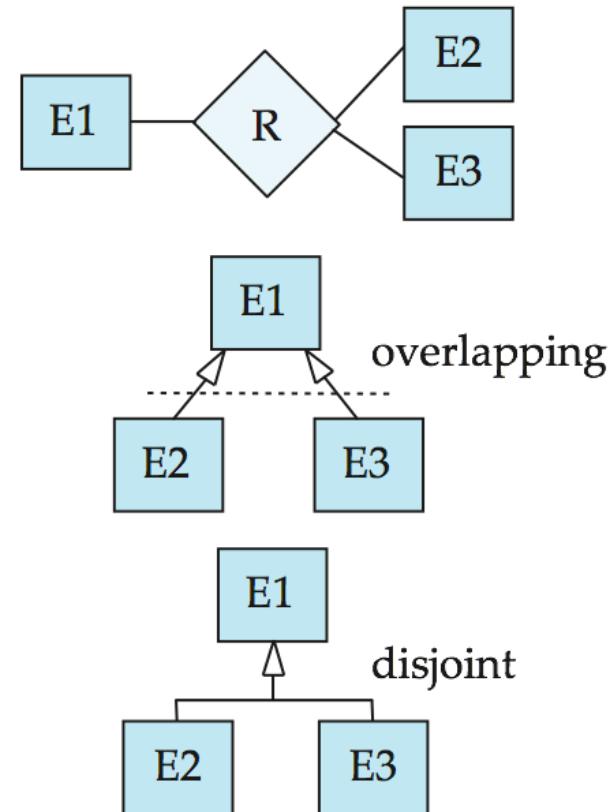


# ER vs. UML Class Diagrams

## 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.