Principles of Database Systems (CS307)

Lecture 7-8: Application Development; Database Design Using the E-R Model

Yuxin Ma

Department of Computer Science and Engineering
Southern University of Science and Technology

- Most contents are from slides made by Stéphane Faroult and the authors of Database System Concepts (7th Edition).
- Their original slides have been modified to adapt to the schedule of CS307 at SUSTech.

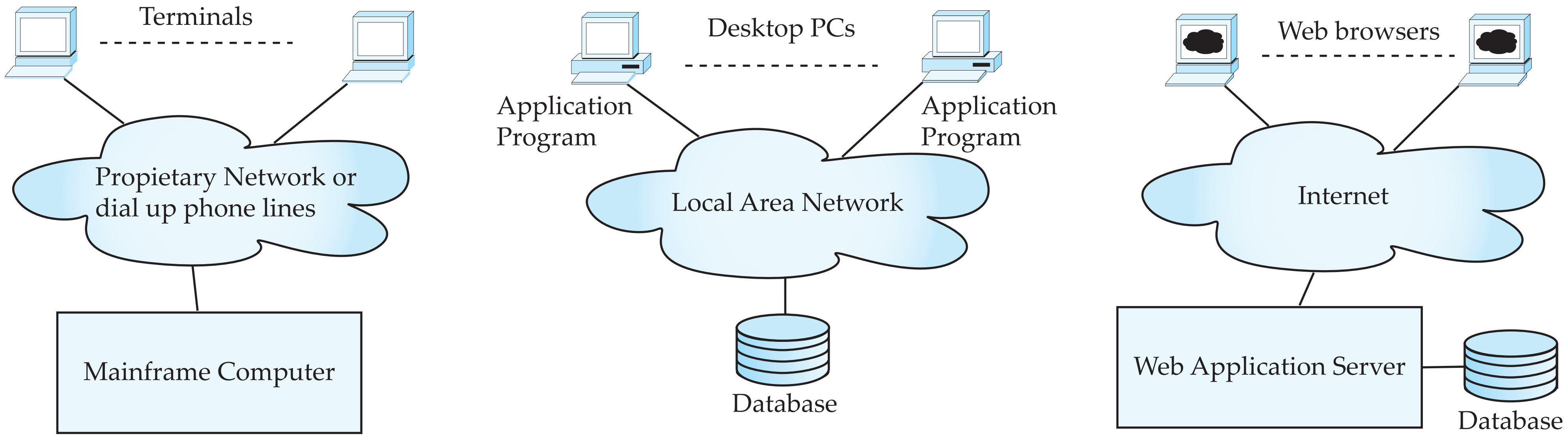
Application Development

Application Programs and User Interfaces

- Most database users do *not* use a query language like SQL
- An application program acts as the intermediary between users and the database
  - Applications split into
    - front-end
    - middle layer
    - backend
- Front-end: user interface
  - Forms
  - Graphical user interfaces
  - Many interfaces are Web-based

Application Architecture Evolution

- Three distinct era's of application architecture
  - Mainframe (1960' s and 70' s)
  - Personal computer era (1980' s)
  - Web era (mid 1990' s onwards)
  - Web and Smartphone era (2010 onwards)



(a) Mainframe Era (b) Personal Computer Era (c) Web era

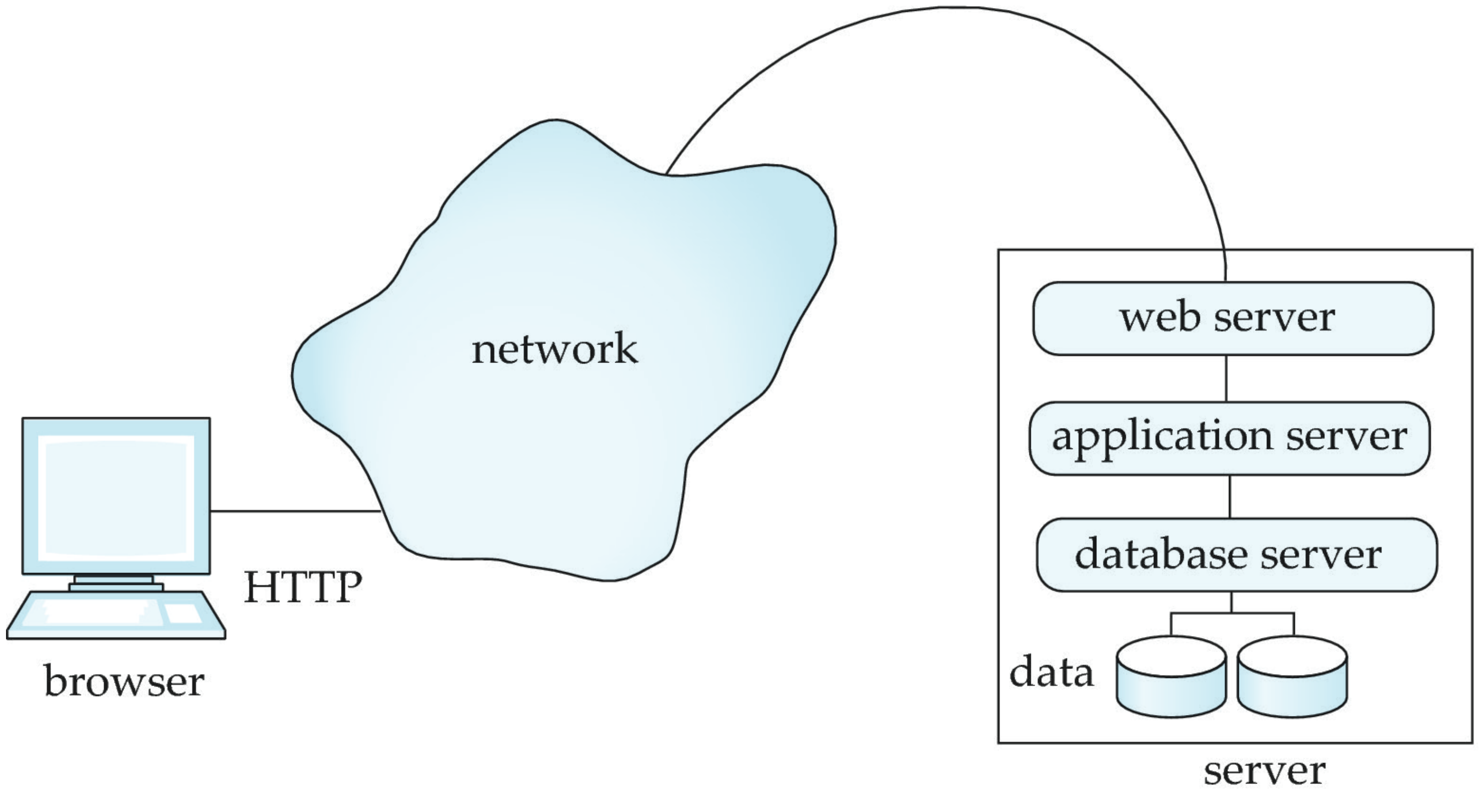
Web Interface

- Web browsers have become the de-facto standard user interface to databases
  - Enable large numbers of users to access databases from anywhere
  - Avoid the need for downloading/installing specialized code, while providing a good graphical user interface
    - JavaScript, ~~Flash~~ and other scripting languages run in browser, but are downloaded transparently
  - Examples: banks, airline and rental car reservations, university course registration and grading, and so on.

The World Wide Web

- The Web is a distributed information system based on hypertext.
- Most Web documents are hypertext documents formatted via the HyperText Markup Language (HTML)
- HTML documents contain
  - text along with font specifications, and other formatting instructions
  - hypertext links to other documents, which can be associated with regions of the text.
  - forms, enabling users to enter data which can then be sent back to the Web server

Three-Layer Web Architecture



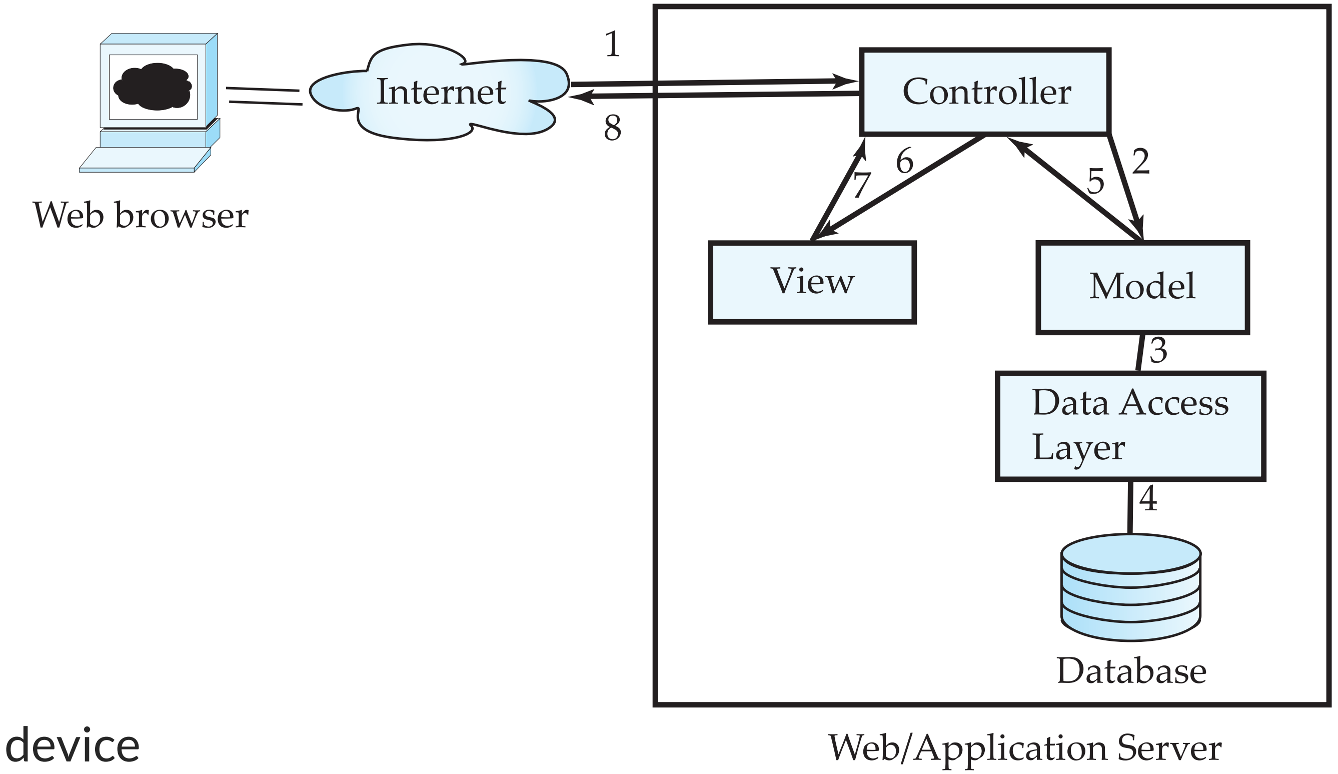
HTML and HTTP

- HTML provides formatting, hypertext link, and image display features
  - including tables, stylesheets (to alter default formatting), etc.
- HTML also provides input features
  - Select from a set of options
    - Pop-up menus, radio buttons, check lists
  - Enter values
    - Text boxes
  - Filled in input sent back to the server, to be acted upon by an executable at the server
- HyperText Transfer Protocol (HTTP) used for communication with the Web server

JavaScript

- JavaScript very widely used
  - Forms basis of new generation of Web applications (called Web 2.0 applications) offering rich user interfaces
- JavaScript functions can
  - Check input for validity
  - Modify the displayed Web page, by altering the underling document object model (DOM) tree representation of the displayed HTML text
    - Communicate with a Web server to fetch data and modify the current page using fetched data, without needing to reload/refresh the page
    - Forms basis of AJAX technology used widely in Web 2.0 applications
    - E.g. on selecting a country in a drop-down menu, the list of states in that country is automatically populated in a linked drop-down menu

Application Architectures



- Application layers
  - Presentation or user interface
    - model-view-controller (MVC) architecture
      - model: business logic
      - view: presentation of data, depends on display device
      - controller: receives events, executes actions, and returns a view to the user
  - business-logic layer
    - provides high level view of data and actions on data
      - often using an object data model
    - hides details of data storage schema
  - data access layer
    - interfaces between business logic layer and the underlying database
    - provides mapping from object model of business layer to relational model of database

Business Logic Layer

- Provides abstractions of entities
  - E.g., students, instructors, courses, etc
- Enforces business rules for carrying out actions
  - E.g., student can enroll in a class only if she has completed prerequsites, and has paid her tuition fees
- Supports workflows which define how a task involving multiple participants is to be carried out
  - E.g., how to process application by a student applying to a university
  - Sequence of steps to carry out task
  - Error handling
    - E.g. what to do if recommendation letters not received on time

Object-Relational Mapping

- Allows application code to be written on top of object-oriented data model, while storing data in a traditional relational database
  - Alternative: implement object-oriented or object-relational database to store object model
    - Has not been commercially successful
- Schema designer must provide a mapping between object data and relational schema
  - E.g., Java class Student mapped to relation student, with corresponding mapping of attributes
  - An object can map to multiple tuples in multiple relations
- Application opens a session, which connects to the database
- Objects can be created and saved to the database using session.save(object)
  - Mapping used to create appropriate tuples in the database
- Query can be run to retrieve objects satisfying specified predicates

Web Services

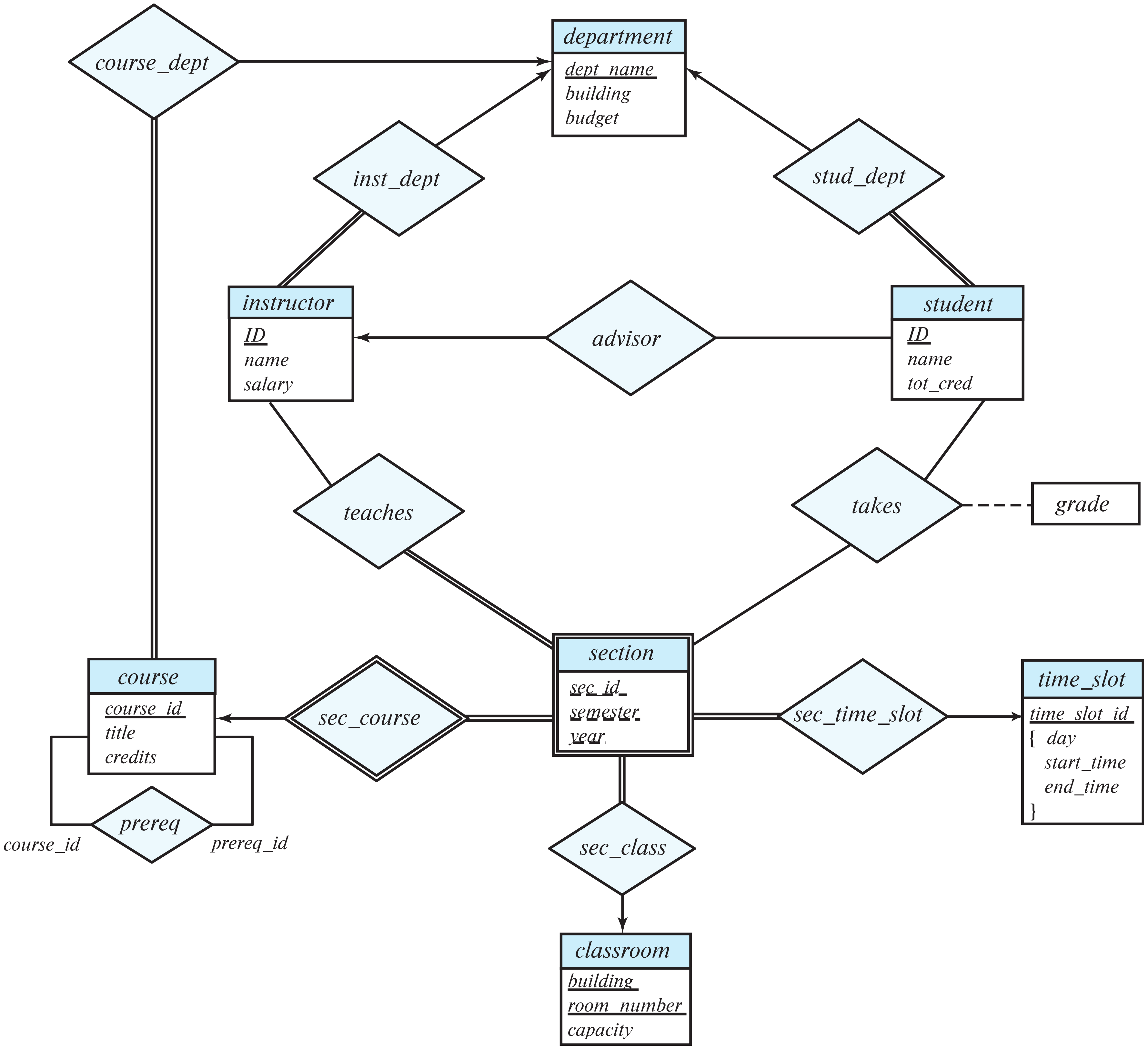
- Allow data on Web to be accessed using remote procedure call mechanism
- Two approaches are widely used
  - Representation State Transfer (REST): allows use of standard HTTP request to a URL to execute a request and return data
    - Returned data is encoded either in XML, or in JavaScript Object Notation (JSON)
  - Big Web Services:
    - Uses XML representation for sending request data, as well as for returning results
    - Standard protocol layer built on top of HTTP

Self Study

- Key points:
  - Techniques / libraries / APIs to connect to a database (e.g. PostgreSQL) and run SQL querys in a program
    - ODBC, JDBC?
    - SQLAlchemy? Spring Boot? Hibernate?
  - (Web) Backend Frameworks
    - Spring Boot, Flask, Django
    - NodeJS
  - Frontend Frameworks
    - React, Vue

Entity-Relationship Model (E-R Model)
Entity-Relationship Diagram (E-R Diagram)

The New Running Example



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

- 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 many good designs from which we must choose

Design Approaches

- Entity Relationship Model (covered in this chapter)
  - Models an enterprise as <u>a collection of</u> 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 (E-R diagram)

- Normalization Theory (coming in the next few weeks)
  - Formalize what designs are bad, and test for them

Entity Sets

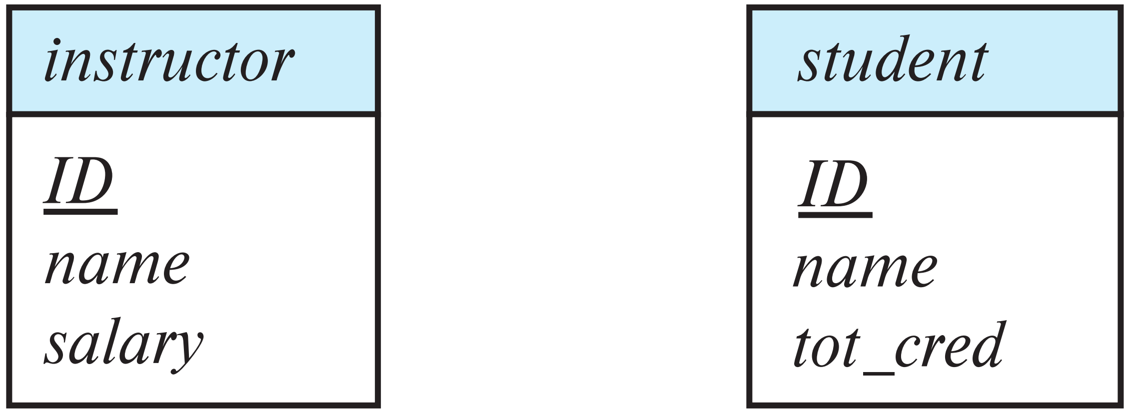
- An entity is an object that exists and is distinguishable from other objects
  - Example: specific person, company, event, plant
- An entity set is a set of entities of the same type that share the same properties
  - Example: set of all persons, companies, trees, holidays
- An entity is represented 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

```
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

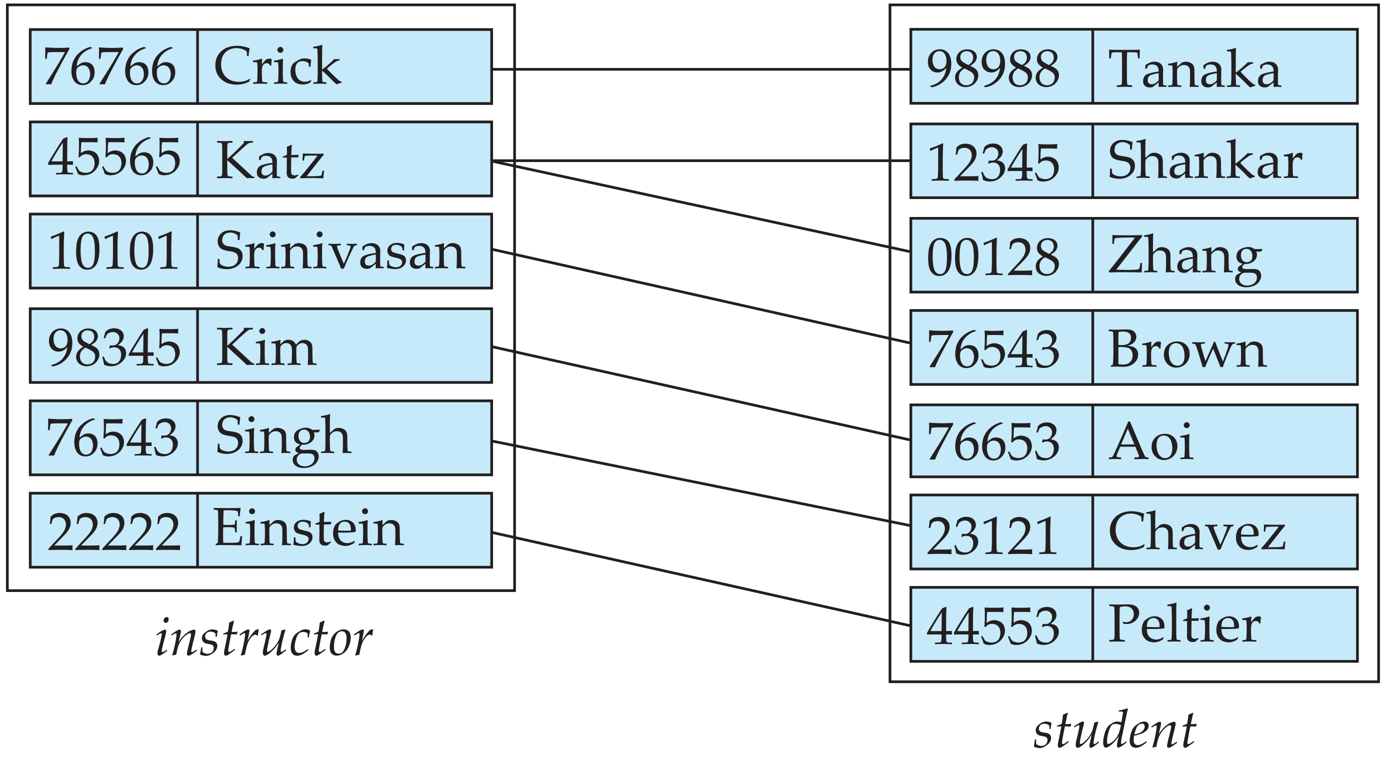
$$\{(e_1, e_2, \ldots e_n) \mid e_1 \in E_1, e_2 \in E_2, \ldots, e_n \in E_n\}$$

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

- Example: $(44553, 22222) \in advisor$

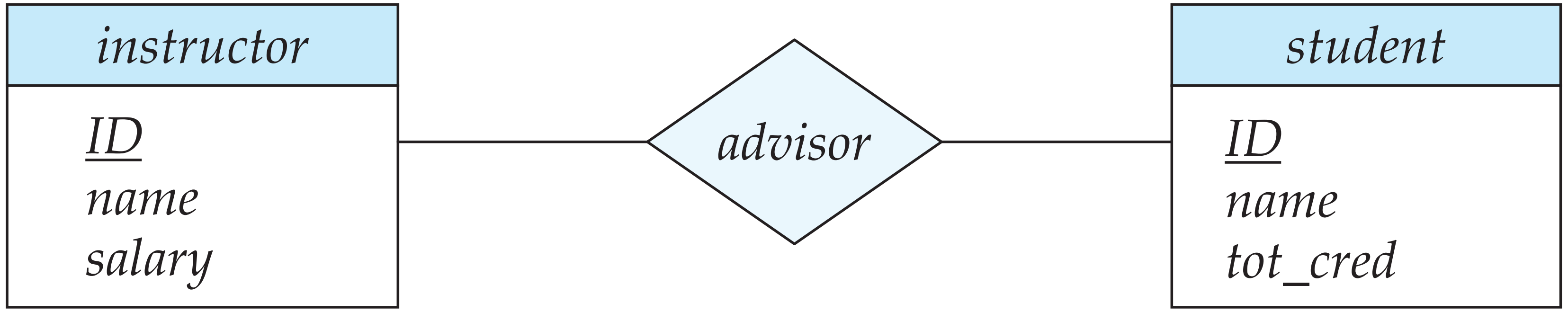
Relationship Sets

- 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



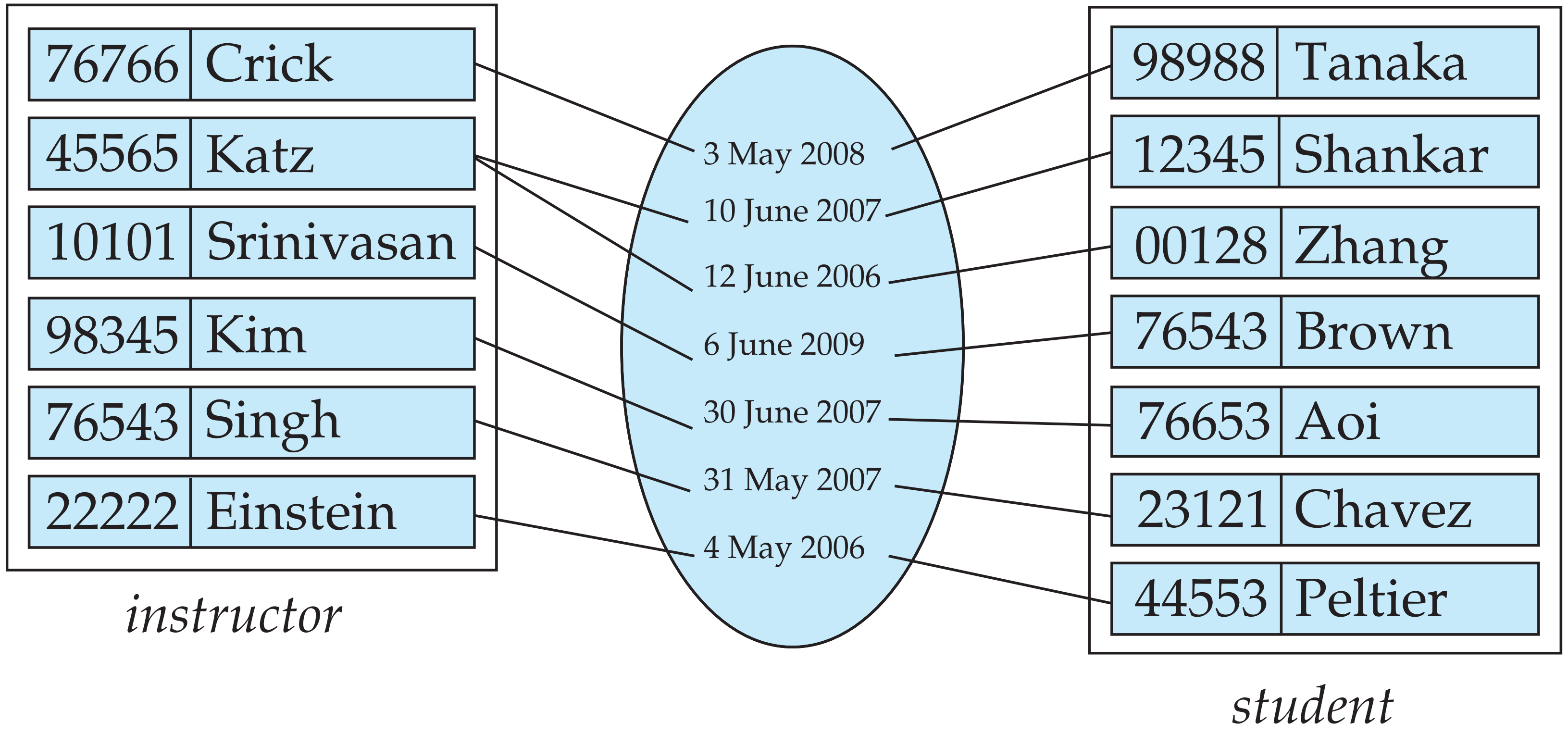
Representing Relationship Sets via E-R Diagrams

- Diamonds represent relationship sets

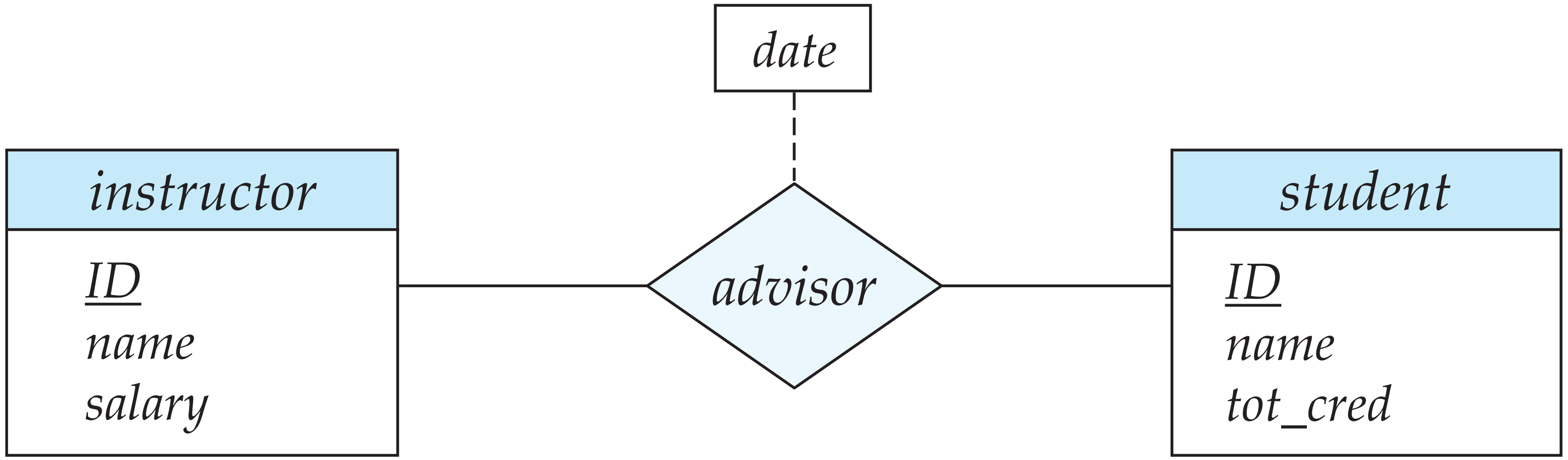


Relationship Sets (Cont.)

- 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



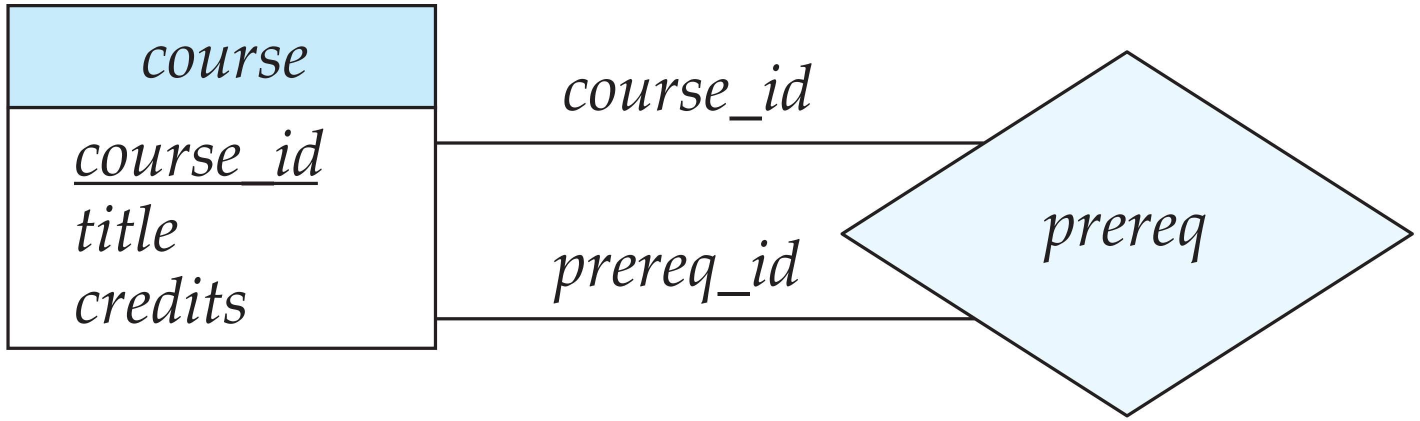
Relationship Sets with Attributes



Roles

- Entity sets of a relationship need not be distinct
  - That is to say, we can create self-pointing relationships for an entity set
  - Each occurrence of an entity set plays a "role" in the relationship

  - Example: A relationship set to represent the prerequisites of a course
    - E.g., Data Structure depends on Introduction to Programming
    - The labels "`course_id`" and "`prereq_id`" are called roles

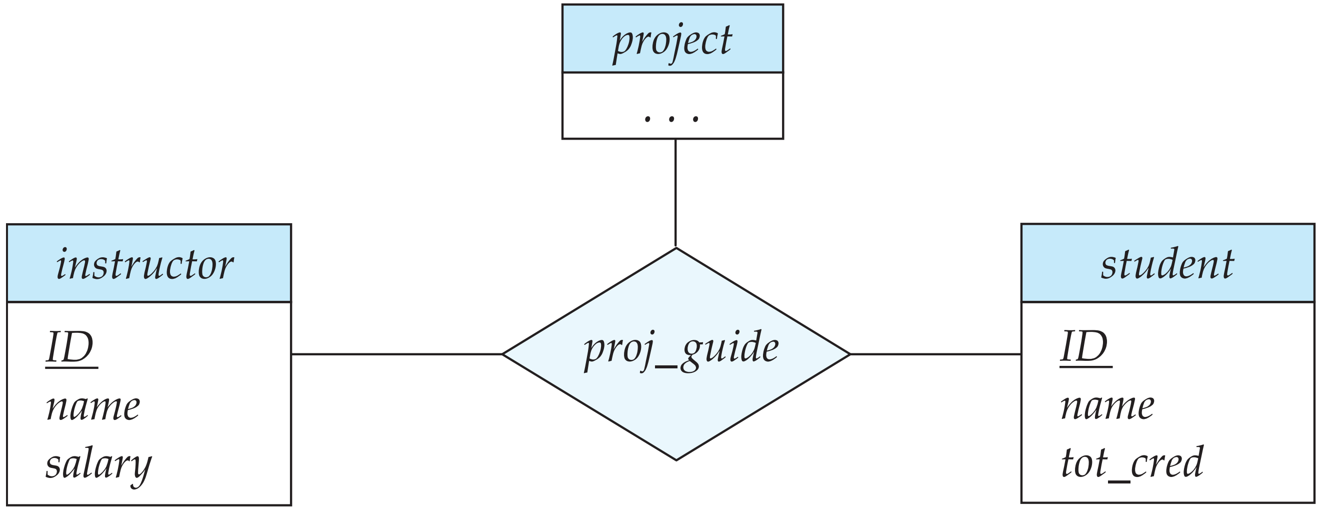


Degree of a Relationship Set

- 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
  - 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



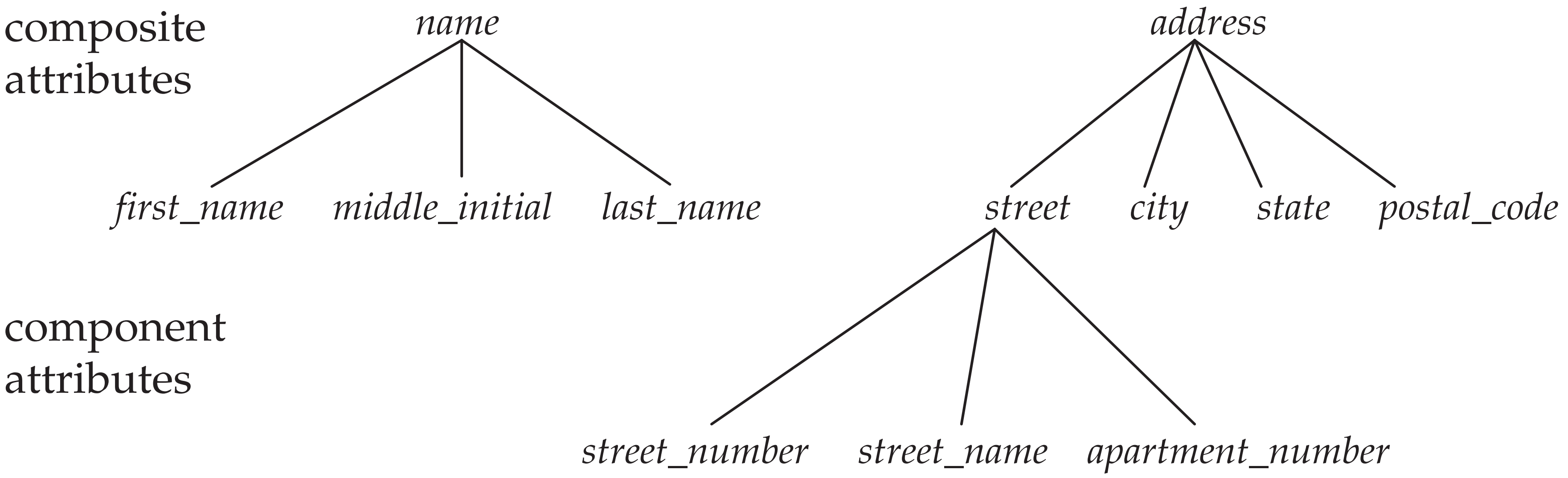
Complex Attributes

- Attribute types:
  - Simple and composite attributes.
  - Single-valued and multivalued attributes
    - Example: multivalued attribute: phone_numbers
      - A person can have 1 or more phone numbers at the same time
  - 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 divided attributes into subparts (other attributes)
  - Sometimes we may only use part of the attributes, where the composite attribute is a good design choice

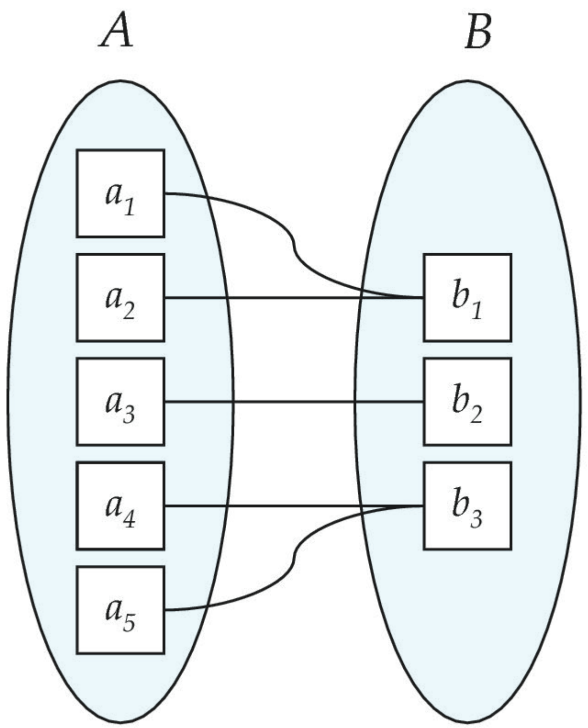


instructor
<u>ID</u> *name*   *first_name*   *middle_initial*   *last_name* *address*   *street*   *street_number*   *street_name*   *apt_number*   *city*   *state*   *zip* *{ phone_number }* *date_of_birth* *age ( )*

Mapping Cardinality Constraints

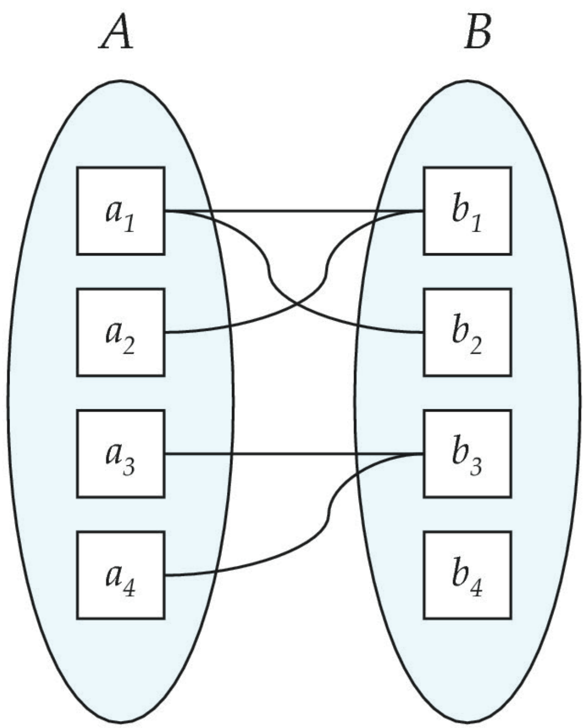
- Mapping Cardinality（映射基数）
  - Express the number of entities to which another entity can be associated via a relationship set.
    - Most useful in describing binary relationship sets
- For a binary relationship set, the mapping cardinality must be one of the following types:
  - One to one
  - One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

Mapping Cardinalities



(a)

Many to one

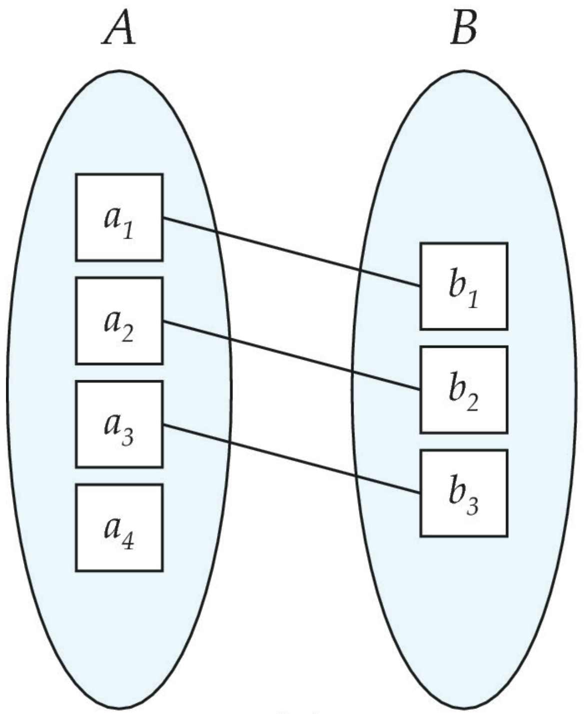


(b)

Many to many

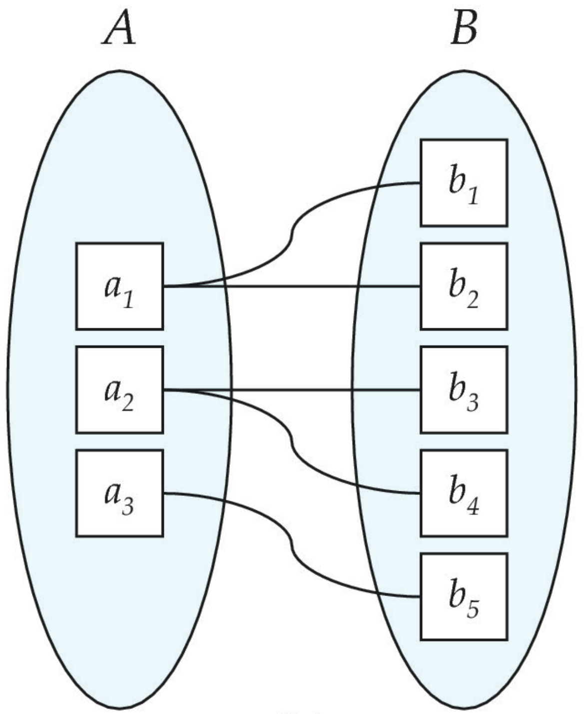
Note: Some elements in *A* and *B* may not be mapped to any elements in the other set

Mapping Cardinalities



(a)

One to one



(b)

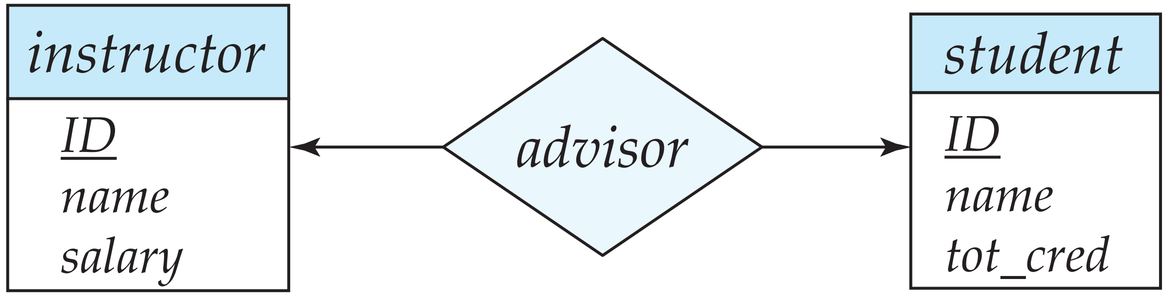
One to many

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

Representing Cardinality Constraints in ER Diagram

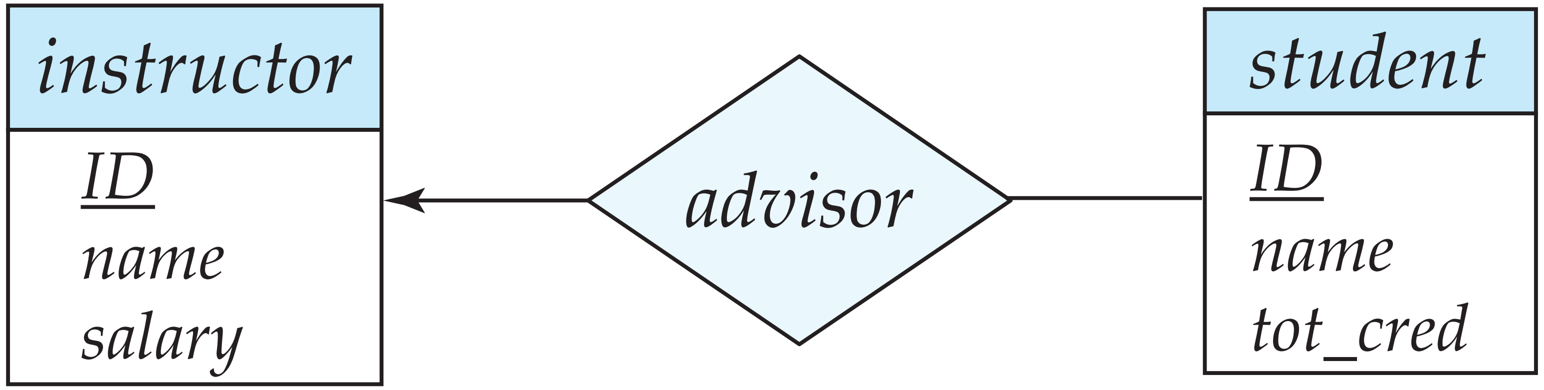
- We express cardinality constraints by:
  - drawing either a directed line (→), signifying "one,"
  - or an undirected line (—), signifying "many,"
- … between the relationship set and the entity set.

- <u>One-to-one relationship</u> between an instructor and a student :
  - A student is associated with at most one instructor via the relationship advisor



Representing Cardinality Constraints in ER Diagram

- 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

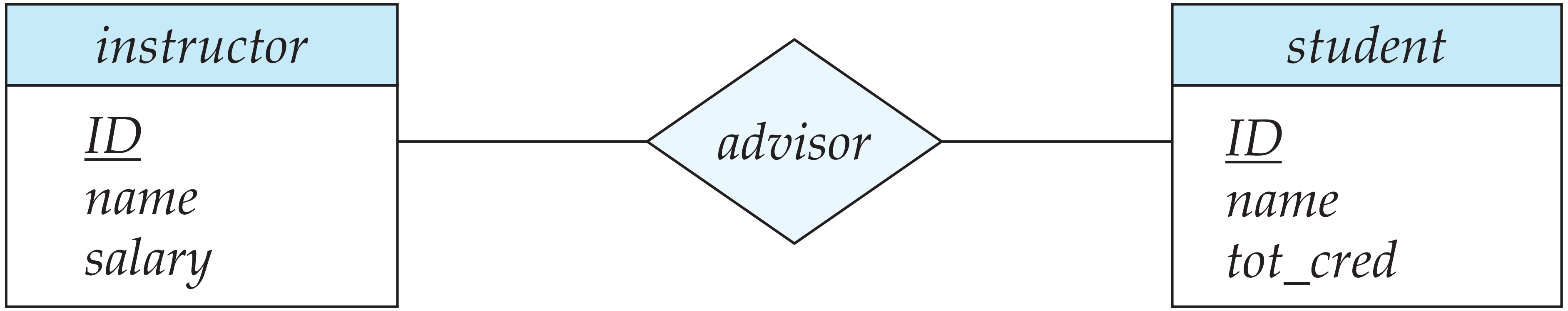


Representing Cardinality Constraints in ER Diagram

- In a <u>many-to-one relationship</u> 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

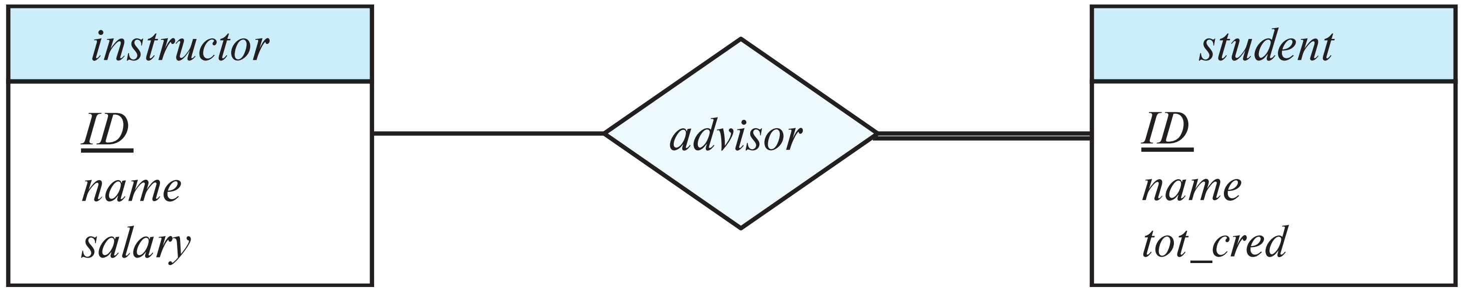
Representing Cardinality Constraints in ER Diagram

- 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



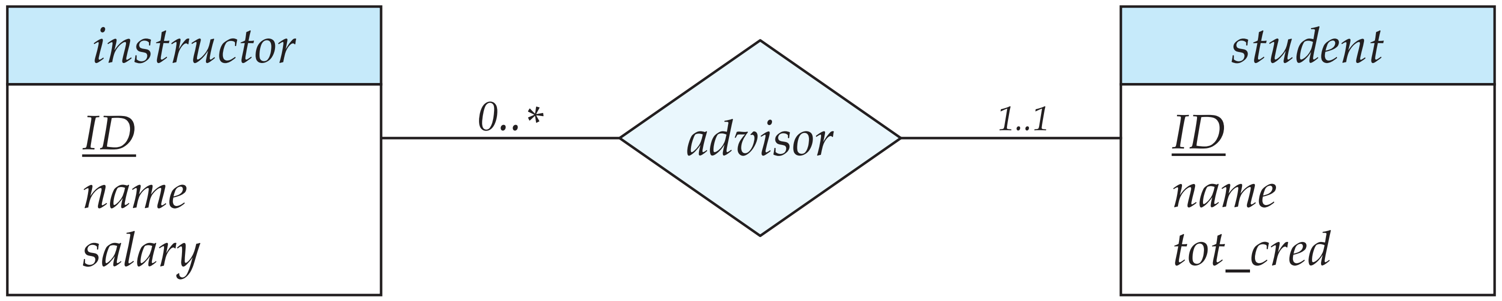
Total and Partial Participation

- Total participation (indicated by *double line*)
  - Every entity in the entity set participates in at least one relationship in the relationship set
  - Example: Participation of student in advisor relation is total
    - i.e., every student must have an associated instructor
- Partial participation
  - Some entities may not participate in any relationship in the relationship set
  - 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

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 <u>suffice to distinguish entities from each other</u>

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 E1, E2, .. En
  - The primary key for R consists of the union of the primary keys of entity sets E1, E2, ..En
  - If the relationship set R has attributes $a_1, a_2, .., a_m$ associated with it, the primary key of R also includes the attributes $a_1, a_2, .., a_m$
- 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.

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-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.

* *K* is a **superkey** of *R* if values for *K* are sufficient to identify a unique tuple of each possible relation *r(R)*

  Example: {*ID*} and {ID,name} are both superkeys of *instructor*.

Choice of Primary key for Binary Relationship

- 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.

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

- 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

- 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

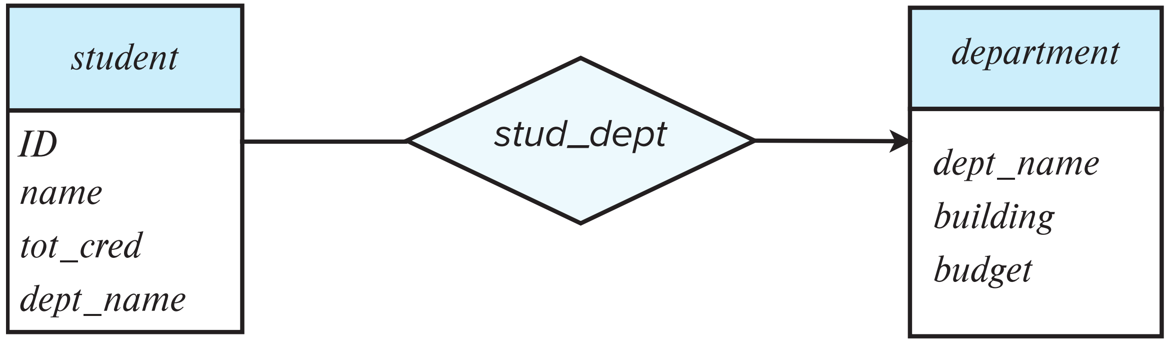
- Note that the relational schema we eventually create from the entity set section does have the attribute course_id, for reasons that will become clear later, even though we have dropped the attribute course_id from the entity set section.

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.



(a) Incorrect use of attribute

- BUT: when converting back to tables, in some cases the attribute gets reintroduced.

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

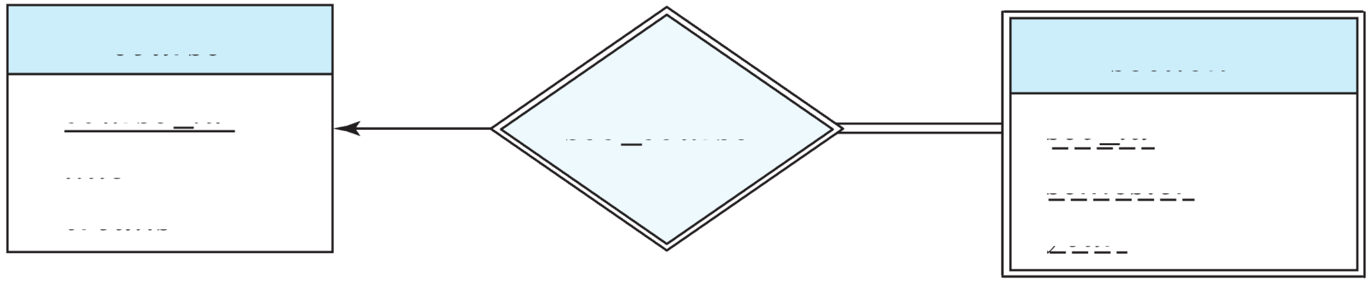
- A strong entity set reduces to a schema with the same attributes

  student(<u>ID</u>, 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 (<u>course_id, sec_id, sem, year</u>)

- Example



Representation of Entity Sets with Composite Attributes

- Composite attributes are flattened out by creating a separate attribute for each component attribute
  - Example: given entity set instructor with composite attribute name with component attributes first_name and last_name the schema corresponding to the entity set has two attributes name_first_name and name_last_name
    - Prefix omitted if there is no ambiguity (name_first_name could be first_name)
- Ignoring multivalued attributes, extended instructor schema is
  - instructor(ID,
    first_name, middle_initial, last_name,
    street_number, street_name,
    apt_number, city, state, zip_code,
    date_of_birth)

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

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****)*

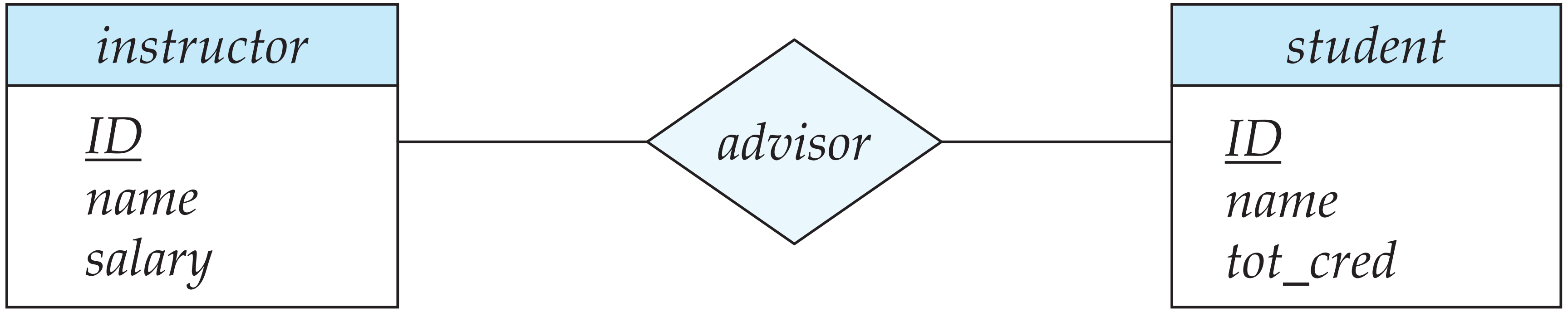
  - Each value of the multivalued attribute maps to a separate tuple of the relation on schema EM
  - For example, an instructor entity with primary key 22222 and phone numbers 456-7890 and 123-4567 maps to two tuples:

    (22222, 456-7890) and *(22222, 123-4567)*

Representing Relationship Sets

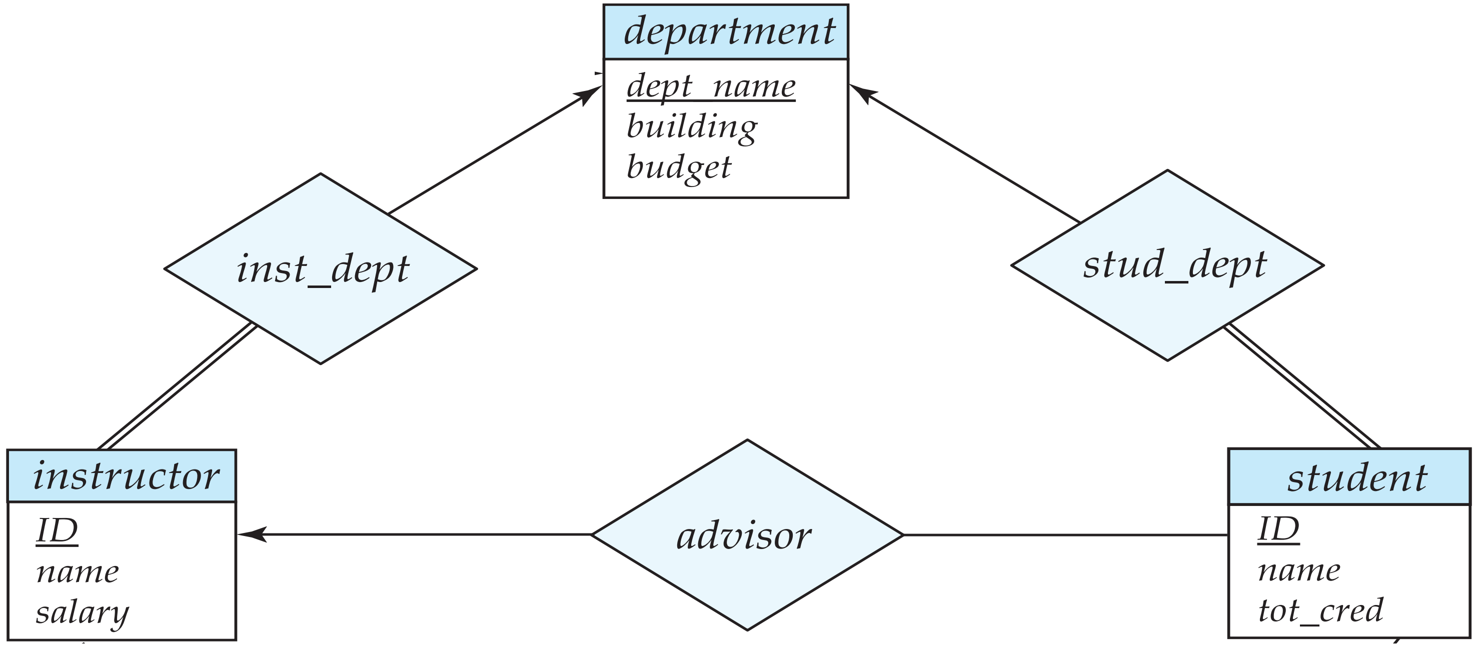
- A many-to-many relationship set is represented as a schema with attributes for the primary keys of the two participating entity sets, and any descriptive attributes of the relationship set.
  - Example: schema for relationship set advisor

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



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



Redundancy of Schemas

- 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

- 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