Part 3: The Entity-Relationship Model

Database System Concepts, 7th Ed.

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 that will be performed on the data

Design Phases

- 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?
    - Computing 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 repeated 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
  - Models an enterprise as a collection of *entities* and *relationships*
    - Entity: a “thing” or “object” in the enterprise that is distinguishable from other objects
      - Described by a set of *attributes*
    - Relationship: an association among several entities
  - Represented diagrammatically by an *entity-relationship diagram*
- Normalization Theory
  - Formalize what designs are bad, and test for them

Outline of the ER Model

ER model -- Database Modeling

- The ER data model was developed to facilitate database design by allowing specification of an **enterprise schema** that represents the overall logical structure of a database
- The ER data model employs three basic concepts:
  - entity sets
  - relationship sets
  - attributes
- The ER model also has an associated diagrammatic representation, the **ER diagram**, which can express the overall logical structure of a database graphically

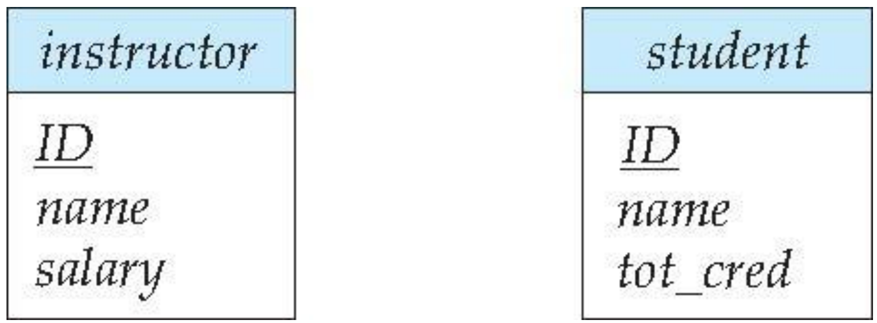
Entity Sets

- An **entity** is an object that exists and is distinguishable from other objects.
  - Example: specific person, company, event, plant
- An **entity set** is a set of entities of the same type that share the same properties
  - Example: set of all persons, companies, products
- 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)	*advisor*	22222 (Einstein)
student entity	relationship set	*instructor* entity

- A **relationship set** is a mathematical relation among entities, each taken from entity sets

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

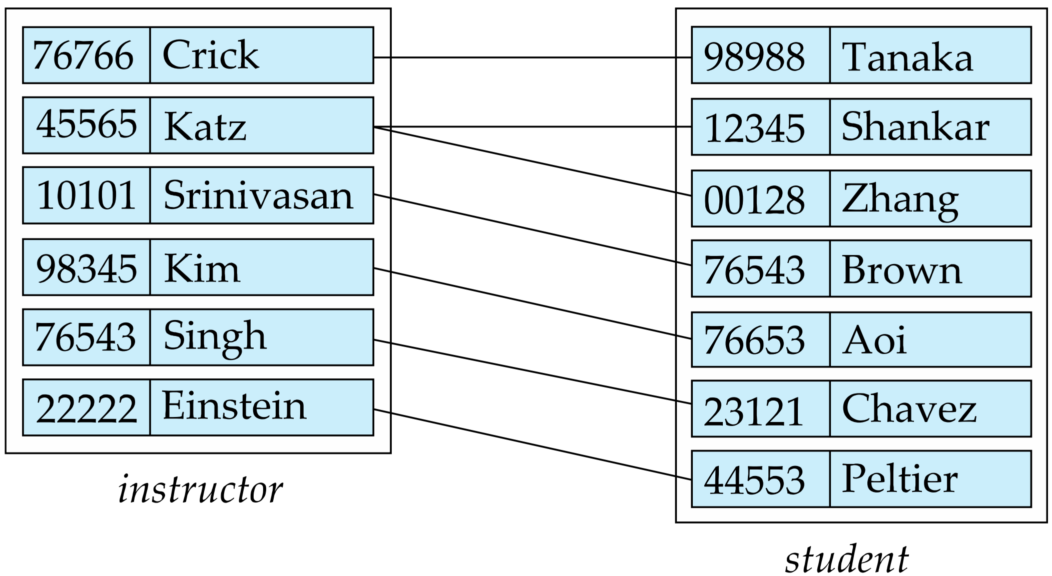
  where $(e_1, e_2, \dots, e_n)$ is a relationship

  - Example:

    $(44553, 22222) \in advisor$

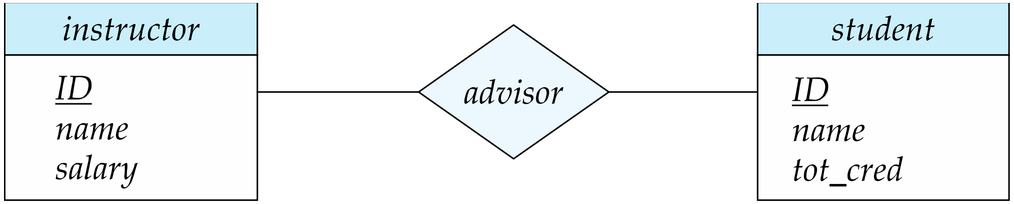
Relationship Sets

- 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



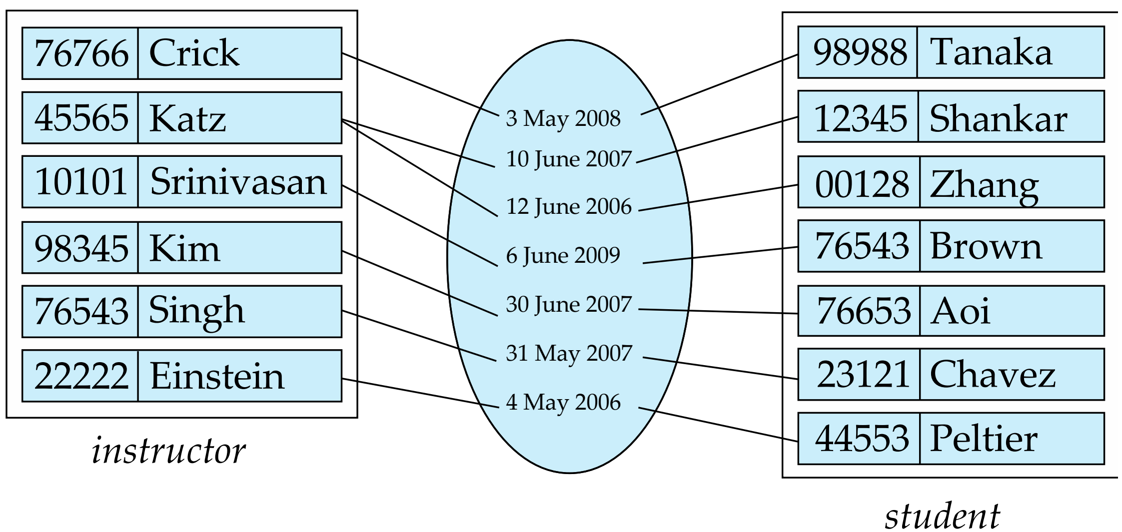
Representing Relationship Sets via ER Diagrams

- Diamonds represent relationship sets

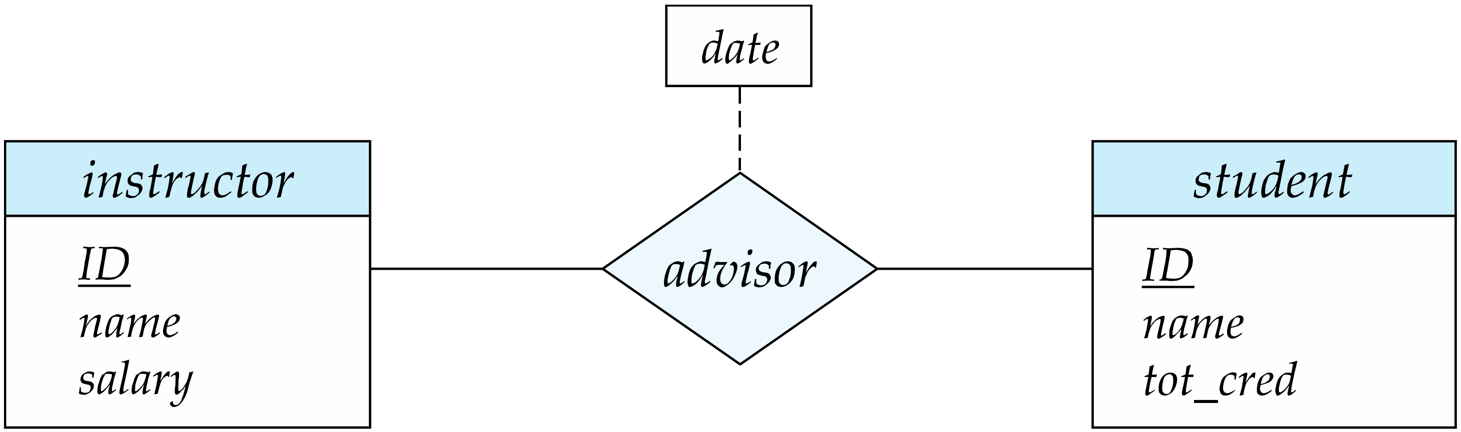


Relationship Sets

- An attribute can also be associated with a relationship set, sometimes called a **descriptive attribute**
- 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

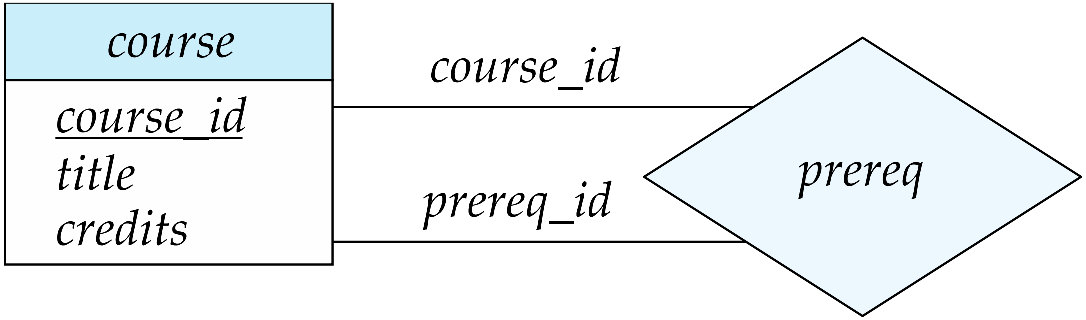


Relationship Sets with Attributes



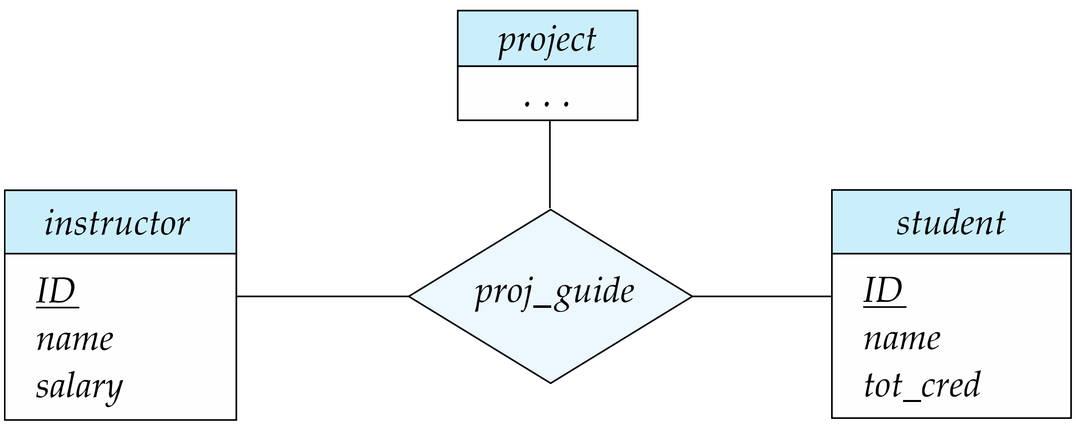
Roles

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

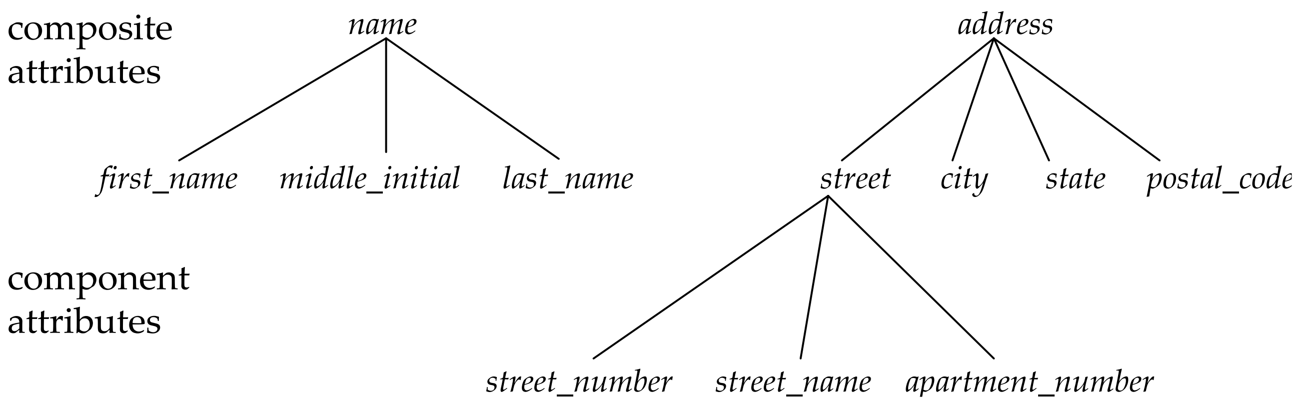


Complex Attributes

- Attribute types:
  - **Simple** and **composite** attributes
    - Name may consist of first name and last name
  - **Single-valued** and **multivalued** attributes
    - Example: multivalued attribute: phone_numbers
  - **Derived** attributes
    - Can be computed from other attributes
    - Example: age, given date_of_birth
- **Domain** – the set of permitted values for each attribute

Composite Attributes

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



Representing Complex Attributes in ER Diagram

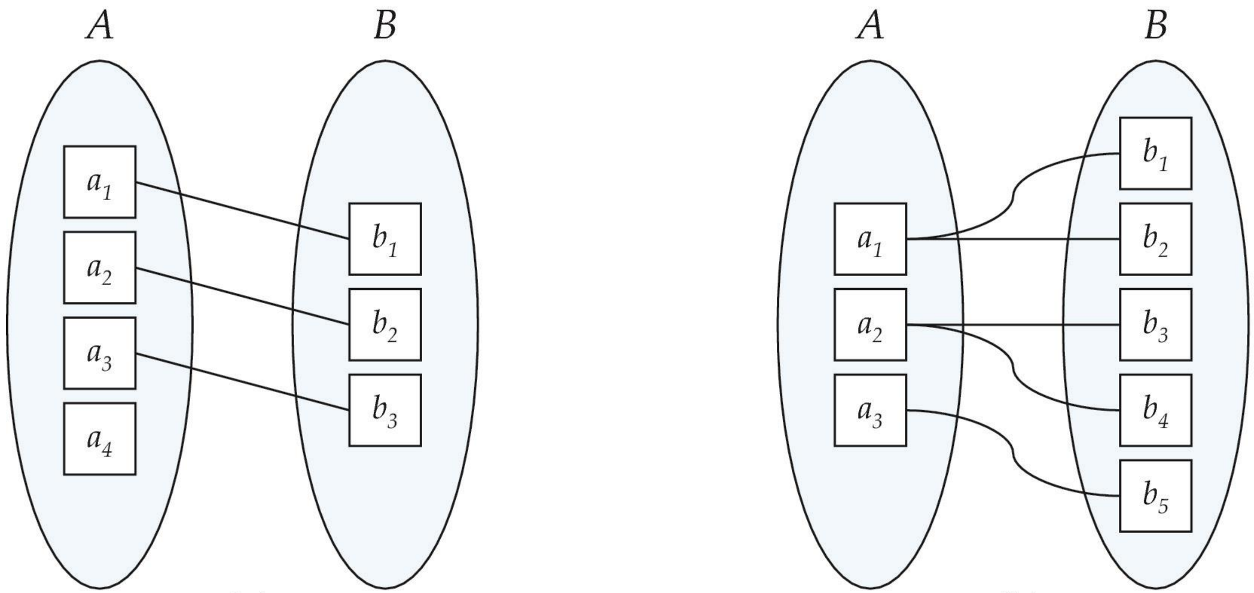
instructor

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

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

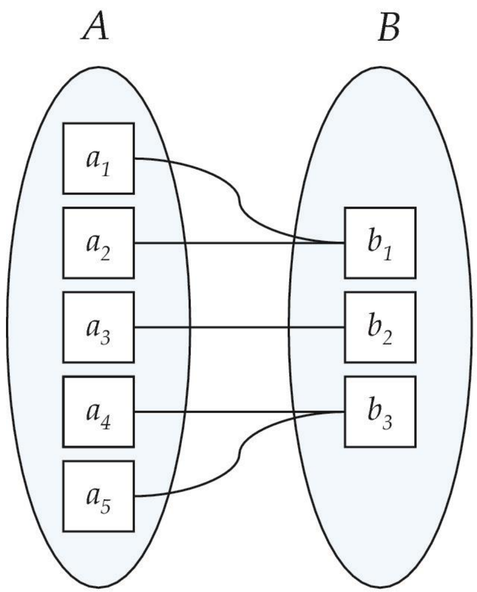


(a) (b)

One-to-one One-to-many

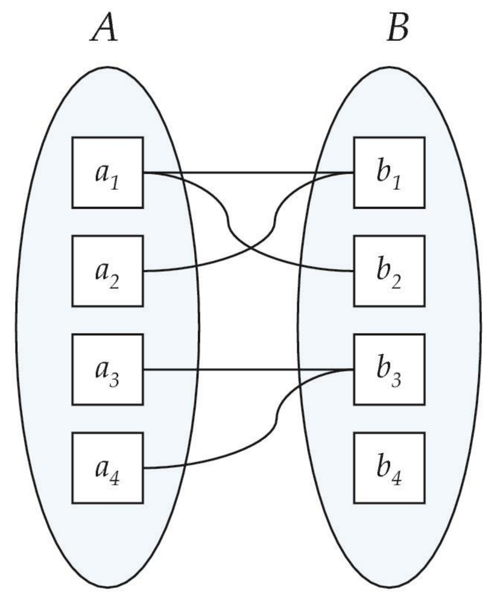
Note: Some elements in *A* and *B* may not be mapped to any elements in the other set

Mapping Cardinalities



(a)

Many-to-one



(b)

Many-to-many

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

Representing Cardinality Constraints in ER Diagram

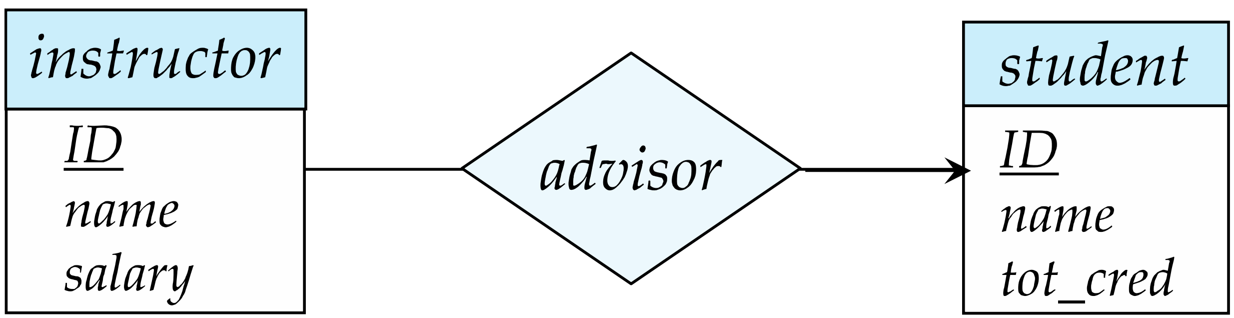
- We express cardinality constraints by drawing either a directed line (→), signifying "one," or an undirected line (—), signifying "many," between the relationship set and the entity set
- One-to-one relationship between an *instructor* and a *student*

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*

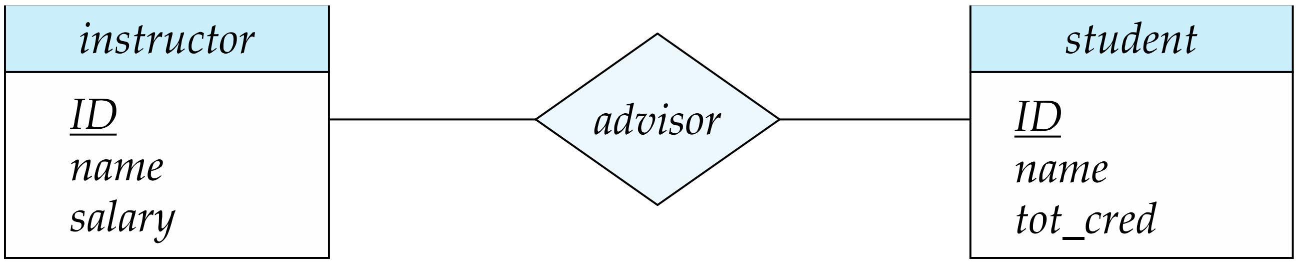
Many-to-One Relationship

- 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

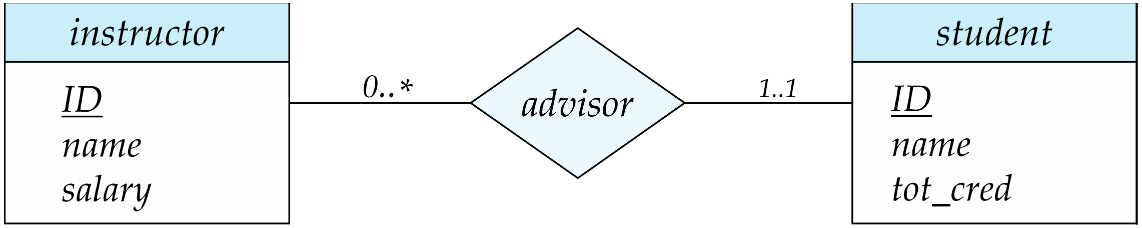
participation of *student* in *advisor r*elation is total

  - 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

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 a 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, ... E_n
  - The primary key for R is consists of the union of the primary keys of entity sets E_1, E_2, ... E_n
- Example: relationship set *advisor*
  - The primary key consists of *instructor.ID* and *student.ID*

Choice of Primary key for Binary Relationship

- Many-to-Many relationships. The 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
  - If a department has many instructors, then the primary key of the instructor-department relationship is simply the primary key of instructor
- Many-to-one relationships. The primary key of the "Many" side is a minimal superkey and is used as the primary key
  - If the relationship is many-to-one from *student* to *instructor*—that is, each student can have at most one advisor—then the primary key of *advisor* is simply the primary key of *student*
- 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 some 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

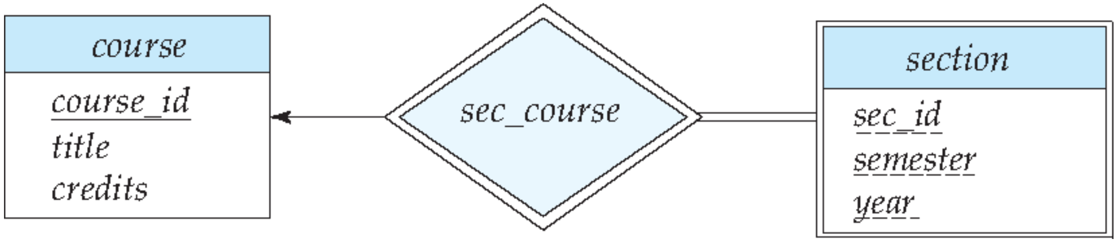
- An alternative way to deal with this redundancy is to not store the attribute *course_id* in the *section* entity and to only store the remaining attributes *section_id*, *year*, and *semester*
  - However, the entity set *section* then does not have enough attributes to identify a particular *section* 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 attributes**, to uniquely identify a weak entity

Weak Entity Sets

- 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*, even though we have dropped the attribute *course_id* from the entity set *section*

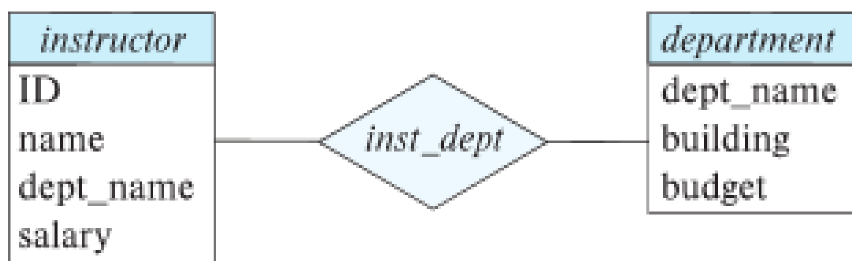
Expressing Weak Entity Sets

- 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
- In general, a weak entity must have total participation in its identifying relationship set, and the relationship is many-to-one towards the identifying entity set
- Primary key for *section* – (*course_id, sec_id, semester, year*)



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* in *instructor* replicates information present in the relationship and is therefore redundant
- When converting back to tables, in some cases (e.g., when each instructor has exactly one associated department) the attribute gets reintroduced, as we will see later



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

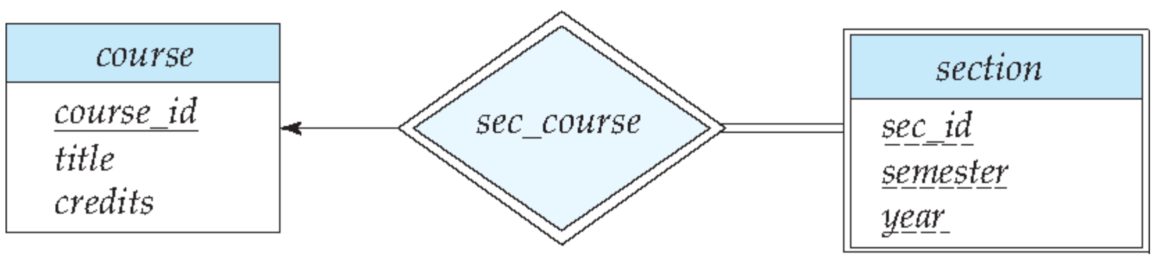
- A strong entity set reduces to a schema with the same attributes

  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, semester, year)

- Example



Representation of Entity Sets with Composite Attributes

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

- 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)

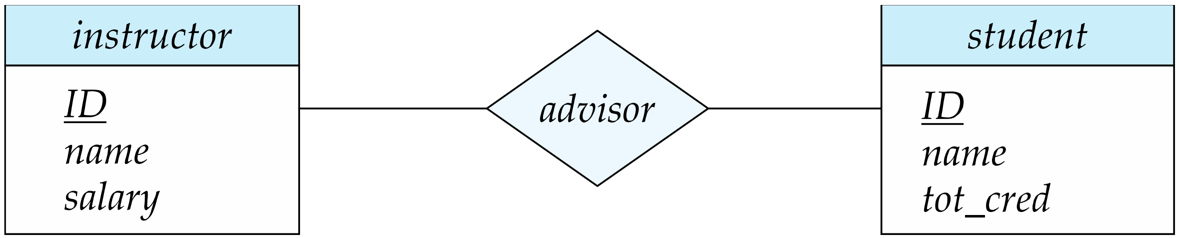
Representation of Entity Sets with Multivalued Attributes

- 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 = (*ID*, *phone_number*)
- Each value of the multivalued attribute maps to a separate tuple of the relation on schema *EM*
  - For example, an *instructor* entity with primary key 22222 and phone numbers 456-7890 and 123-4567 maps to two tuples:
    (22222, 456-7890) and (22222, 123-4567)

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*

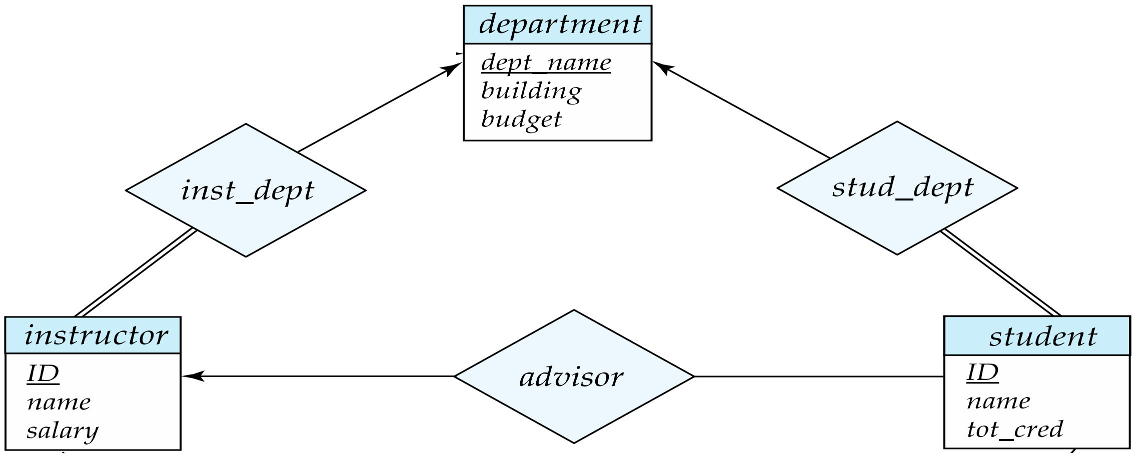
advisor = (*s_id, i_id*)



Combination 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*
- 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

Combination of Schemas



Inst_dept

- The schema *inst_dept* can be combined with the *instructor* schema - the resulting *instructor* schema consists of the attributes {*ID*, *name*, *dept_name*, *salary*}

stud_dept

- The schema *stud_dept* can be combined with the *student* schema - the resulting *student* schema consists of the attributes {*ID*, *name*, *dept_name*, *tot_cred*}

Combination of Schemas

- As we saw, a weak entity set becomes a table that includes a column for the primary key of the identifying strong entity set - the resulting *section* schema consists of the attributes

  {*course_id*, *sec_id*, *semester*, *year*}

- *sec_class*. The schema *sec_class* can be combined with the *section* schema obtained above - the resulting *section* schema now consists of the attributes

  {*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*}

- The classroom table is still needed to store classroom info (such as capacity), which is necessary, since even if there is no class held in a classroom, its capacity info is still available