



Chapter 6: Database Design Using the E-R Model

Database System Concepts, 7th Ed.

©Silberschatz, Korth and Sudarshan

See www.db-book.com for conditions on re-use



Design Phases

- **Initial phase** -- characterize fully the data needs of the prospective database users.
- **Second phase** -- choosing a data model
 - Applying the concepts of the chosen data model
 - Translating these requirements into a conceptual schema of the database.
 - A fully developed conceptual schema indicates the functional requirements of the enterprise.
 - Describe the kinds of operations (or transactions) that will be performed on the data.



Design Phases (Cont.)

- **Final Phase** -- Moving from an abstract data model to the implementation of the database
 - 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 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
 - **Incompleteness**: a bad design may make certain aspects of the enterprise **difficult or impossible to model**.
- Avoiding bad designs is not enough. There may be a large number of good designs from which we must choose.



Design Approaches

- Entity Relationship Model (covered in this chapter) 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 (Chapter 7)
 - Formalize what designs are bad, and test for them



Outline of the ER Model



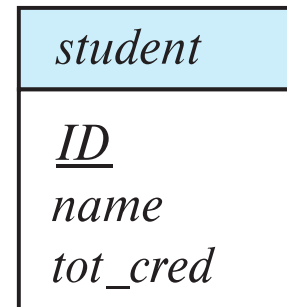
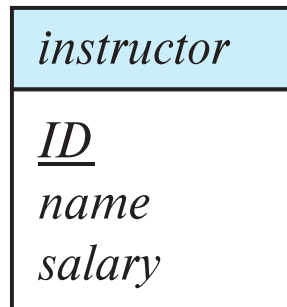
Entity Sets

- 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





Relationship Sets

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

Example:

44553 (Peltier)
student entity

advisor
relationship set

22222 (Einstein)
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

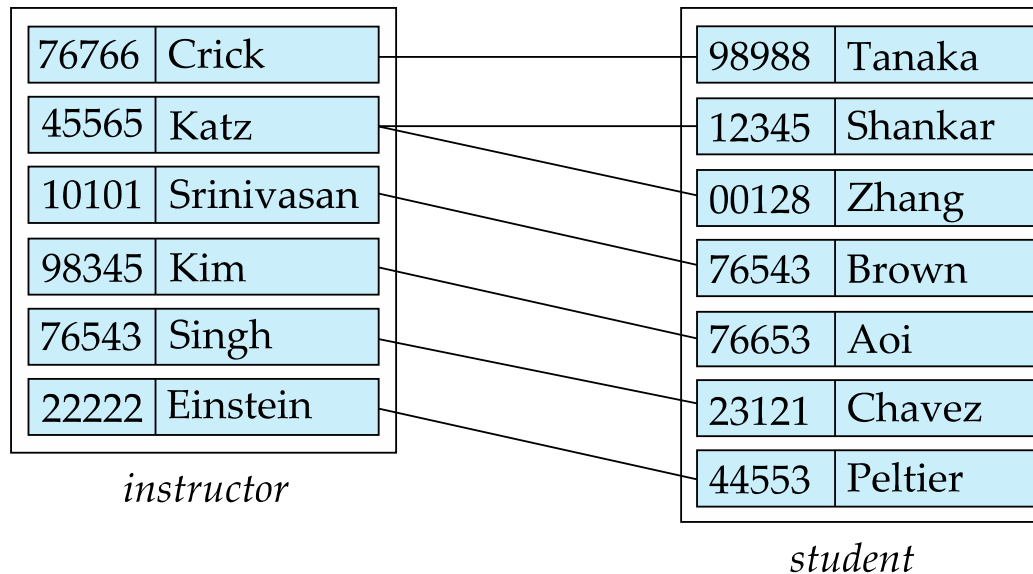
- Example:

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



Relationship Sets (Cont.)

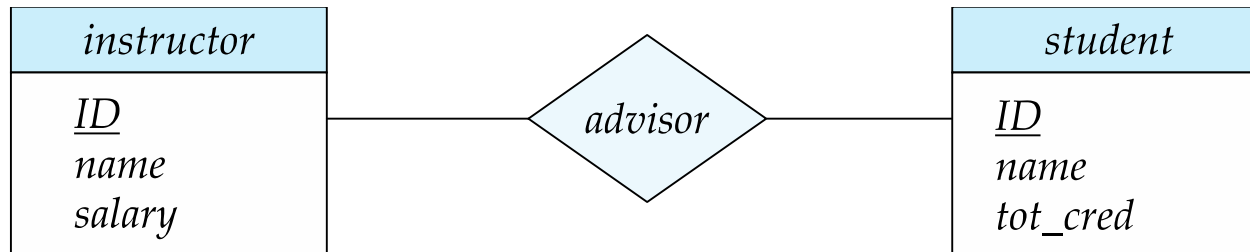
- Example: we define the relationship set *advisor* to denote the associations between students and the instructors who act as their advisors.
- Pictorially, we draw a line between related entities.





Representing Relationship Sets via ER Diagrams

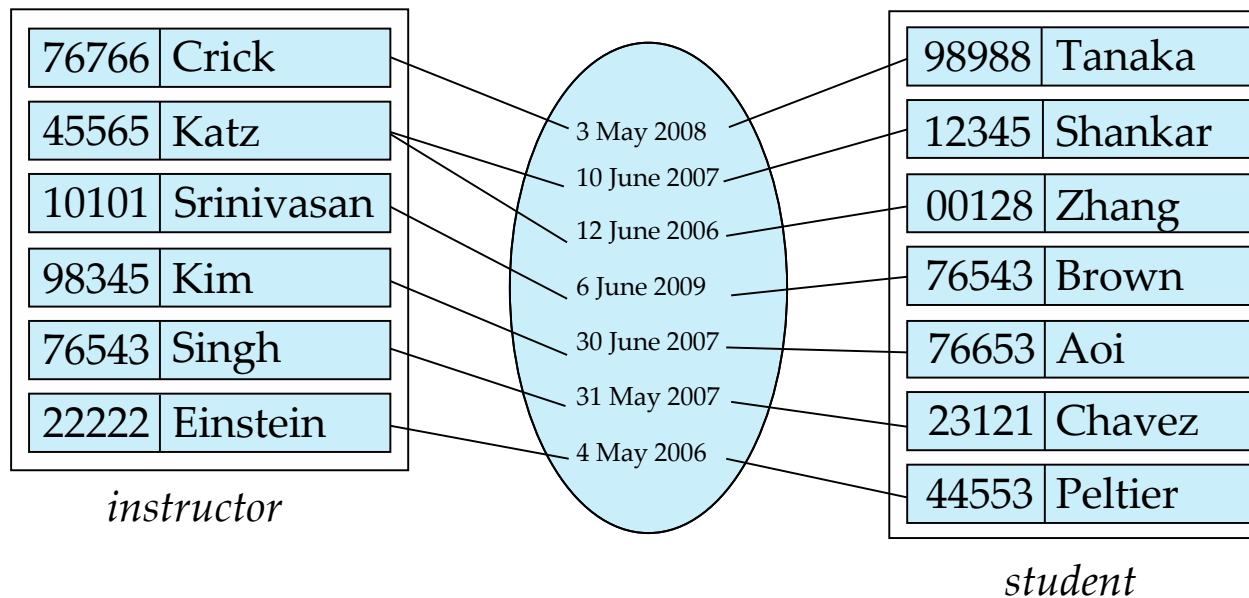
- Diamonds represent relationship sets.





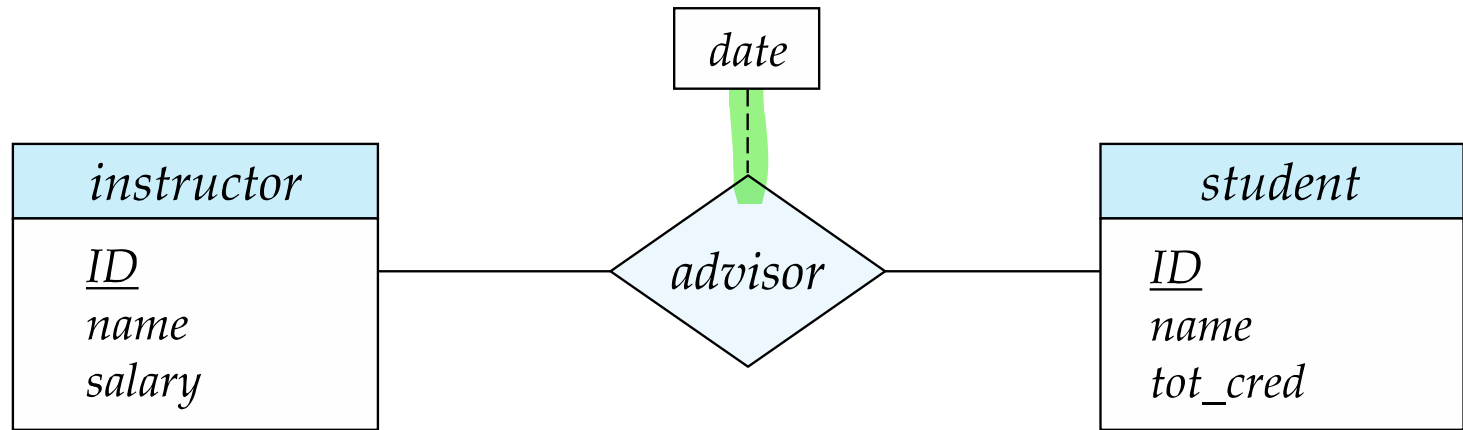
Relationship Sets (Cont.)

- 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





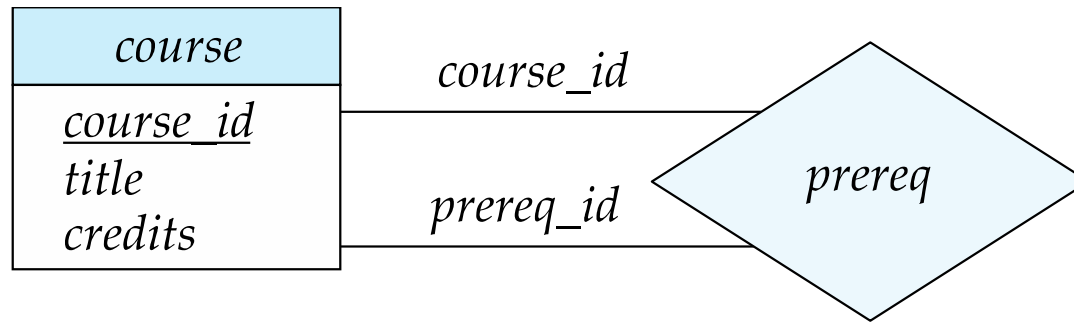
Relationship Sets with Attributes





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**.





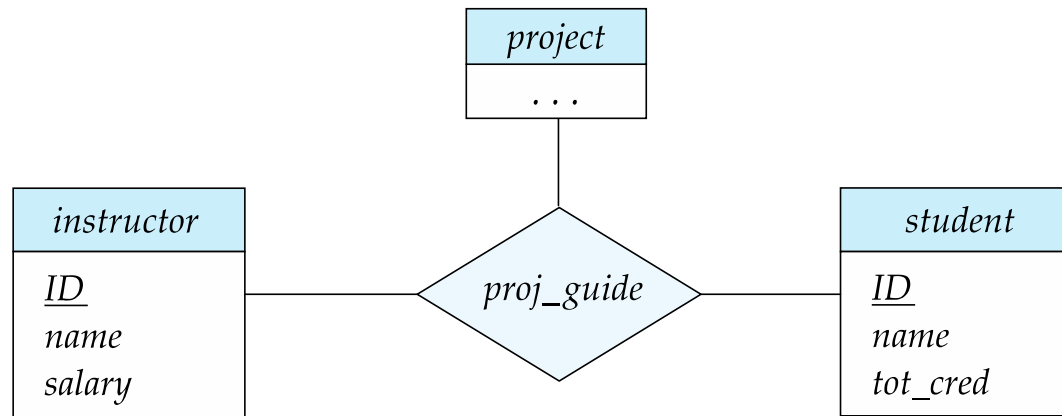
Degree of a Relationship Set

- 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*



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





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

Simple Attribute: An attribute that holds a single atomic value (e.g., Age, Name).

Single-Valued Attribute: An attribute that has only one value for each entity (e.g., DateOfBirth is single-valued because each student has only one birth date.).

Multi-Valued Attribute: An attribute that can have multiple values for a single entity (e.g., Phone Numbers).

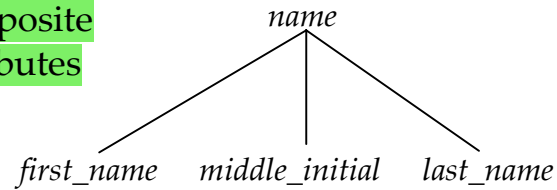
Complex Attribute: A nested attribute combining multiple attributes, including multi-valued ones (e.g., Full Address with Street, City, and Pincode).



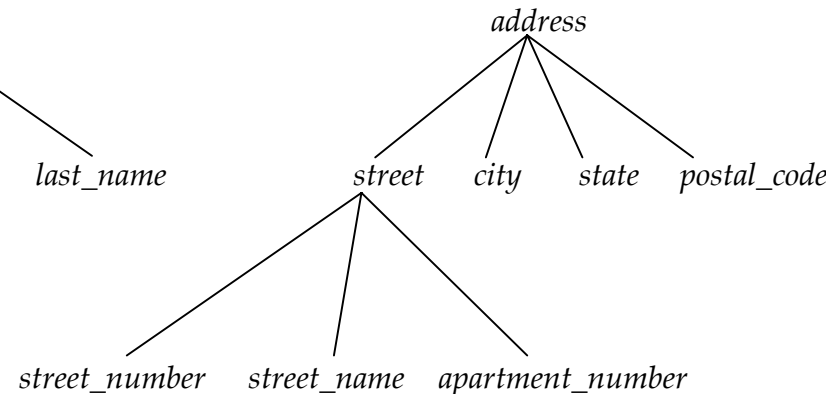
Composite Attributes

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

composite
attributes

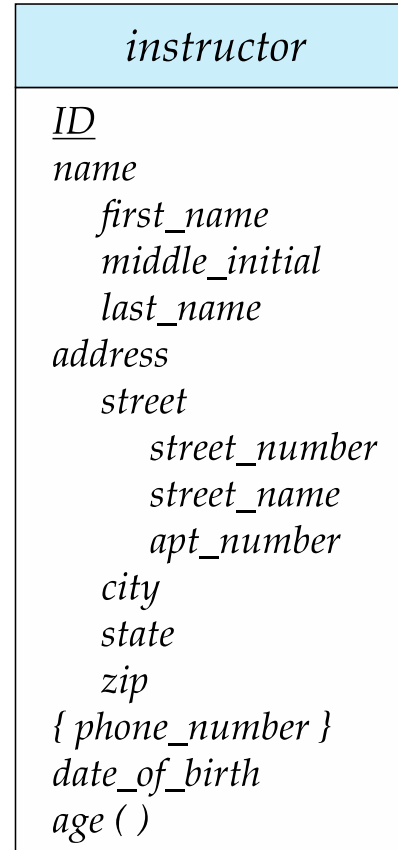


component
attributes





Representing Complex Attributes in ER Diagram



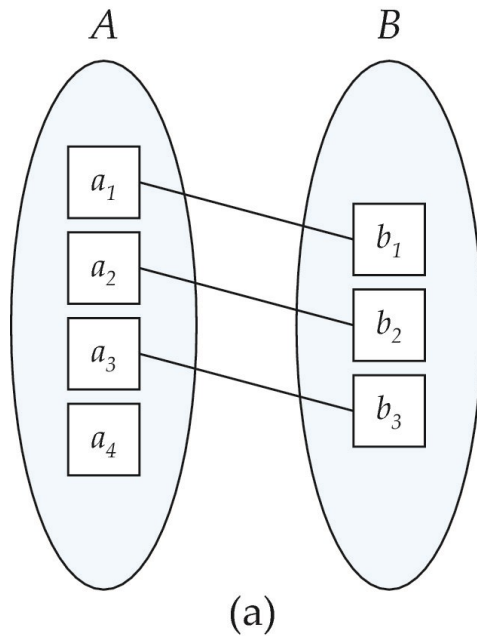


Mapping Cardinality Constraints

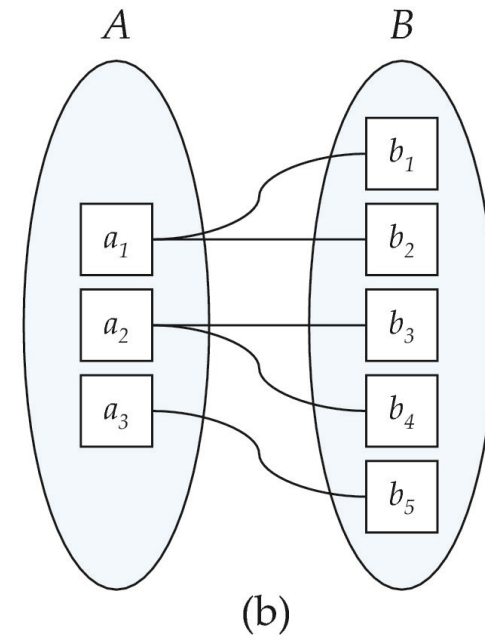
- Express the number of entities to which another entity can be associated via a relationship set.
- Most useful in describing binary relationship sets.
- For a binary relationship set the mapping cardinality must be one of the following types:
 - One to one
 - One to many
 - Many to one
 - Many to many



Mapping Cardinalities



One to one

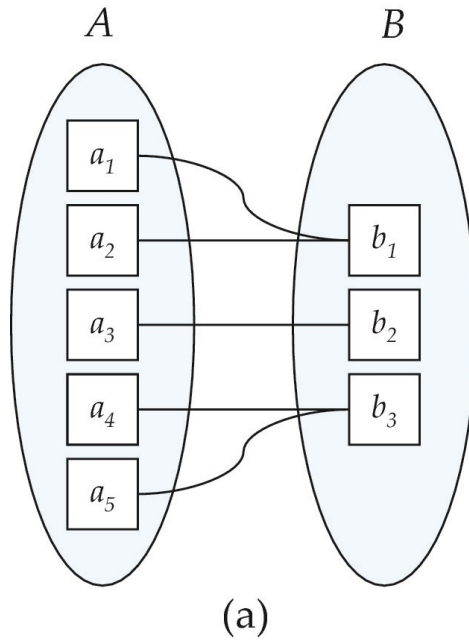


One to many

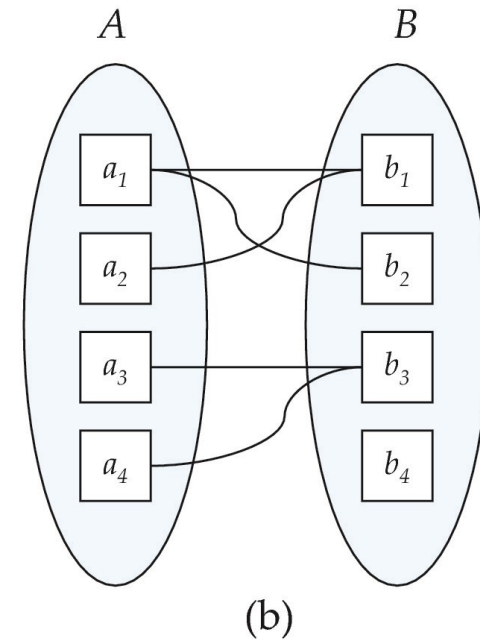
Note: Some elements in A and B may not be mapped to any elements in the other set



Mapping Cardinalities



Many to one



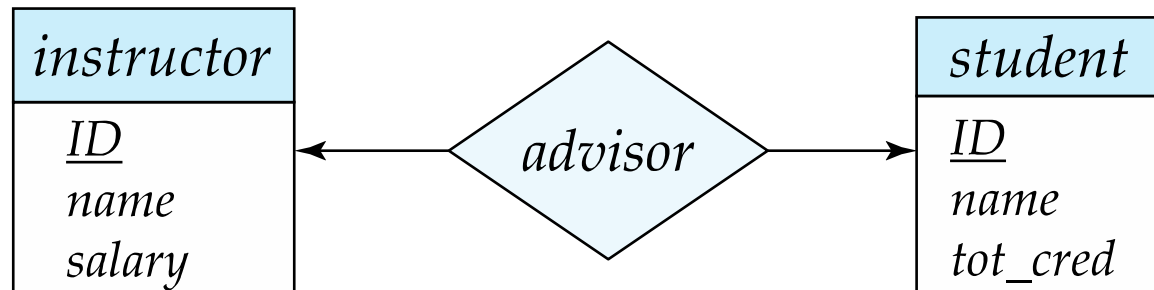
Many to many

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



Representing Cardinality Constraints in ER Diagram

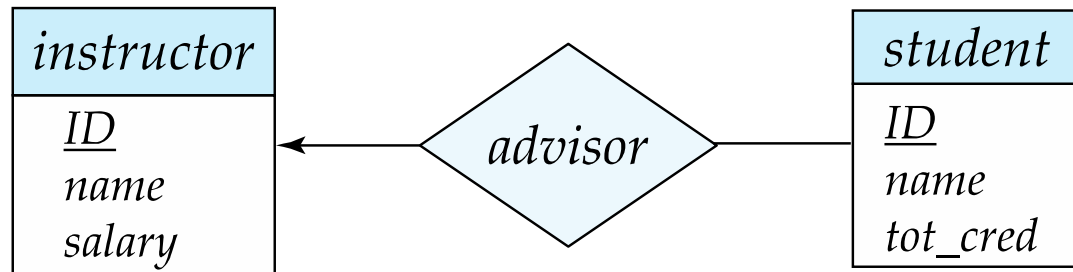
- We express cardinality constraints by drawing either a directed line (\rightarrow), signifying “one,” or an undirected line ($—$), signifying “many,” between the relationship set and the entity set.
- One-to-one relationship between an *instructor* and a *student* :
 - A student is associated with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





One-to-Many Relationship

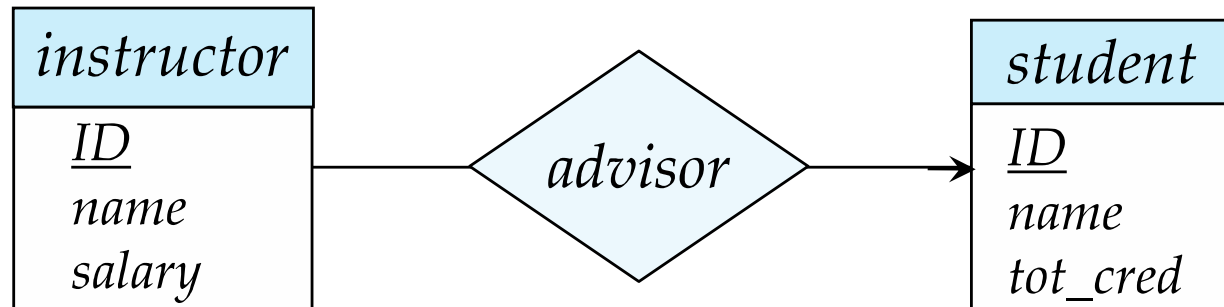
- one-to-many relationship between an *instructor* and a *student*
 - an instructor is associated with several (including 0) students via *advisor*
 - a student is associated with at most one instructor via *advisor*,





Many-to-One Relationships

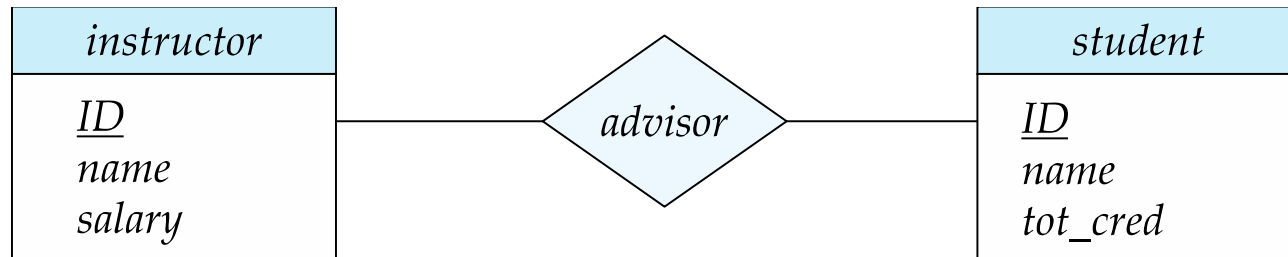
- In a many-to-one relationship between an *instructor* and a *student*,
 - an instructor is associated with **at most one** student via *advisor*,
 - and a student is associated with **several (including 0)** instructors via *advisor*





Many-to-Many Relationship

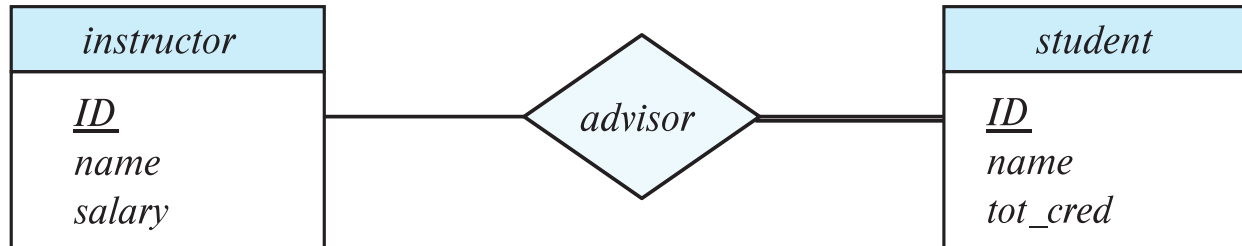
- An instructor is associated with several (possibly 0) students via *advisor*
- A student is associated with several (possibly 0) instructors via *advisor*





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



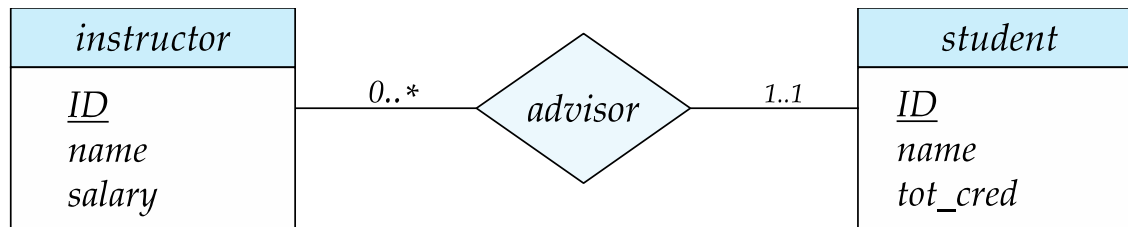
participation of *student* in *advisor* relation is total

- 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.
- Example



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



Cardinality Constraints on Ternary Relationship

`project_guide` is relationship set

- 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 relations are distinguished. We will consider:
 - Entity sets
 - Relationship sets.
 - Weak entity sets



Primary key for Entity Sets

- By definition, individual entities are distinct.
- From database perspective, the differences among them must be expressed in terms of their attributes.
- The values of the attribute values of an entity must be such that they can uniquely identify the entity.
 - No two entities in an entity set are allowed to have exactly the same value for all attributes.
- A key for an entity is a set of attributes that suffice to distinguish entities from each other



Primary Key for Relationship Sets

- To distinguish among the various relationships of a relationship set we use the individual primary keys of the entities in the relationship set.
 - Let R be a relationship set involving entity sets E_1, E_2, \dots, E_n
 - The primary key for R consists of the union of the primary keys of entity sets E_1, E_2, \dots, E_n
 - If the relationship set R has attributes a_1, a_2, \dots, a_m associated with it, then the primary key of R also includes the attributes a_1, a_2, \dots, a_m
- Example: relationship set “advisor”.
 - The primary key consists of *instructor.ID* and *student.ID*
- The choice of the primary key for a relationship set depends on the mapping cardinality of the relationship set.

The choice of the primary key for a relationship set depends on its mapping cardinality because the key should uniquely identify each relationship instance. The mapping cardinality (one-to-one, one-to-many, or many-to-many) determines which attributes or combination of attributes must be included in the primary key.

For example, consider a “Works_On” relationship between “Employee” and “Project”:

One-to-One (1:1) – Each employee is assigned to at most one project, and each project has only one assigned employee. Here, either Emp_ID or Project_ID alone can serve as the primary key.

One-to-Many (1:M) – An employee can work on multiple projects, but each project has only one employee. In this case, (Emp_ID, Project_ID) together form the primary key since a single Emp_ID can appear multiple times, but each (Emp_ID, Project_ID) pair is unique.

Many-to-Many (M:N) – An employee can work on multiple projects, and each project can have multiple employees. Here, neither Emp_ID nor Project_ID alone is sufficient, so the primary key must be (Emp_ID, Project_ID) to uniquely identify each association. Thus, the primary key selection depends on ensuring uniqueness for relationship instances while adhering to the mapping cardinality.



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.



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
- To deal with this problem, we treat the relationship *sec_course* as a special relationship that provides extra information, in this case, the *course_id*, required to identify *section* entities uniquely.
- A **weak entity set** is one whose existence is dependent on another entity, called its **identifying entity**
- Instead of associating a primary key with a weak entity, we use the identifying entity, along with extra attributes called **discriminator** to uniquely identify a weak entity.



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

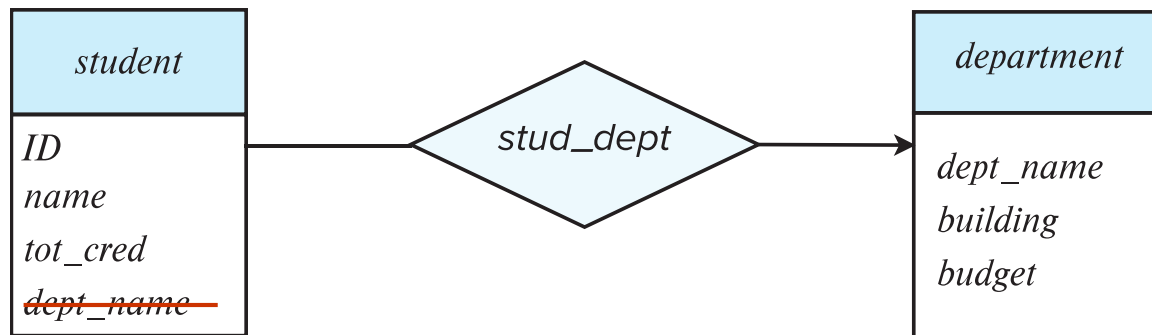
- In E-R diagrams, a weak entity set is depicted via a double rectangle.
- We underline the discriminator of a weak entity set with a dashed line.
- The relationship set connecting the weak entity set to the identifying strong entity set is depicted by a double diamond.
- Primary key for *section* –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





Redundant Attributes

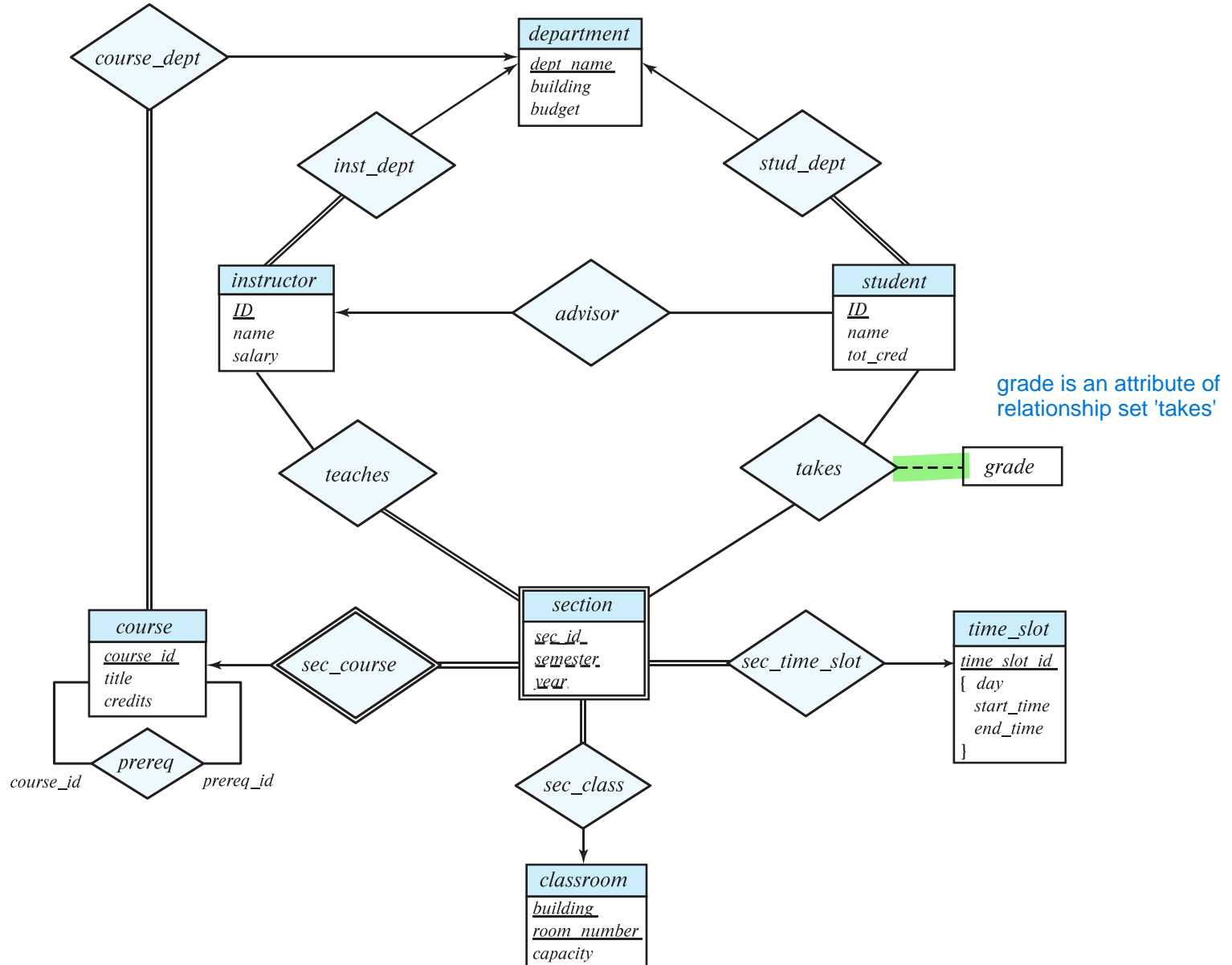
- Suppose we have entity sets:
 - *student*, with attributes: *ID*, *name*, *tot_cred*, *dept_name*
 - *department*, with attributes: *dept_name*, *building*, *budget*
- We model the fact that each student has an associated department using a relationship set *stud_dept*
- The attribute *dept_name* in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



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*)

- Example





Representation of Entity Sets with Composite Attributes

<i>instructor</i>
<u>ID</u>
name
<i>first_name</i>
<i>middle_initial</i>
<i>last_name</i>
address
street
<i>street_number</i>
<i>street_name</i>
<i>apt_number</i>
city
state
zip
{ <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

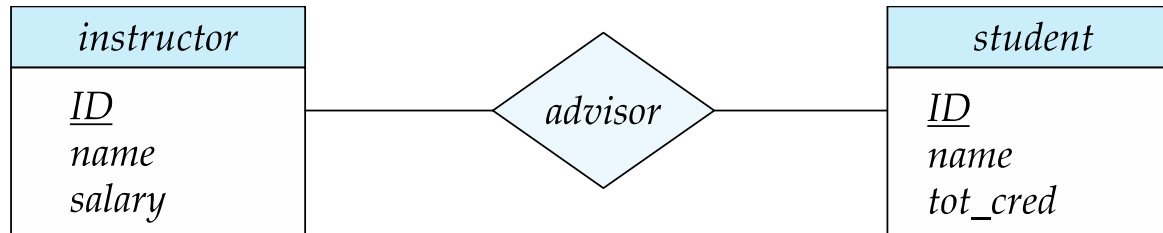
- A multivalued attribute M of an entity E is represented by a separate schema EM
- Schema EM has attributes corresponding to the primary key of E and an attribute corresponding to multivalued attribute M
- Example: Multivalued attribute *phone_number* of *instructor* is represented by a schema:
 $inst_phone = (\underline{ID}, \underline{phone_number})$
- 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)



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*)





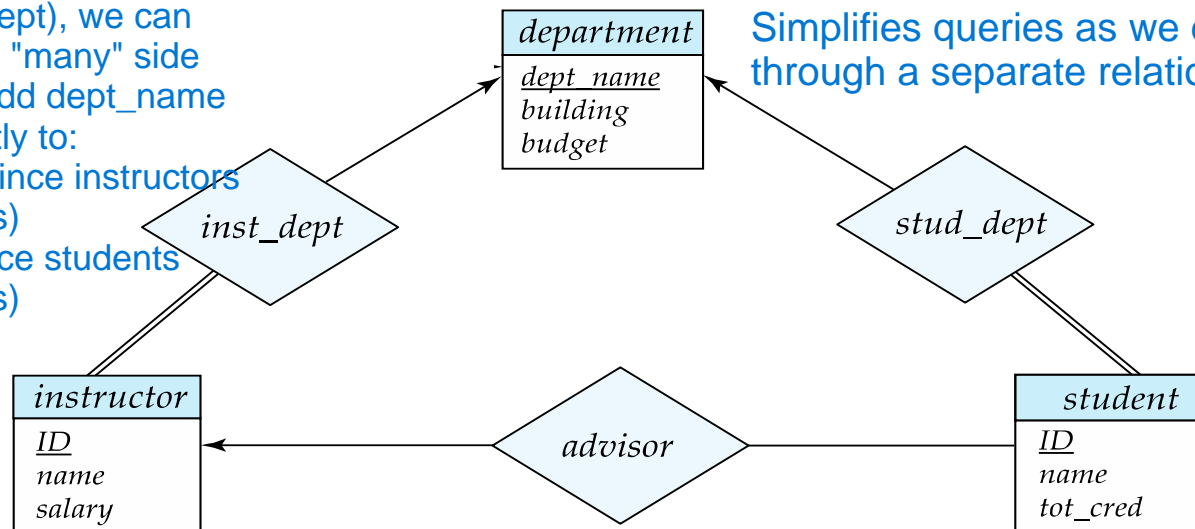
Redundancy of Schemas

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

Instead of creating separate schemas for these relationships (*inst_dept* and *stud_dept*), we can add an attribute to the “many” side
In this case, we can add *dept_name* as a foreign key directly to:
The instructor table (since instructors belong to departments)
The student table (since students belong to departments)

Benefits:

Reduces the number of tables needed in the database
Simplifies queries as we don't need to join through a separate relationship table





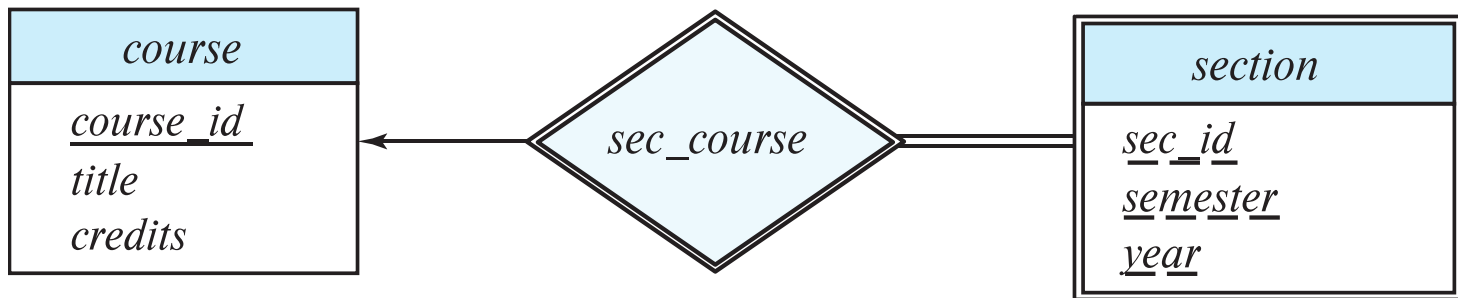
Redundancy of Schemas (Cont.)

- 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





Extended E-R Features



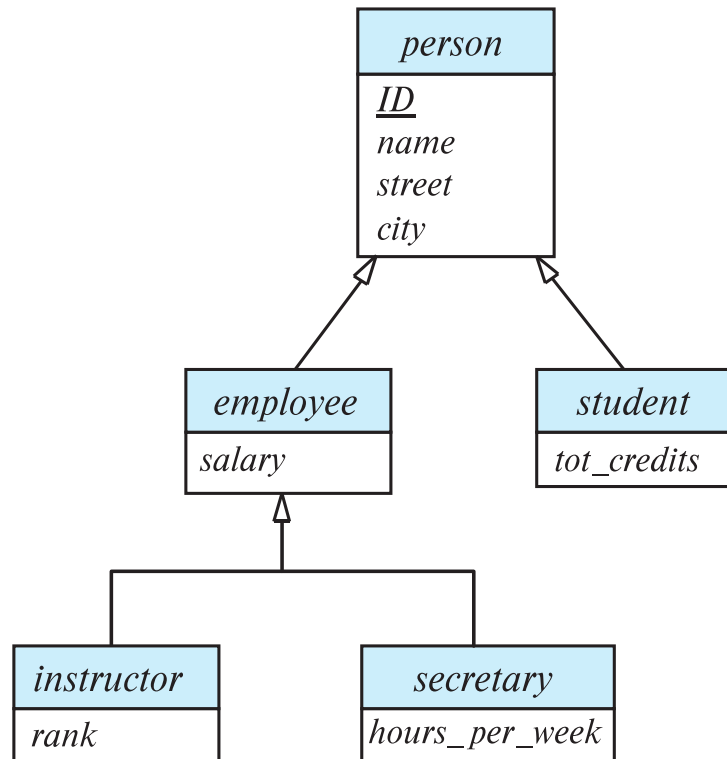
Specialization

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



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.



Completeness constraint

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



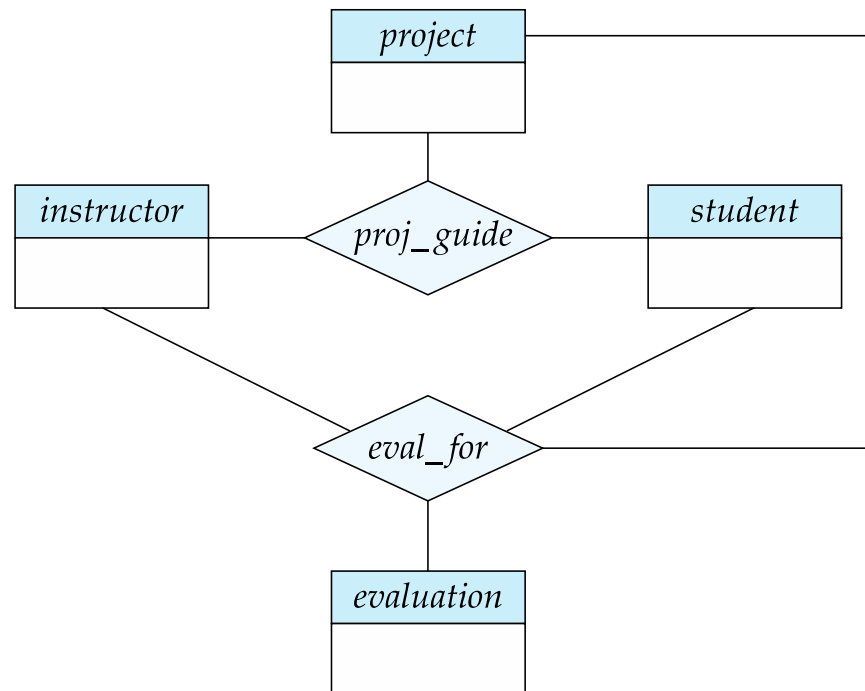
Completeness constraint (Cont.)

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



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





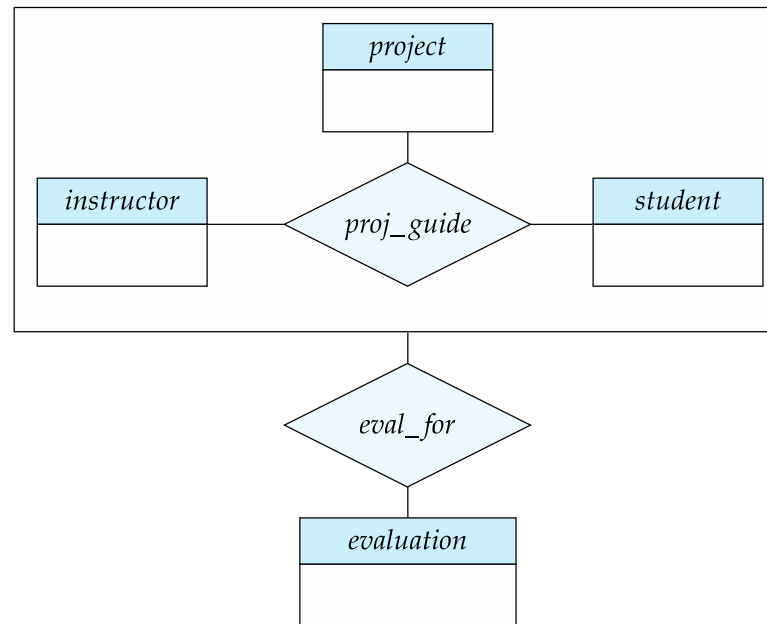
Aggregation (Cont.)

- 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





Reduction to Relational Schemas

- To represent aggregation, create a schema containing
 - Primary key of the aggregated relationship,
 - The primary key of the associated entity set
 - Any descriptive attributes
- In our example:
 - The schema *eval_for* is:
$$eval_for(s_ID, project_id, i_ID, evaluation_id)$$
 - The schema *proj_guide* is redundant.

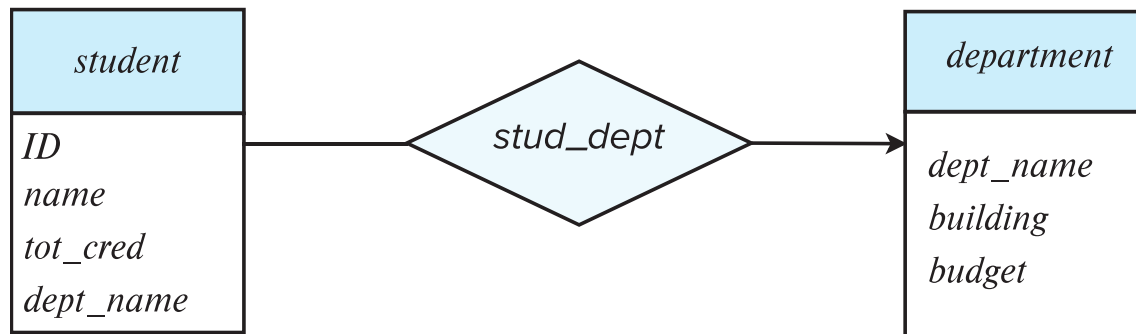


Design Issues

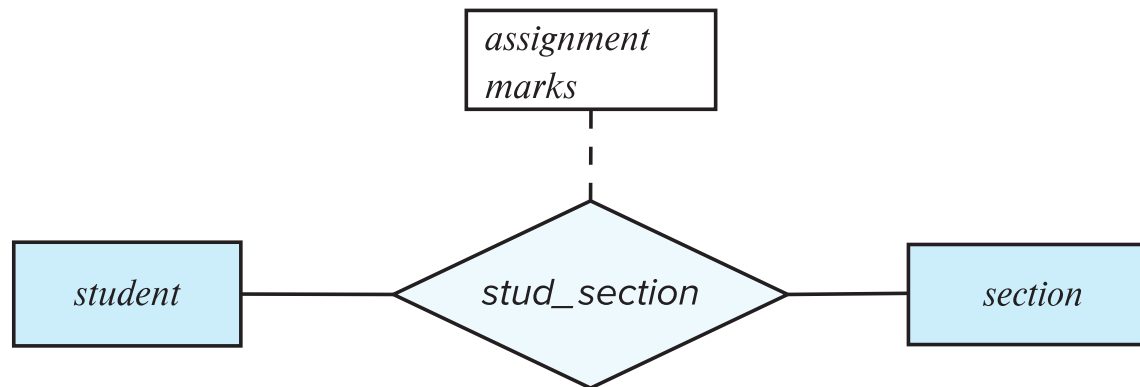


Common Mistakes in E-R Diagrams

- Example of erroneous E-R diagrams



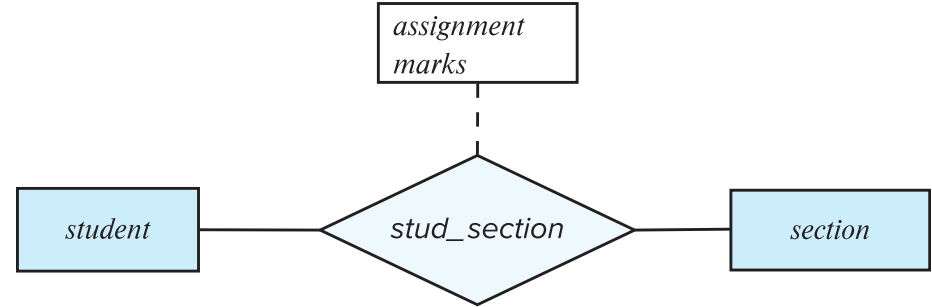
(a) Incorrect use of attribute



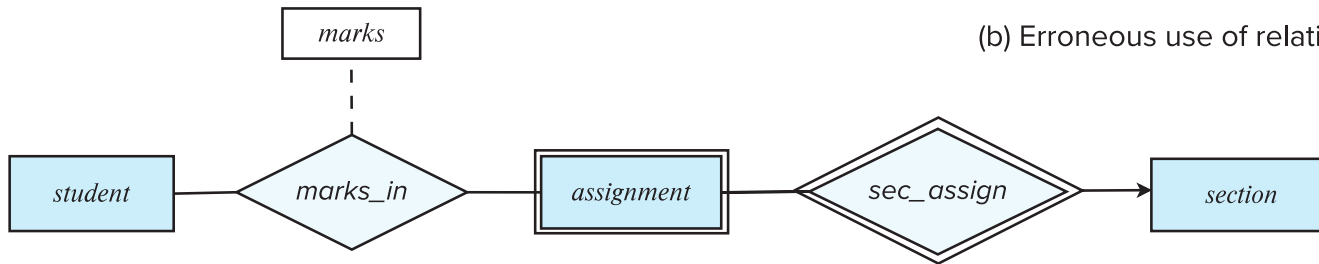
(b) Erroneous use of relationship attributes



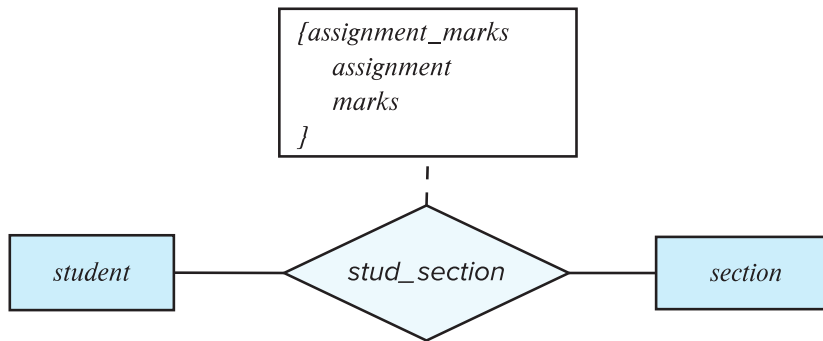
Common Mistakes in E-R Diagrams (Cont.)



(b) Erroneous use of relationship attributes



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

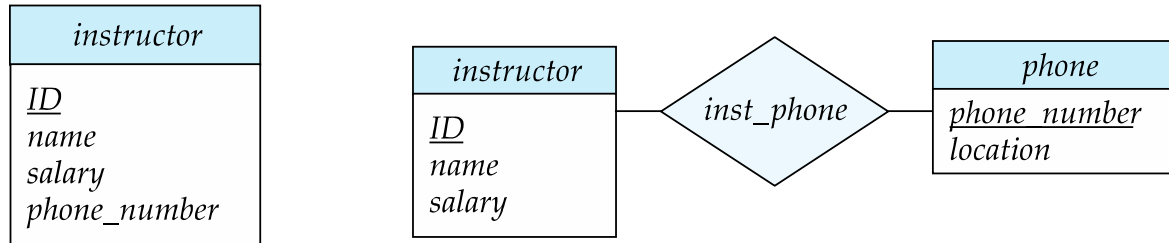


(d) Correct alternative to erroneous E-R diagram (b)



Entities vs. Attributes

- Use of entity sets vs. attributes



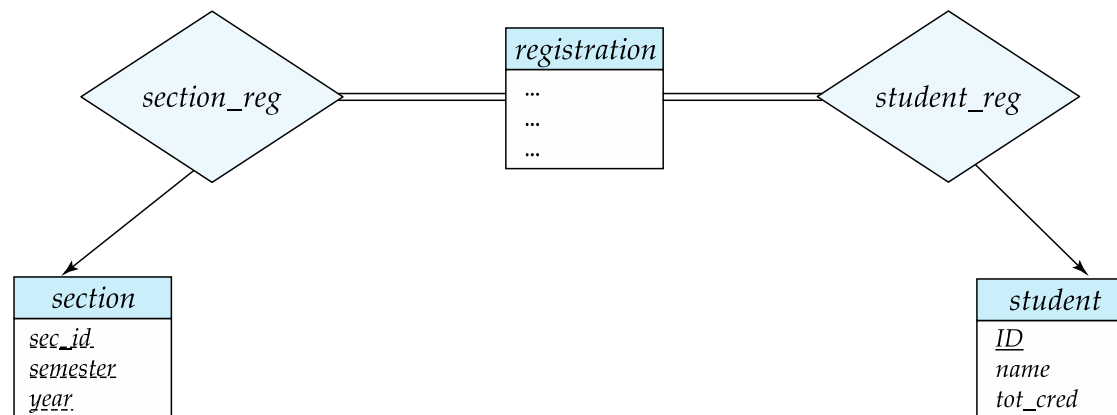
- Use of phone as an entity allows extra information about phone numbers (plus multiple phone numbers)



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



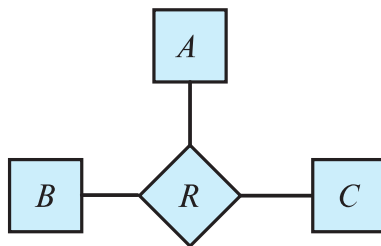
Binary Vs. Non-Binary Relationships

- 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*

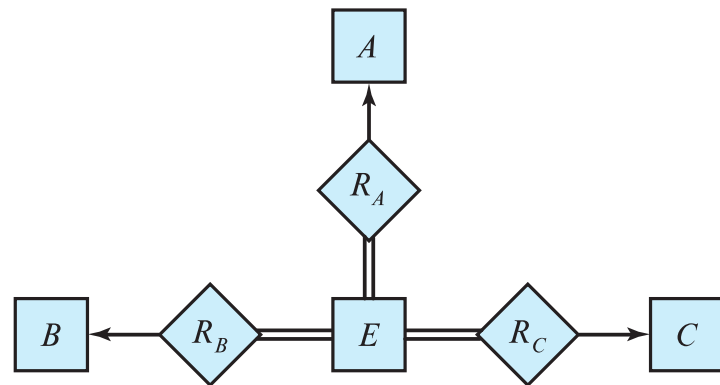


Converting Non-Binary Relationships to Binary Form

- 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

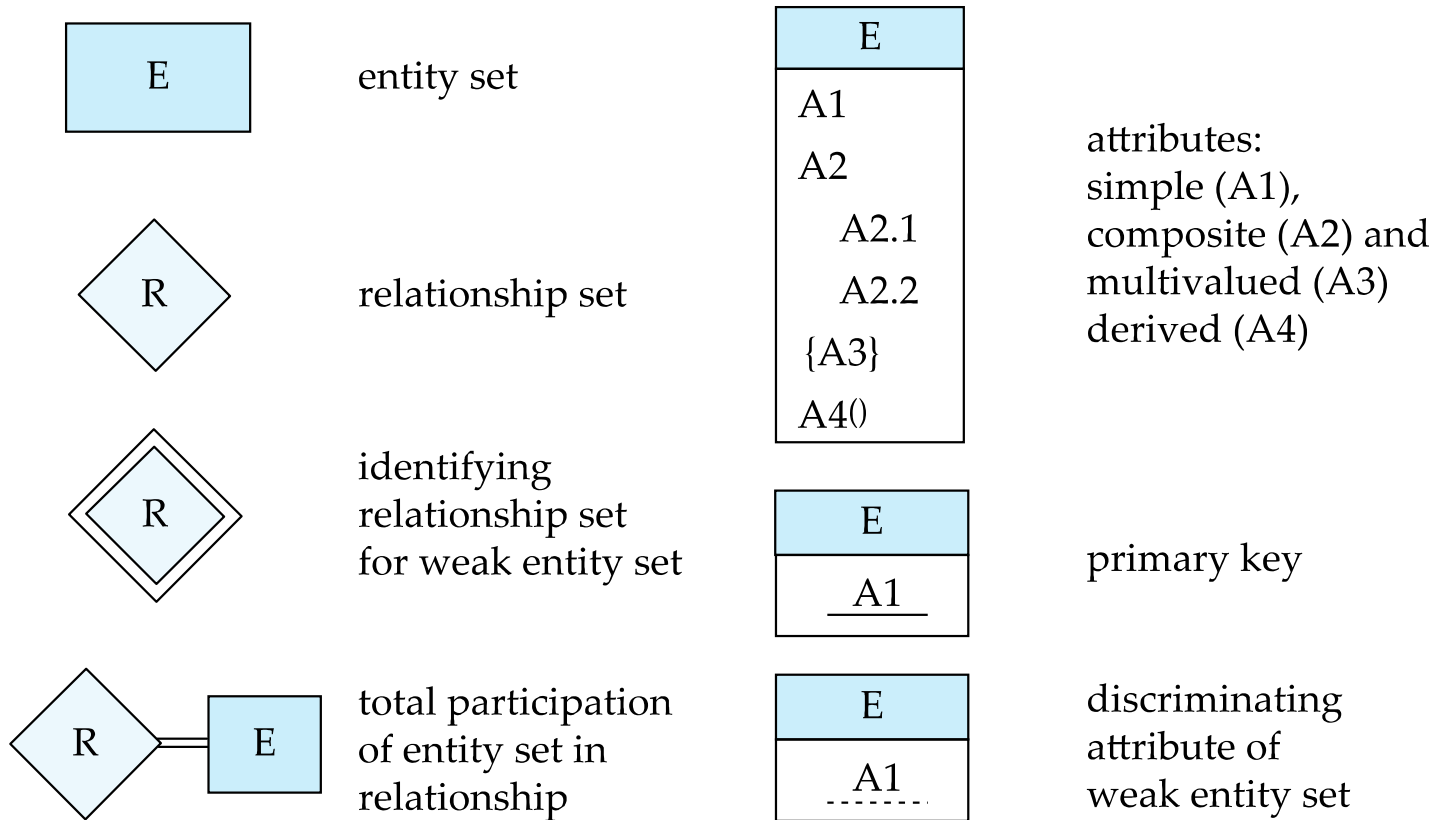


E-R Design Decisions

- 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.

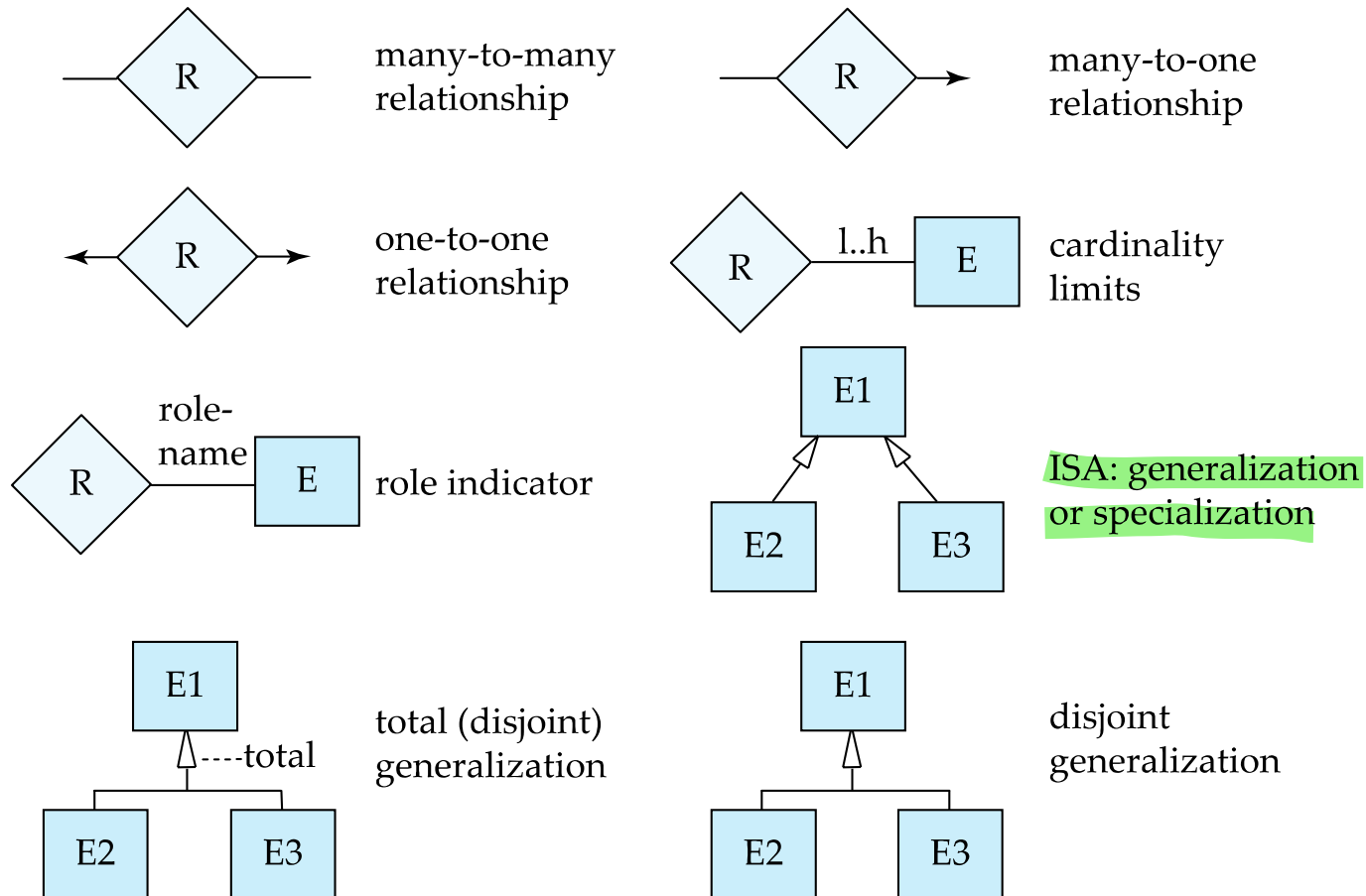


Summary of Symbols Used in E-R Notation





Symbols Used in E-R Notation (Cont.)

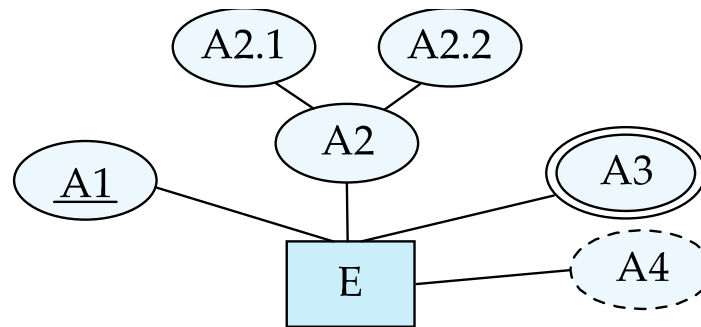




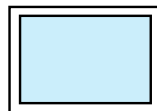
Alternative ER Notations

- Chen, IDE1FX, ...

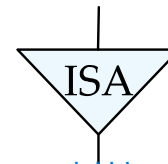
entity set E with
simple attribute A1,
composite attribute A2,
multivalued attribute A3,
derived attribute A4,
and primary key A1



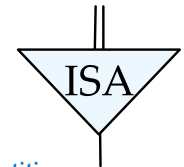
weak entity set



generalization



total
generalization



In databases, ISA (Is-A) relationships are used in generalization and specialization to model hierarchical relationships between entities.

Generalization

Definition: Combining multiple specific entities into a higher-level, more general entity.

Example: Consider the entities Car, Truck, and Bike. They all share common attributes like vehicle_id and license_plate. These entities can be generalized into a Vehicle entity.

Implementation:

Single Table Inheritance: One table for Vehicle with a type column (e.g., vehicle_type).

Table per Entity: Separate tables for Car, Truck, and Bike, each inheriting Vehicle attributes.

Specialization

Definition: The reverse of generalization—breaking down a general entity into more specialized entities based on specific attributes or behaviors.

Example: A Person entity can be specialized into Student and Professor entities, each with unique attributes (grade for Student, salary for Professor).

Implementation:

Separate tables for Student and Professor, linked to the Person table via foreign keys.

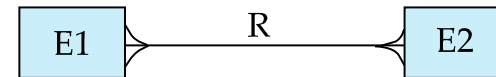
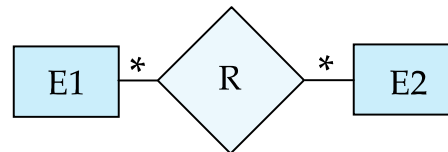


Alternative ER Notations

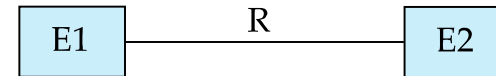
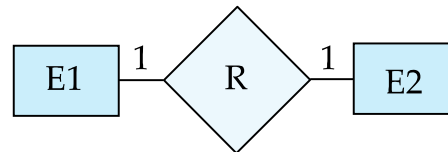
Chen

IDE1FX (Crows foot notation)

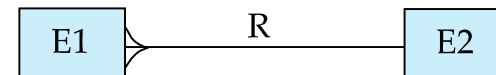
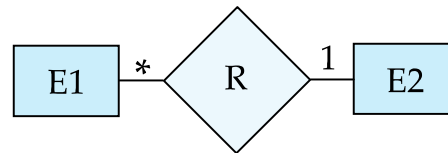
many-to-many
relationship



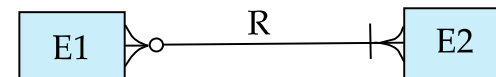
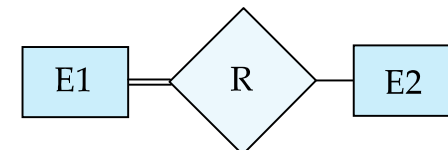
one-to-one
relationship



many-to-one
relationship



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





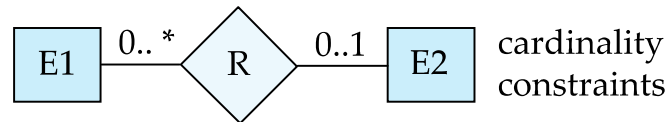
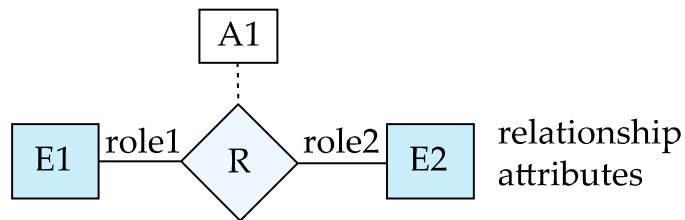
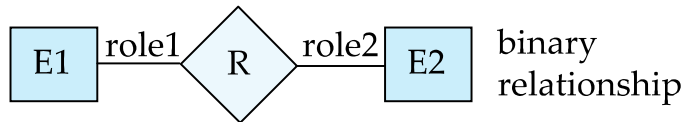
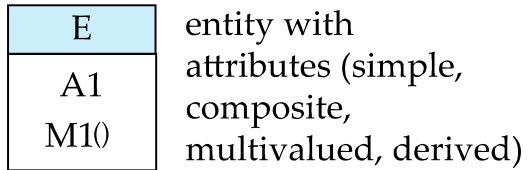
UML

- **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.

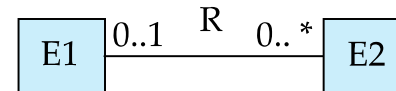
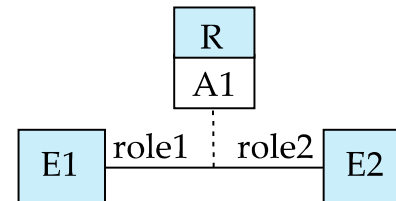
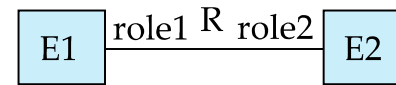
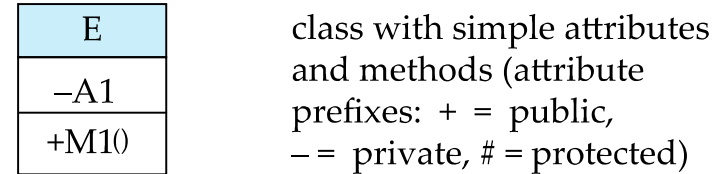


ER vs. UML Class Diagrams

ER Diagram Notation



Equivalent in UML

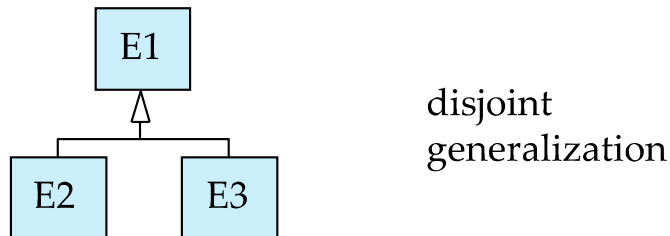
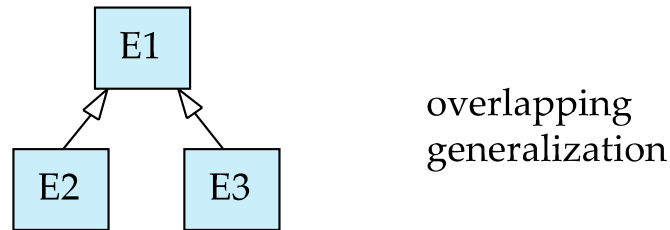
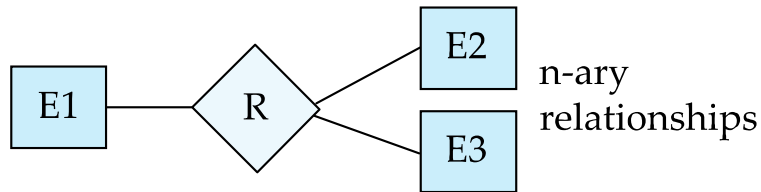


* Note reversal of position in cardinality constraint depiction

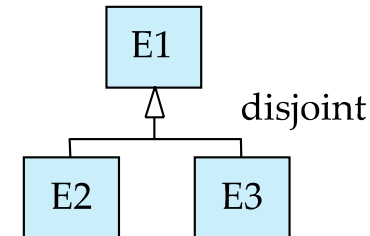
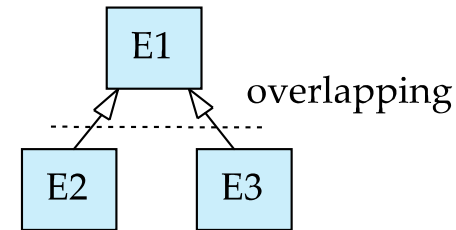
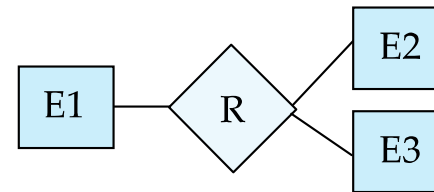


ER vs. UML Class Diagrams

ER Diagram Notation



Equivalent in UML



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



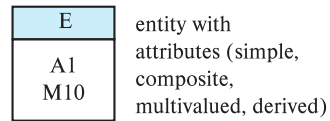
UML Class Diagrams (Cont.)

- 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.

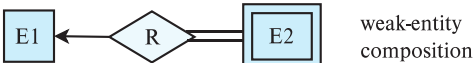
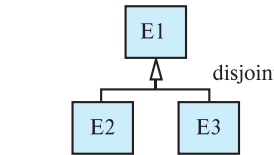
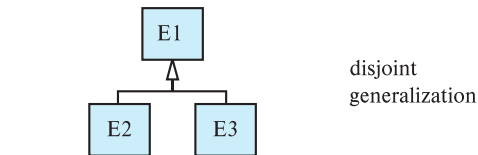
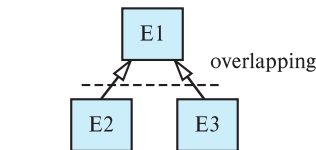
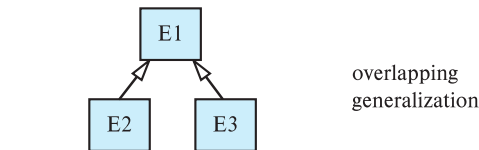
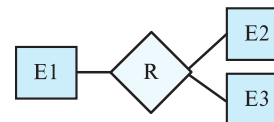
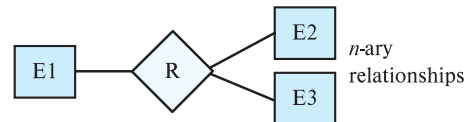
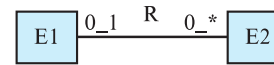
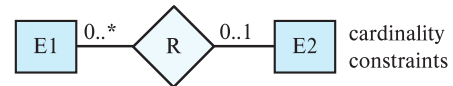
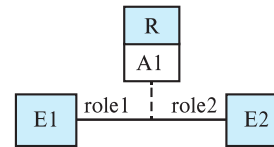
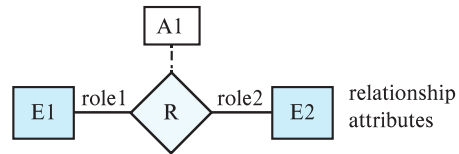
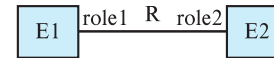
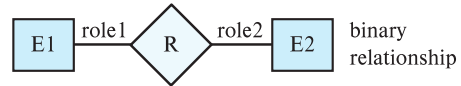
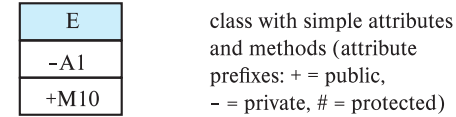


ER vs. UML Class Diagrams

ER Diagram Notation



Equivalent in UML





Other Aspects of Database Design

- Functional Requirements
- Data Flow, Workflow
- Schema Evolution



End of Chapter 6