Principles of Database Systems (CS307)

Lecturer's Cut: Relational Database Design

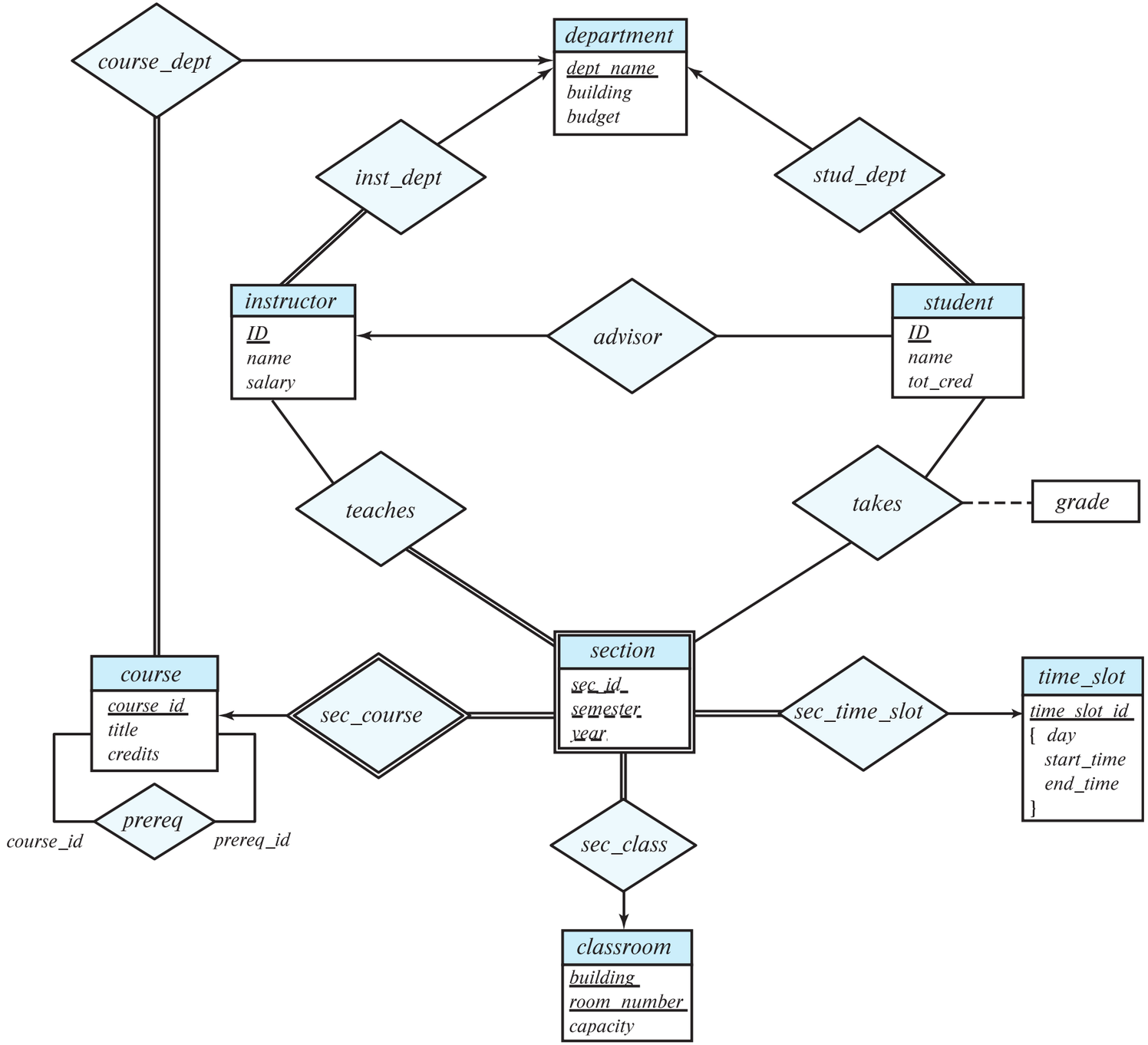
Yuxin Ma

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.

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 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 (E-R diagram)

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

Entity Sets

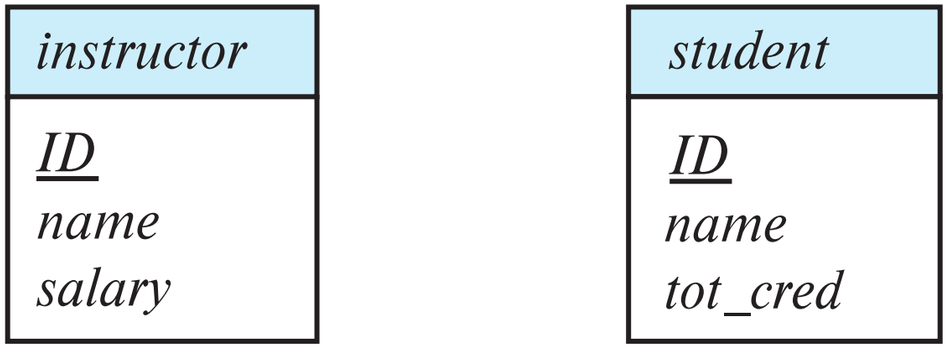
- An entity is an object that exists and is distinguishable from other objects
  - Example: specific person, company, event, plant
- An entity set is a set of entities of the same type that share the same properties
  - Example: set of all persons, companies, trees, holidays
- An entity is represented 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 ≥ 2 entities, each taken from entity sets

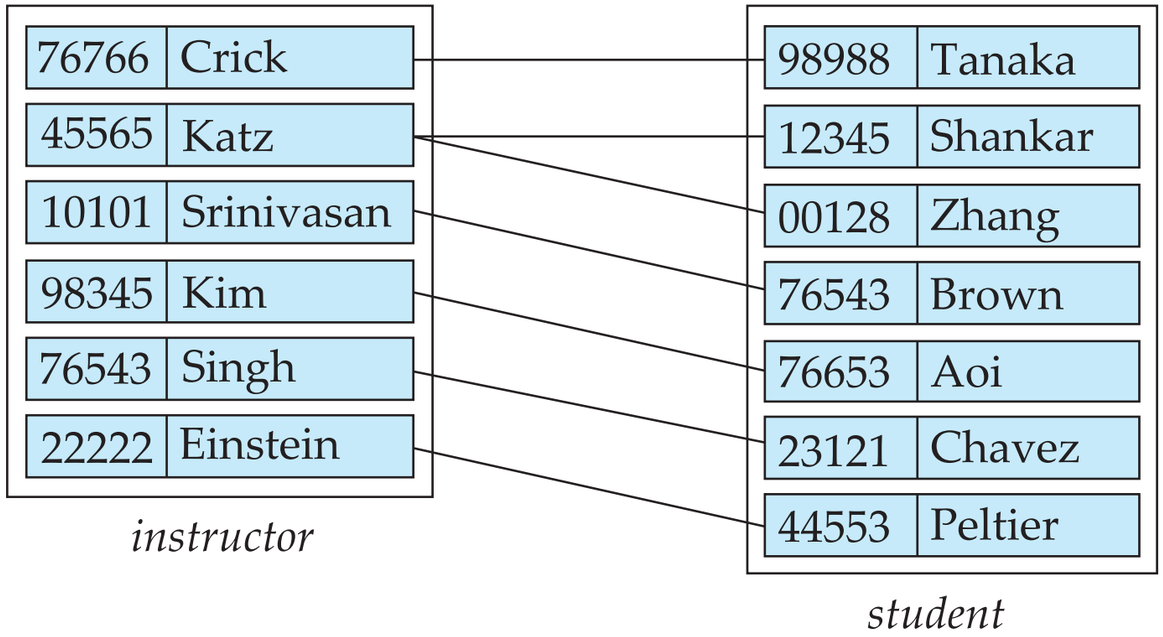
$$\{(e_1, e_2, \ldots e_n) \mid e_1 \in E_1, e_2 \in E_2, \ldots, e_n \in E_n\}$$

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

  - Example: (44553,22222) ∈ advisor

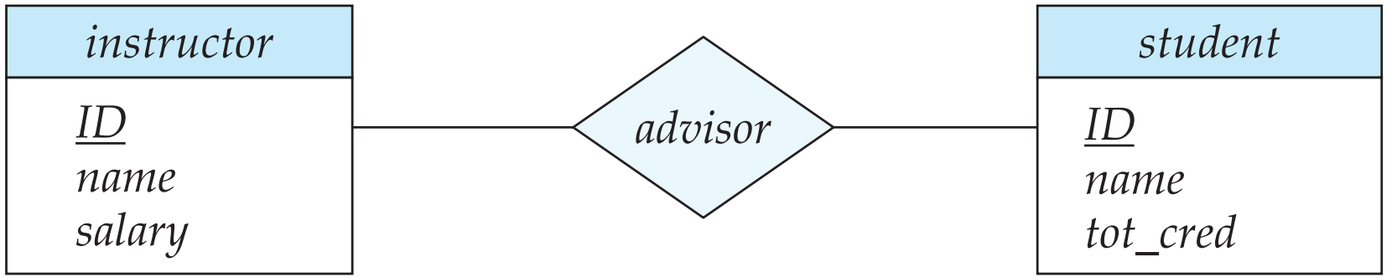
Relationship Sets

- 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



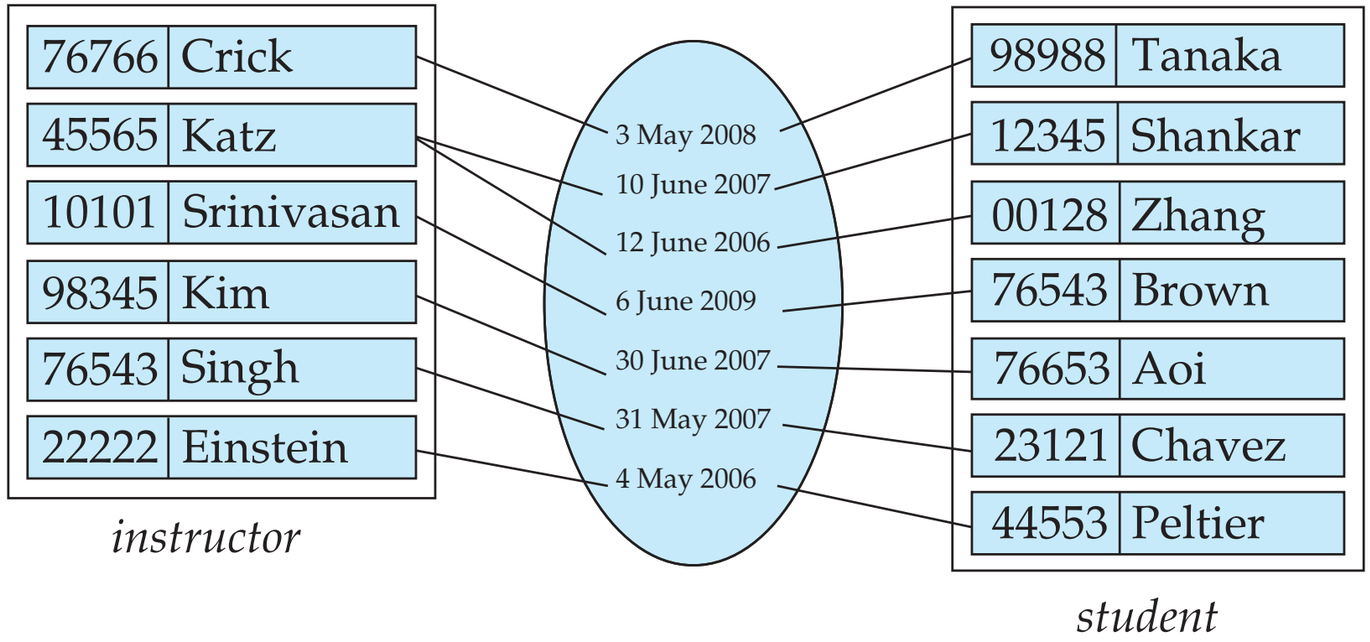
Representing Relationship Sets via E-R Diagrams

- Diamonds represent relationship sets

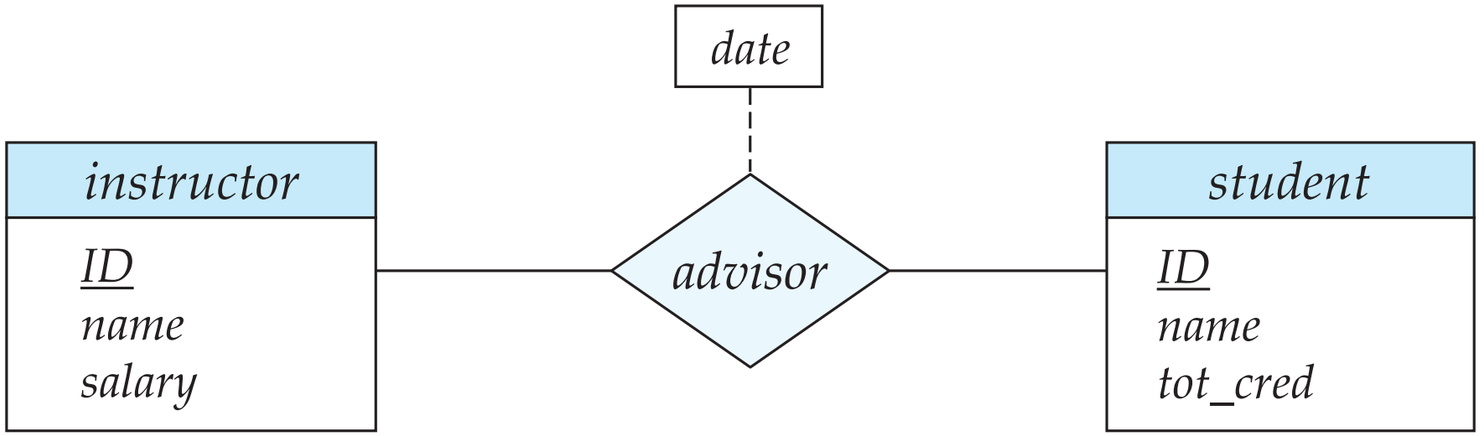


Relationship Sets (Cont.)

- 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



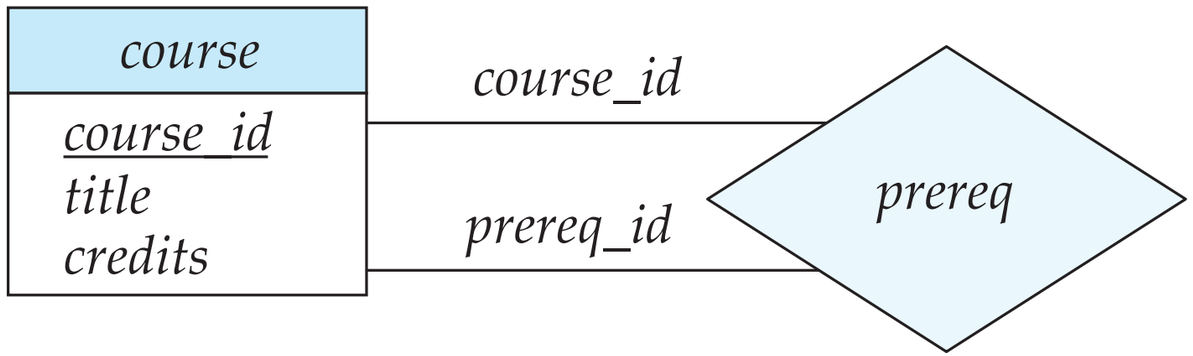
Relationship Sets with Attributes



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

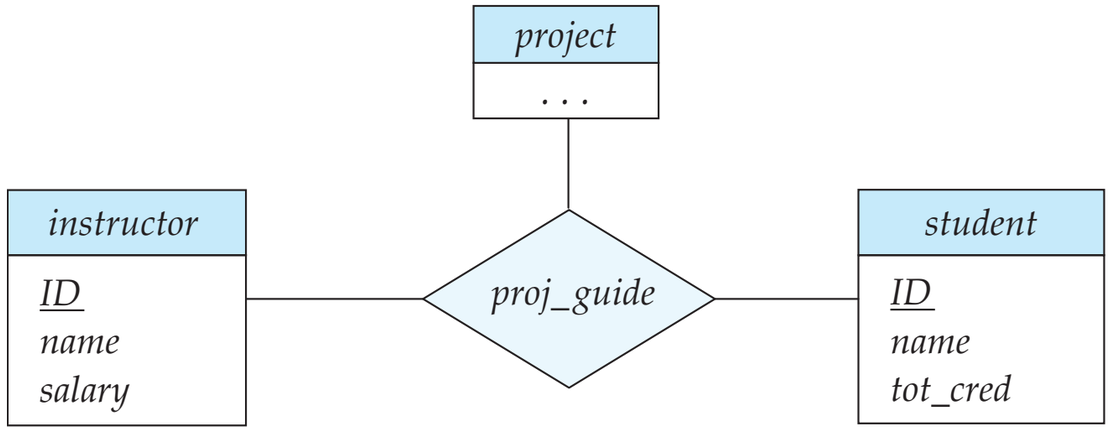


Degree of a Relationship Set

- 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



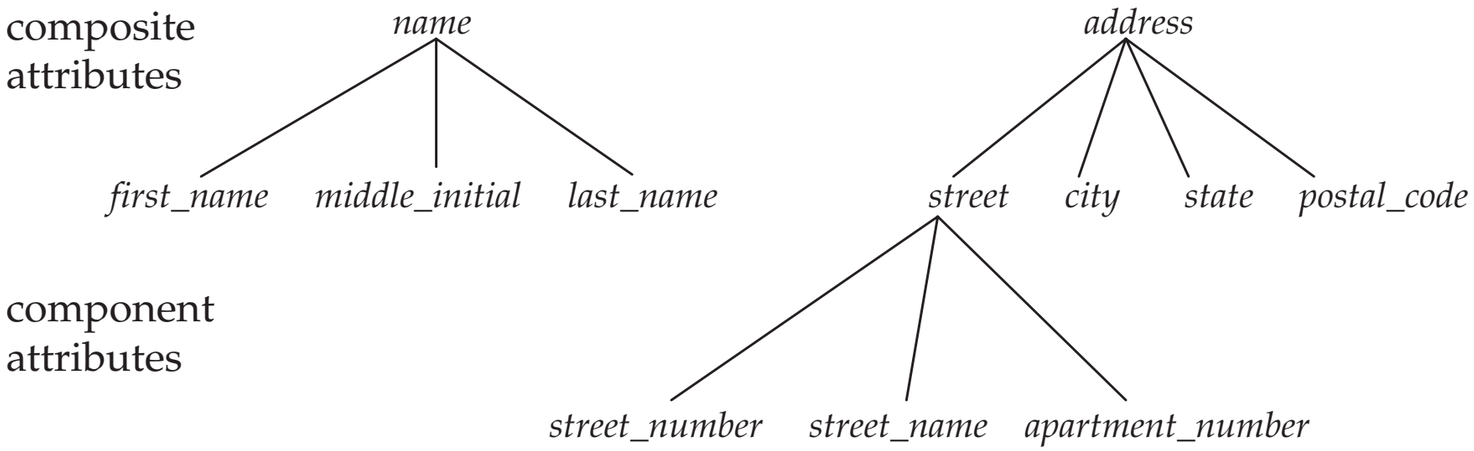
Complex Attributes

- 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



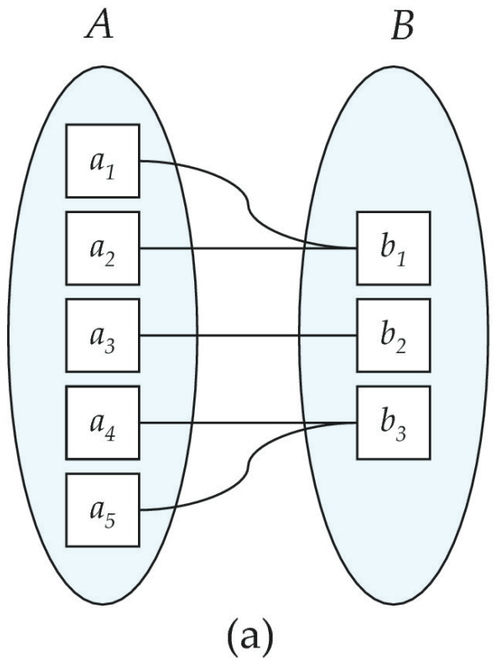
instructor

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 ( )

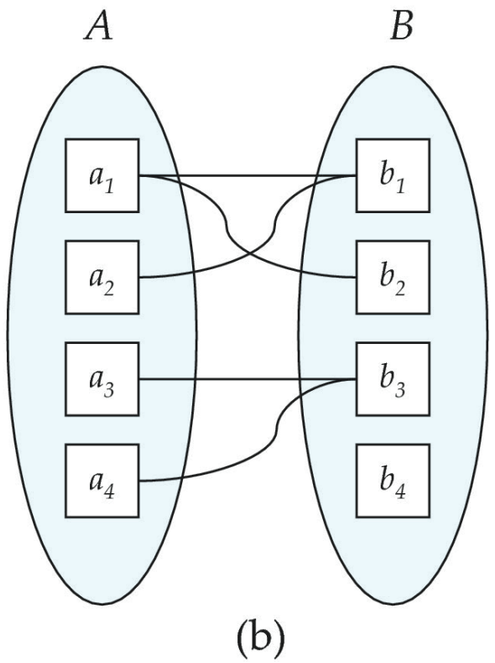
Mapping Cardinality Constraints

- 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



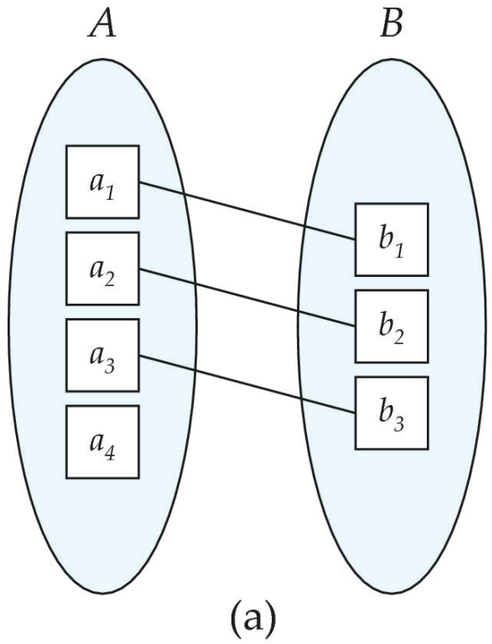
Many to one



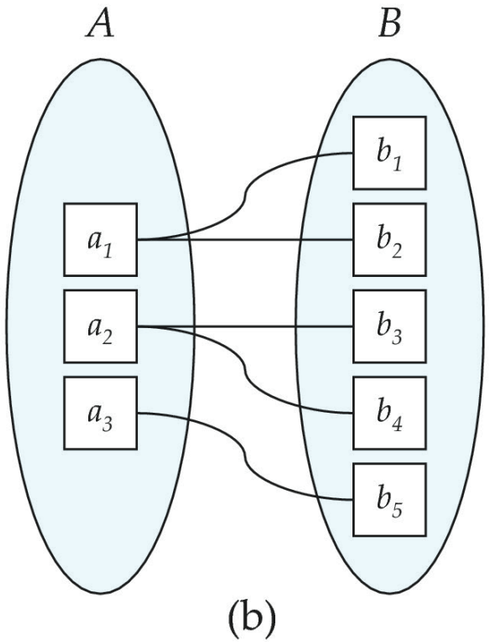
Many to many

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

Mapping Cardinalities



One to one



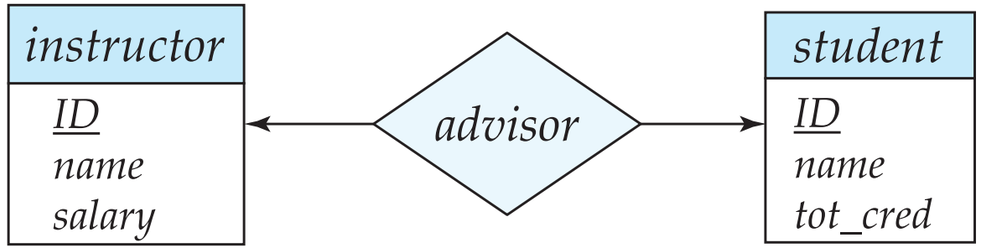
One to many

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

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 :
  - A student is associated with at most one instructor via the relationship advisor



Representing Cardinality Constraints in ER Diagram

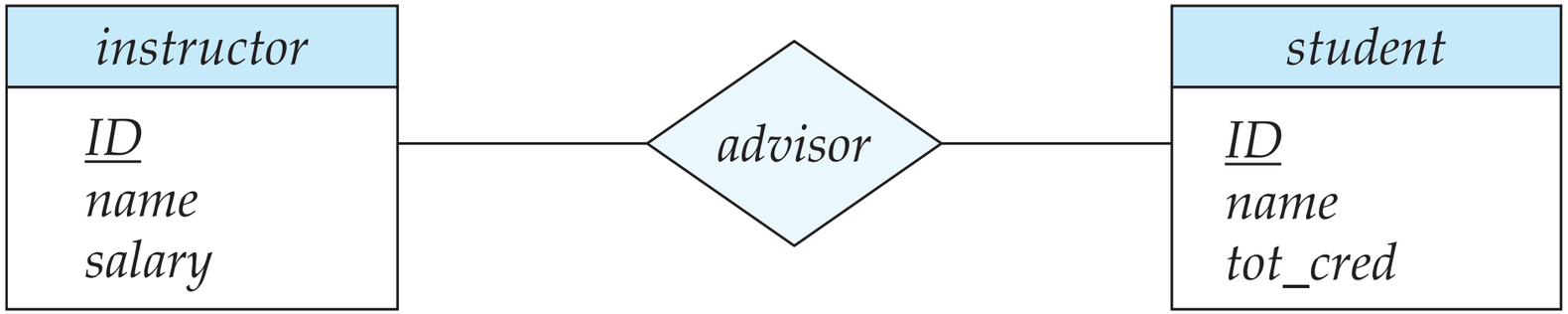
- One-to-many relationship between an instructor and a student
  - an instructor is associated with several (including 0) students via advisor
  - a student is associated with at most one instructor via advisor

Representing Cardinality Constraints in ER Diagram

- In a 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

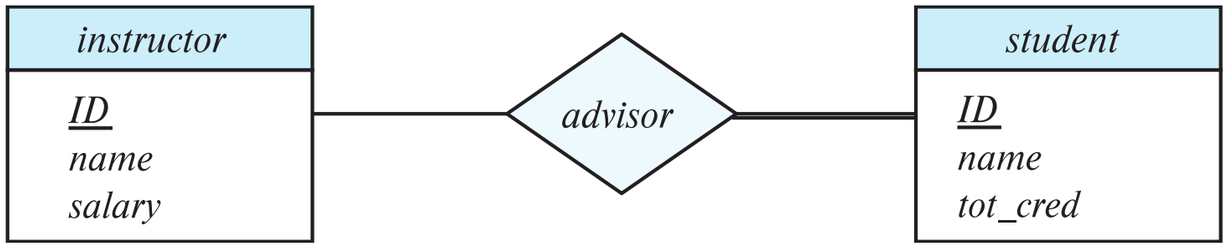
Representing Cardinality Constraints in ER Diagram

- 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



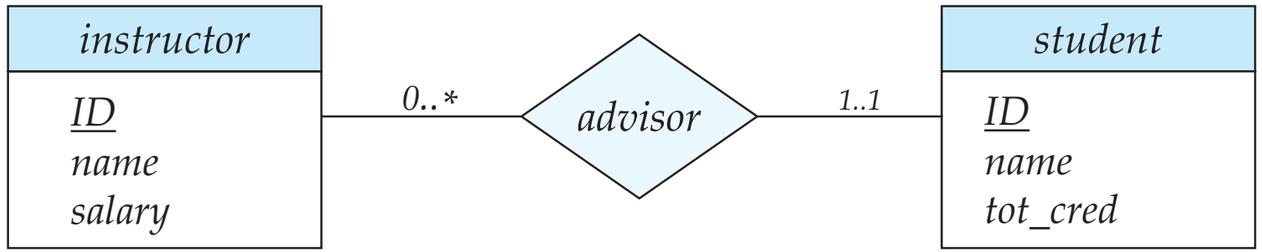
Total and Partial Participation

- 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 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 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

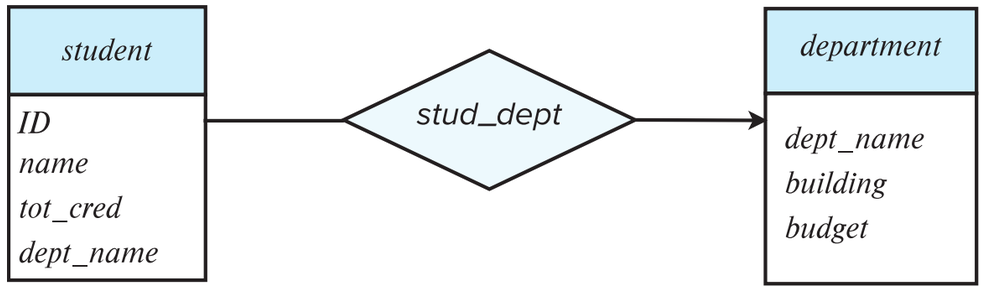
- Note that the relational schema we eventually create from the entity set section does have the attribute course_id, for reasons that will become clear later, even though we have dropped the attribute course_id from the entity set section.

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

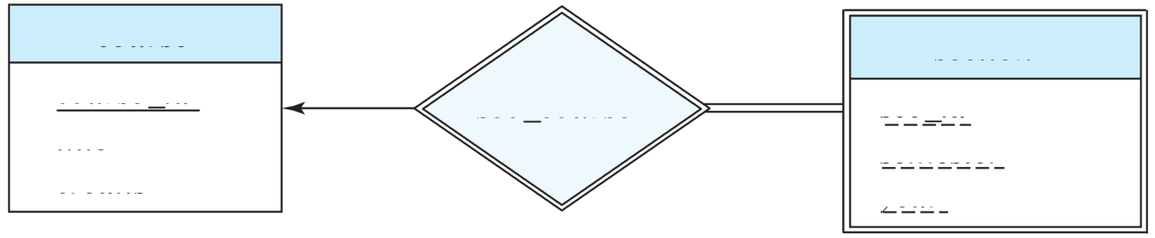
- A strong entity set reduces to a schema with the same attributes

  student(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

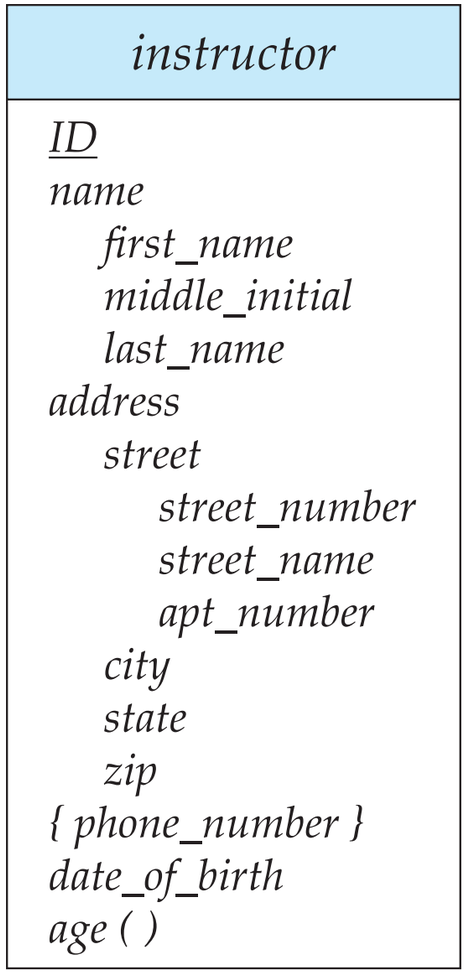
  section (course_id, sec_id, sem, year)

- Example



Representation of Entity Sets with Composite Attributes

- 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:

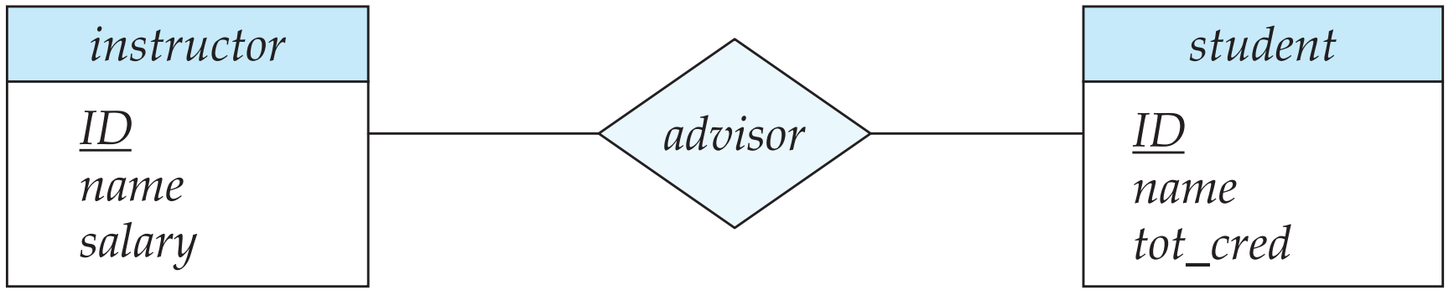
    inst_phone= (ID, phone_number)
  - Each value of the multivalued attribute maps to a separate tuple of the relation on schema EM
  - For example, an instructor entity with primary key 22222 and phone numbers 456-7890 and 123-4567 maps to two tuples:

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

Representing Relationship Sets

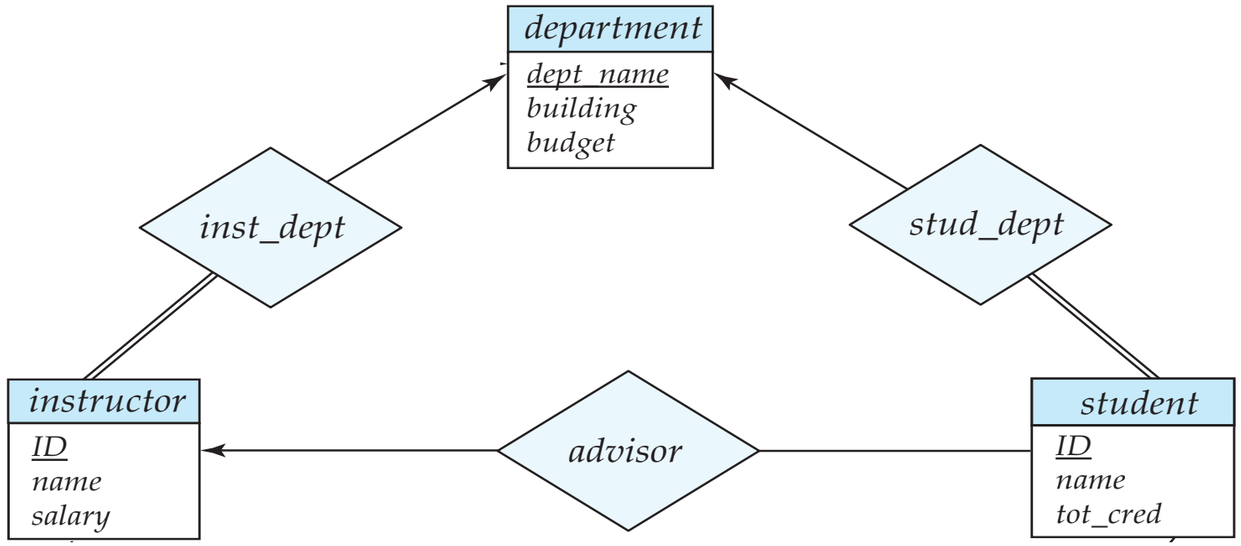
- A many-to-many relationship set is represented as a schema with attributes for the primary keys of the two participating entity sets, and any descriptive attributes of the relationship set.
  - Example: schema for relationship set advisor

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



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

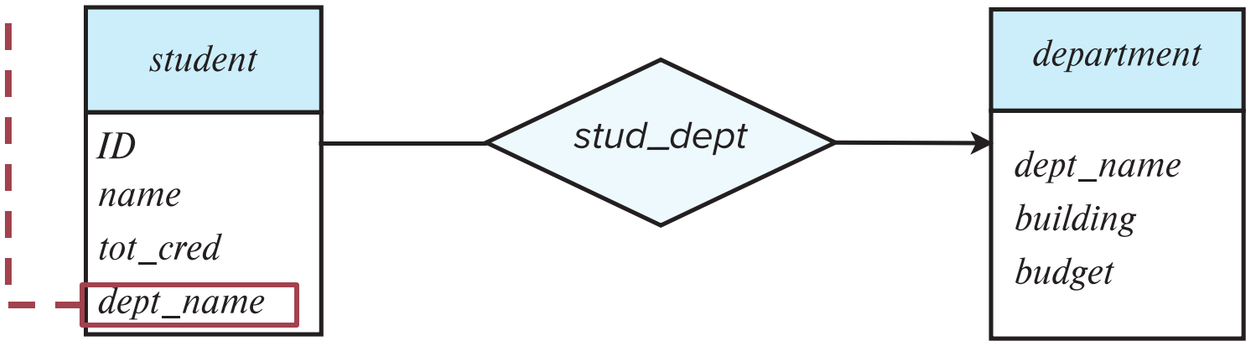
Redundancy of Schemas

- 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

Design Issues

Common Mistakes in E-R Diagrams

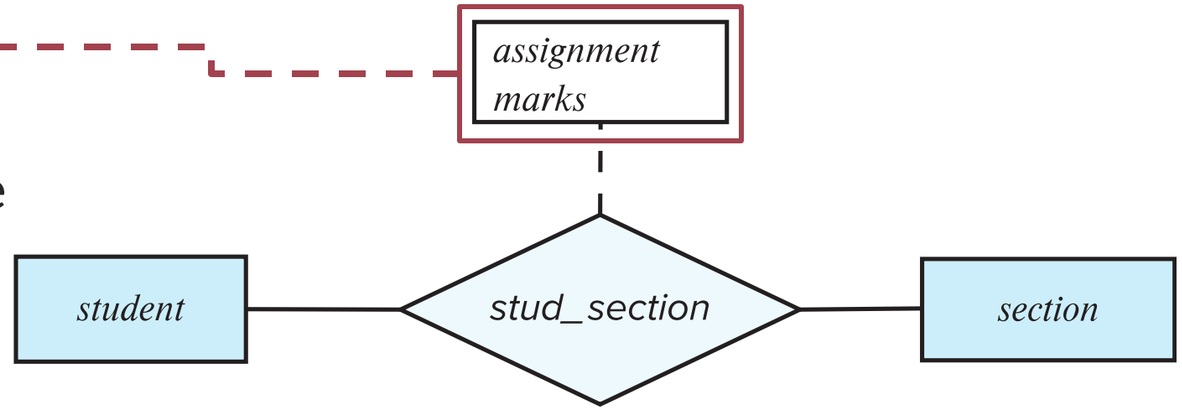
- Examples of erroneous E-R diagrams
  - (a) Unnecessary attribute
    - … which is the primary key of another entity
    - Problem: data redundancy
      - The relationships are already presented in the relationship set (*stud_dept*)



(a) Incorrect use of attribute

Common Mistakes in E-R Diagrams

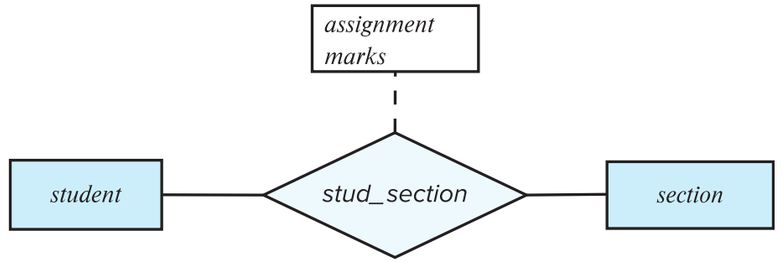
- Examples of erroneous E-R diagrams
  - (b) Erroneous relationship attributes
    - Problem: It cannot represent multiple assignments released in the same section



(b) Erroneous use of relationship attributes

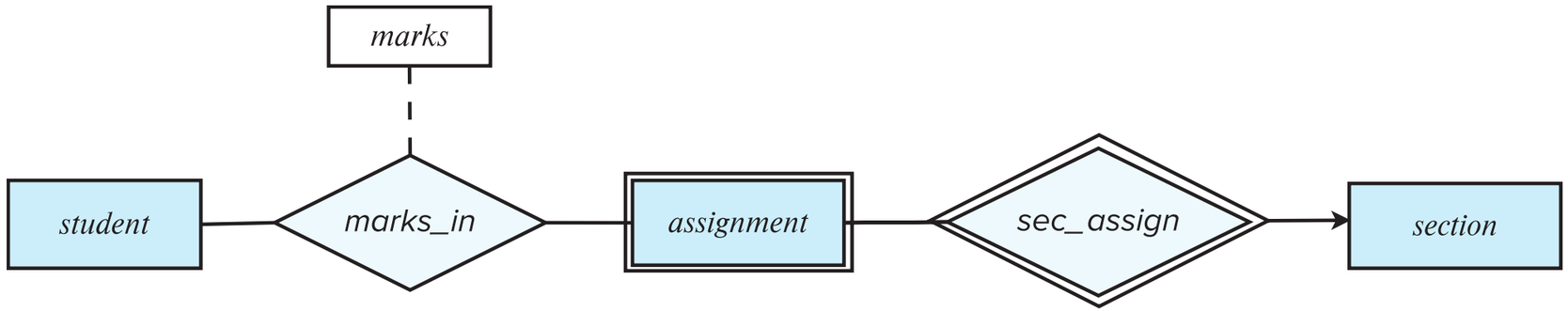
Common Mistakes in E-R Diagrams

- Examples of erroneous E-R diagrams



(b) Erroneous use of relationship attributes

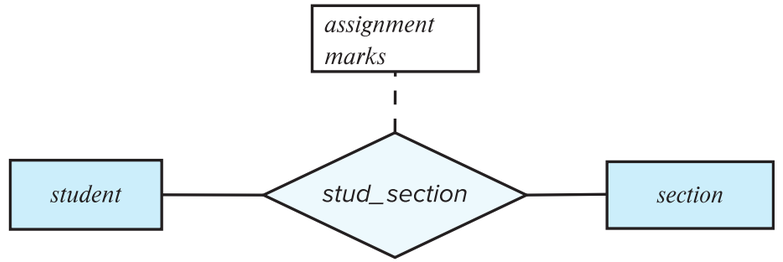
  - (b) Erroneous relationship attributes
    - Problem: It cannot represent multiple assignments released in the same section
    - Solutions:
      - **1) Weak entity set**
      - 2) Composite attributes



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

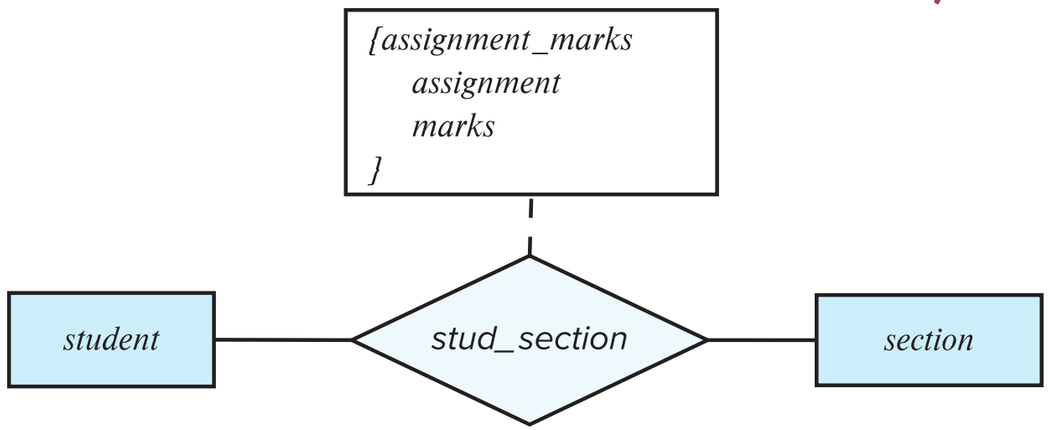
Common Mistakes in E-R Diagrams

- Examples of erroneous E-R diagrams



(b) Erroneous use of relationship attributes

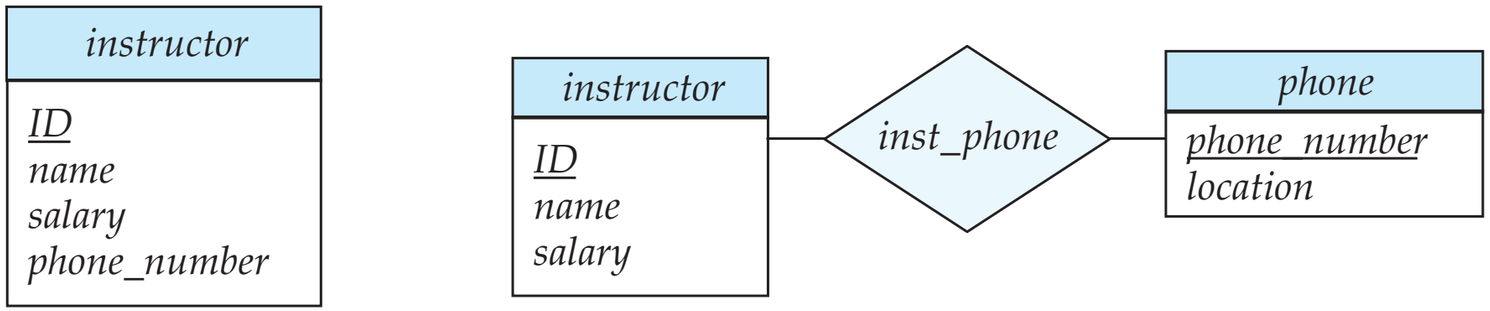
  - (b) Erroneous relationship attributes
    - Problem: It cannot represent multiple assignments released in the same section
    - Solutions:
      - 1) Weak entity set
      - **2) Composite attributes**



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

Entities vs. Attributes

- Use entity sets or attributes?



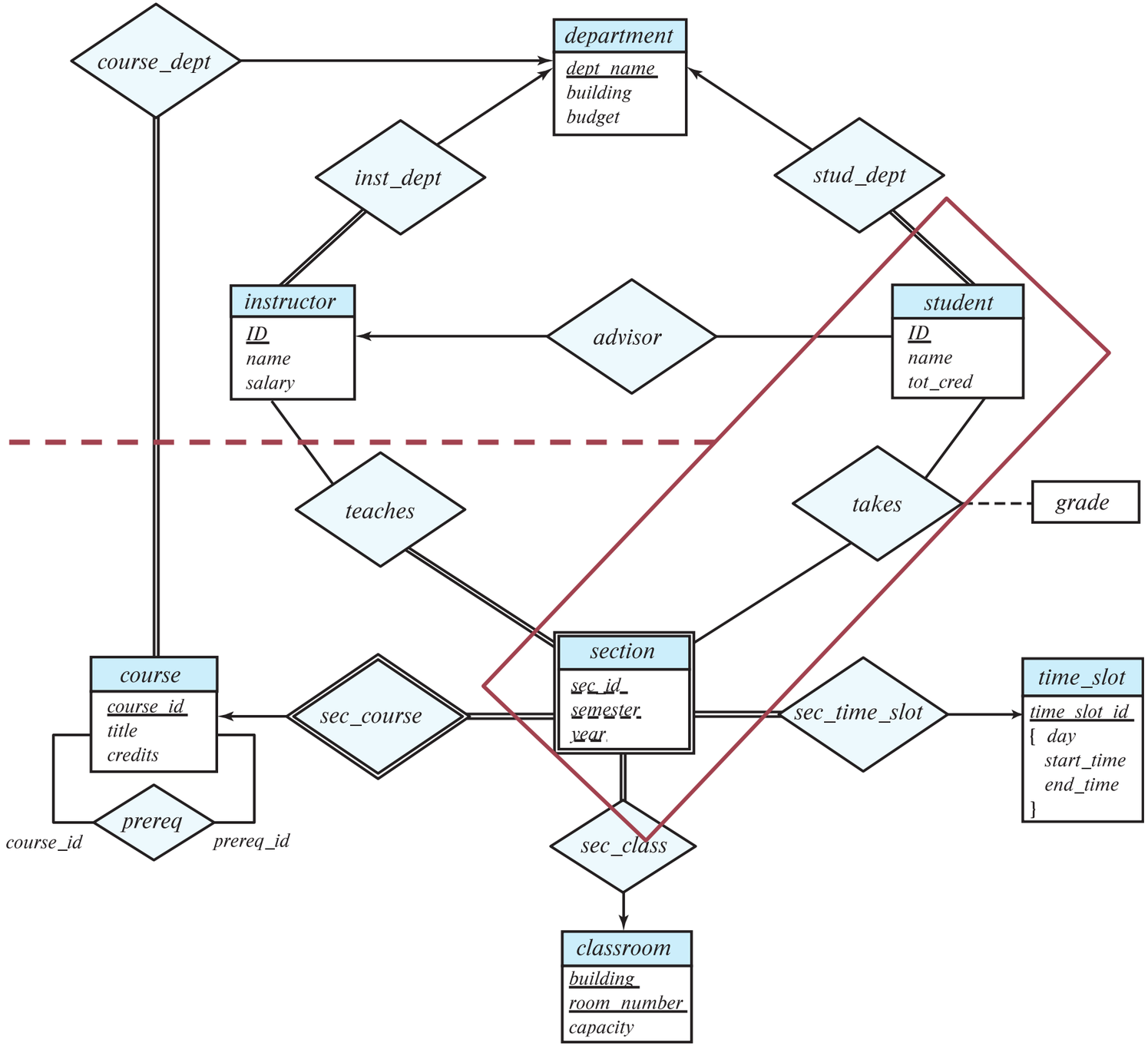
  - Use of *phone* as an entity allows extra information about phone numbers
    - … plus multiple phone numbers

Entities vs. Relationships

- Use entity sets or relationship sets?
  - Well, sometimes it is difficult to answer
  - **A possible guideline**: Use a relationship set to describe an action that occurs between entities

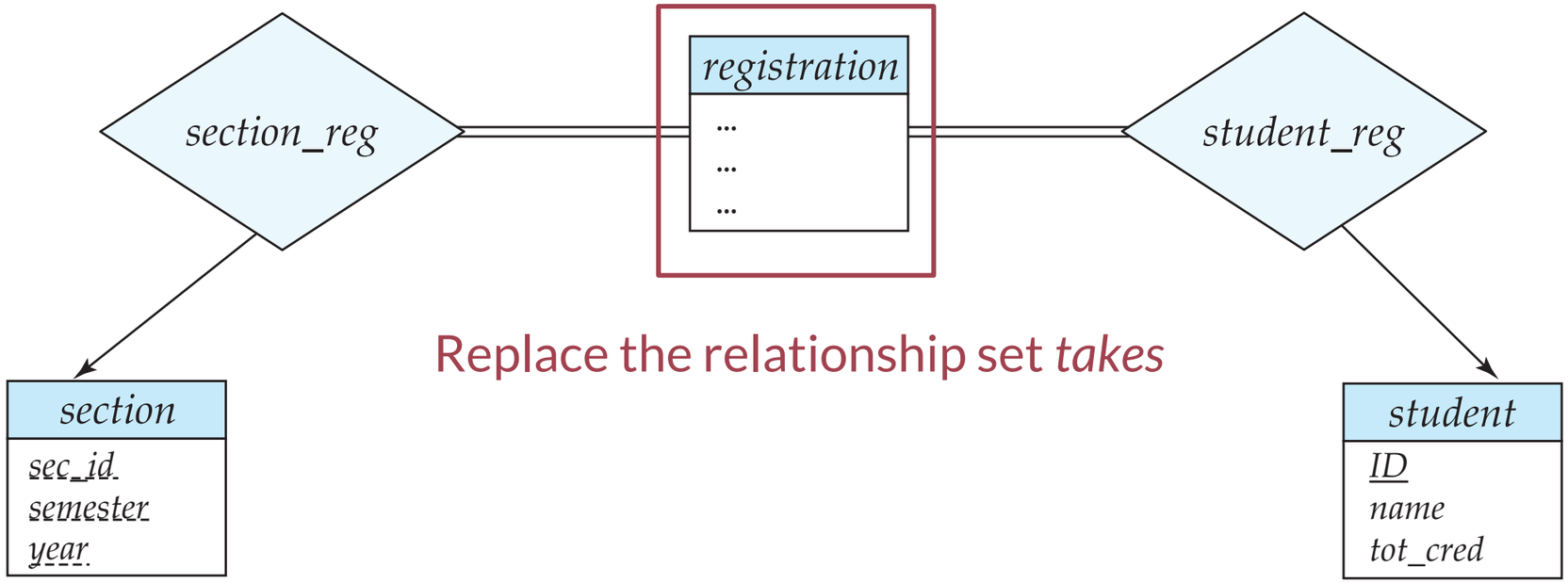
Entities vs. Relationships

- Example: *takes*



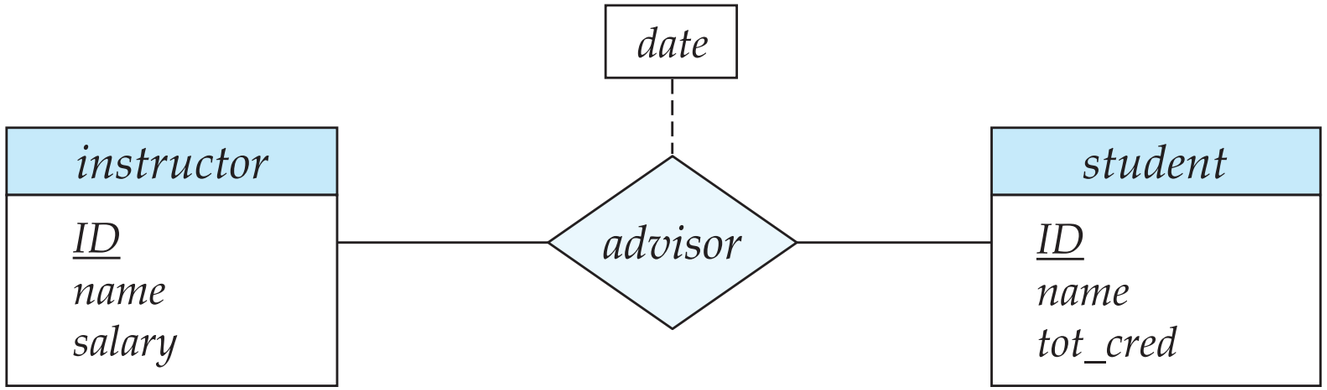
Entities vs. Relationships

- Use entity sets or relationship sets?
  - Well, sometimes it is difficult to answer
  - **A possible guideline**: Use a relationship set to describe an action that occurs between entities



Entities vs. Relationships

- Use entity sets or relationship sets?
  - Well, sometimes it is difficult to answer
  - **A possible guideline**: Use a relationship set to describe an action that occurs between entities
    - This guideline can be used for designing relationship attributes
      - For example, attribute *date* as attribute of advisor or as attribute of student



Binary Vs. Non-Binary Relationships

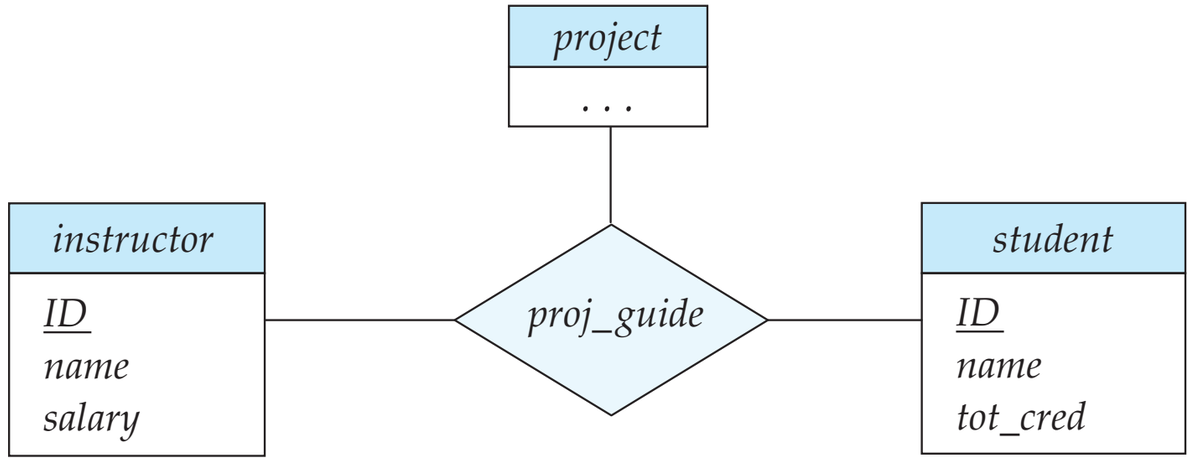
- Although it is possible to replace any non-binary (n-ary, for n > 2) relationship set by a number of distinct binary relationship sets, a n-ary relationship set shows more clearly that several entities participate in a single relationship.

Binary Vs. Non-Binary Relationships

- 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)

Binary Vs. Non-Binary Relationships

- 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*



E-R Design Decisions

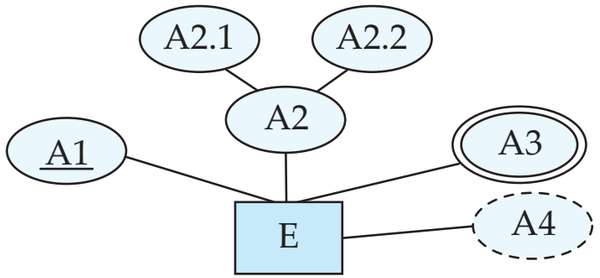
- The use of an attribute or entity set to represent an object
- Whether a real-world concept is best expressed by an entity set or a relationship set
- The use of a ternary relationship versus a pair of binary relationships
- The use of a strong or weak entity set

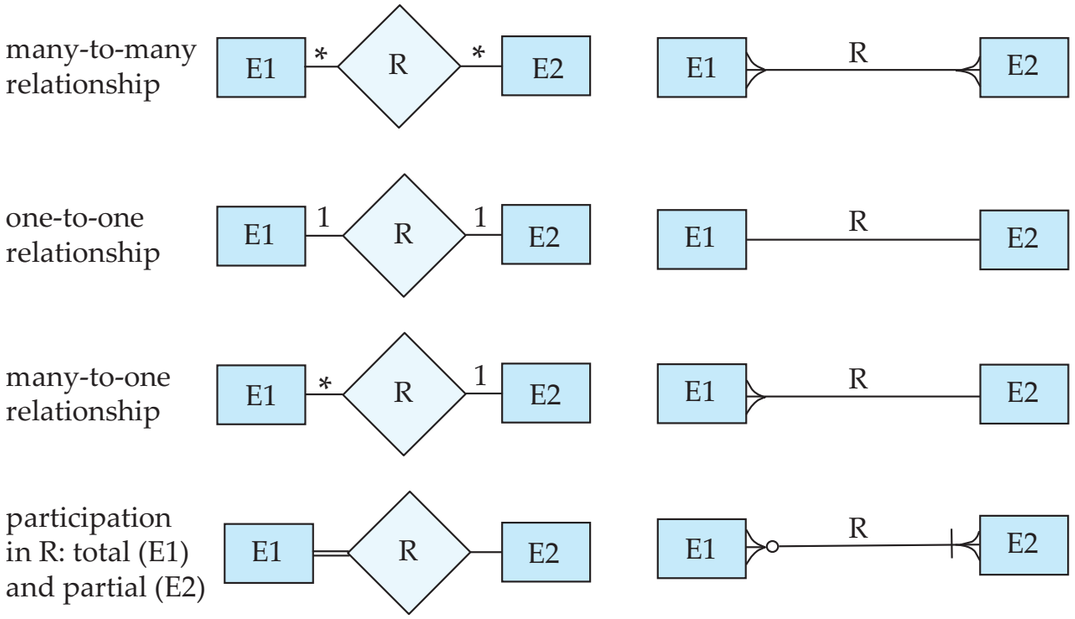
- ** Extra:*
  - ** The use of specialization/generalization – contributes to modularity in the design*
  - ** The use of aggregation – can treat the aggregate entity set as a single unit without concern for the details of its internal structure*

Self Study: Alternative ER Notations

- Chapter 7.10, Database System Concepts (7th Edition)

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





Normalization: A First Look

Recall: 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

Recall: 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

- Do we have any guidelines on how to get a good design?
  - Normal Forms!

First Normal Form (1NF)

- A relational schema R is in **first normal form** if the domains of all attributes of R are atomic
  - Domain is atomic if its elements are considered to be indivisible units
  - Examples of non-atomic domains:
    - Set of names, composite attributes
    - Identification numbers like CS307 that can be broken up into parts
      - However, in practice, we can also consider it atomic
  - Non-atomic values complicate storage and encourage redundant (repeated) storage of data

First Normal Form (1NF)

- Example: Non-atomic attribute

station_id	name	location
1	Luohu(罗湖)	114.11833 , 22.53111
2	Guomao(国贸)	114.11889 , 22.54
3	Laojie(老街)	114.11639 , 22.54444
4	Grand Theater(大剧院)	114.10333 , 22.54472
5	Science Museum(科学馆)	114.08972 , 22.54333
6	Huaqiang Rd(华强路)	114.07889 , 22.54306
7	Gangxia(岗厦)	114.06306 , 22.53778
8	Convention and Exhibition Center Station(会展中心)	114.05472 , 22.5375
9	Shopping Park(购物公园)	114.05472 , 22.53444
10	Xiangmihu(香蜜湖)	114.034 , 22.5417

First Normal Form (1NF)

- Another example: Starring
  - Problems: 1) Redundant names; 2) difficulties in updating/deleting a specific person; 3) extra cost in splitting names; 4) difficulties in making statistics

Movie ID	Movie Title	Country	Year	Director	Starring
0	Citizen Kane	US	1941	welles, o.	Orson Welles, Joseph Cotten
1	La règle du jeu	FR	1939	Renoir, J.	Roland Toutain, Nora Grégor, Marcel Dalio, Jean Renoir
2	North By Northwest	US	1959	HITCHCOCK, A.	Cary Grant, Eva Marie Saint, James Mason
3	Singin' in the Rain	US	1952	Donen/Kelly	Gene Kelly, Debbie Reynolds, Donald O'Connor
4	Rear Window	US	1954	Alfred Hitchcock	James Stewart, Grace Kelly

First Normal Form (1NF)

- Fix it by splitting the names into two columns

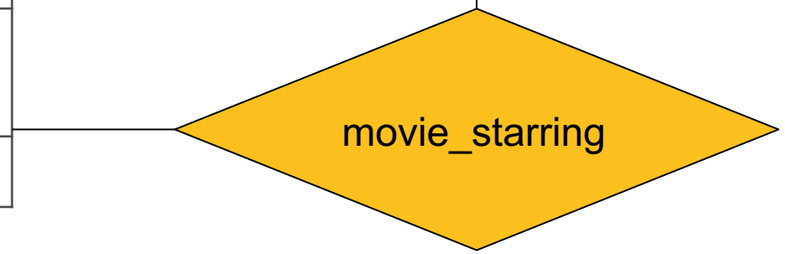
station_id	english_name	chinese_name	longitude	latitude
1	Luohu	罗湖	114.11833	22.53111
2	Guomao	国贸	114.11889	22.54
3	Laojie	老街	114.11639	22.54444
4	Grand Theater	大剧院	114.10333	22.54472
5	Science Museum	科学馆	114.08972	22.54333
6	Huaqiang Rd	华强路	114.07889	22.54306
7	Gangxia	岗厦	114.06306	22.53778
8	Convention and Exhibition Cent…	会展中心	114.05472	22.5375
9	Shopping Park	购物公园	114.05472	22.53444
10	Xiangmihu	香蜜湖	114.034	22.5417

First Normal Form (1NF)

- Fix it by treating the column as a multi-valued attribute

Movie ID	Movie Title	Country	Year	Director
0	Citizen Kane	US	1941	welles, o.
1	La règle du jeu	FR	1939	Renoir, J.
2	North By Northwest	US	1959	HITCHCOCK, A.
3	Singin' in the Rain	US	1952	Donen/Kelly
4	Rear Window	US	1954	Alfred Hitchcock

Star ID	Firstname	Lastname	Born	Died
1				
2				
3				



Second Normal Form (2NF)

- A relation satisfying 2NF must:
  - be in 1NF
  - not have any non-prime attribute that is dependent on any proper subset of any candidate key of the relation
    - A non-prime attribute of a relation is an attribute that is not a part of any candidate key of the relation.

Second Normal Form (2NF)

- Example: Consider this table with the composite primary key (*station_id*, *line_id*)

station_id	english_name	chinese_name	district	line_id	line_color	operator
1	Luohu	罗湖	Luohu	1	Green	Shenzhen Metro Corporation
2	Guomao	国贸	Luohu	1	Green	Shenzhen Metro Corporation
3	Laojie	老街	Luohu	1	Green	Shenzhen Metro Corporation
4	Grand Theater	大剧院	Luohu	1	Green	Shenzhen Metro Corporation
4	Grand Theater	大剧院	Luohu	11	Purple	Shenzhen Metro Corporation
4	Grand Theater	大剧院	Luohu	2	Orange	Shenzhen Metro Corporation
3	Laojie	老街	Luohu	3	DeepSkyBlue	Shenzhen Metro No.3 Line

  - The columns *line_color* and *operator* are not related to *station_id*
    - They are only related to *line_id*, which is only part of (a subset of) the primary key
  - Similarly, *english_name*, *chinese_name*, and *district* are not related to *line_id*
    - They are only related to *station_id*, which is only part of (a subset of) the primary key

Second Normal Form (2NF)

- Example: Consider this table with the composite primary key (*station_id, line_id*)

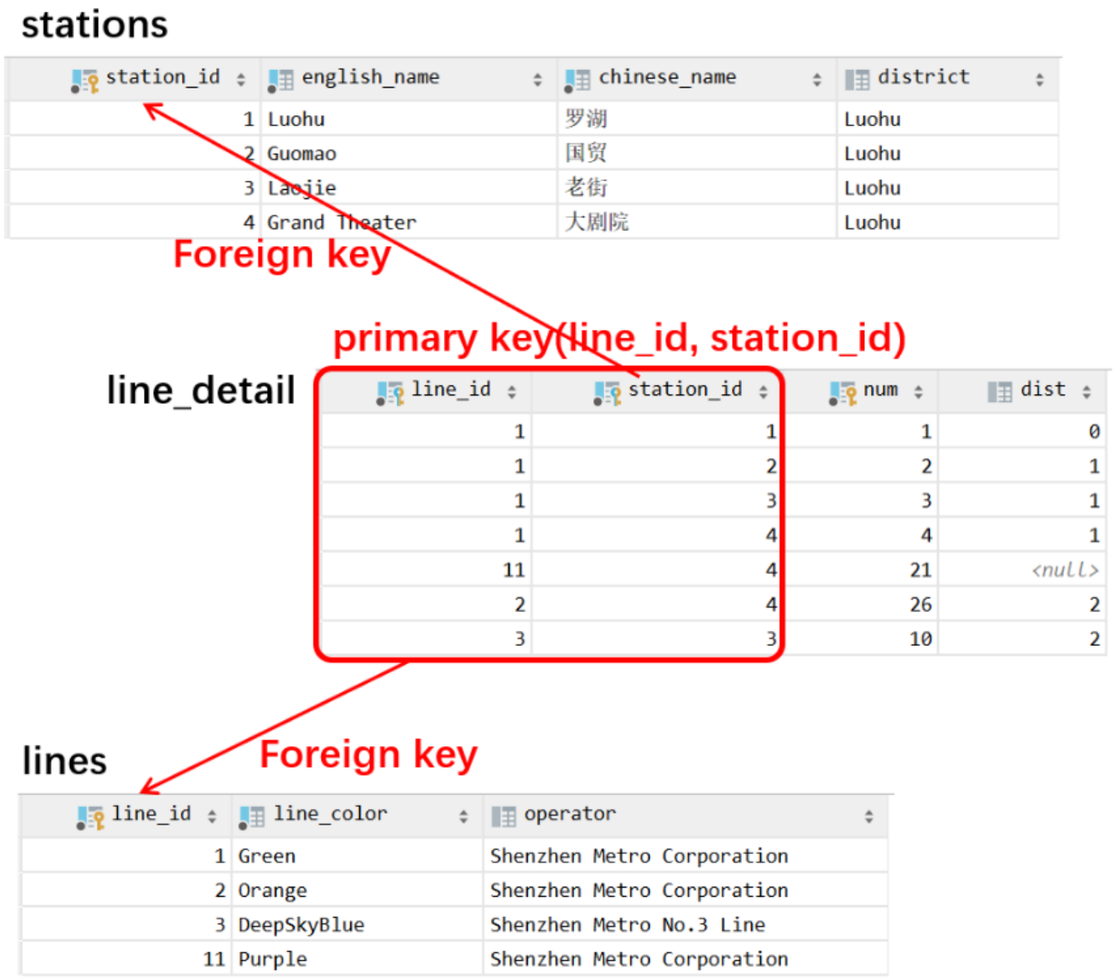
station_id	english_name	chinese_name	district	line_id	line_color	operator
1	Luohu	罗湖	Luohu	1	Green	Shenzhen Metro Corporation
2	Guomao	国贸	Luohu	1	Green	Shenzhen Metro Corporation
3	Laojie	老街	Luohu	1	Green	Shenzhen Metro Corporation
4	Grand Theater	大剧院	Luohu	1	Green	Shenzhen Metro Corporation
4	Grand Theater	大剧院	Luohu	11	Purple	Shenzhen Metro Corporation
4	Grand Theater	大剧院	Luohu	2	Orange	Shenzhen Metro Corporation
3	Laojie	老街	Luohu	3	DeepSkyBlue	Shenzhen Metro No.3 Line

  - Problem when not meeting 2NF: Insertion and deletion anomaly
    - We cannot insert a new station with no lines assigned yet (unless using NULLs)
    - If we delete a line, all stations associated with this line will be deleted as well

Second Normal Form (2NF)

- Fix it by
  - Splitting the two unrelated parts into two different tables of entities
  - And create a relationship set (if it is the many-to-many relationship between the two entities)

- By the way…
  - A relation with a single-attribute primary key is automatically in 2NF once it meets 1NF.



Third Normal Form (3NF)

- A relation satisfying 3NF must:
  - be in 2NF
  - all the attributes in a table are determined only by the candidate keys of that relation and not by any non-prime attributes

Third Normal Form (3NF)

- Example: Consider this table which describes the bus lines and their stops
  - Primary key (*bus_line*)

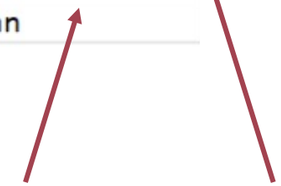
bus_line	station_id	chinese_name	english_name	district
B796	21	鲤鱼门	Liyumen	Nanshan
M343	21	鲤鱼门	Liyumen	Nanshan
M349	21	鲤鱼门	Liyumen	Nanshan
M250	26	坪洲	Pingzhou	Bao'an
374	61	安托山	Antuo Hill	Futian
B733	61	安托山	Antuo Hill	Futian
B828	120	临海	Linhai	Nanshan

  - The column *station_id* depends on the primary key (*bus_line*)
  - However, the columns *chinese_name*, *english_name*, and *district* depend on *station_id*, which is not the primary key.
    - They only have "indirect/transitive" dependence on the primary key
  - Problem: Data redundancy

Third Normal Form (3NF)

- Example: Consider this table which describes the bus lines and their stops
  - Primary key (*bus_line*)

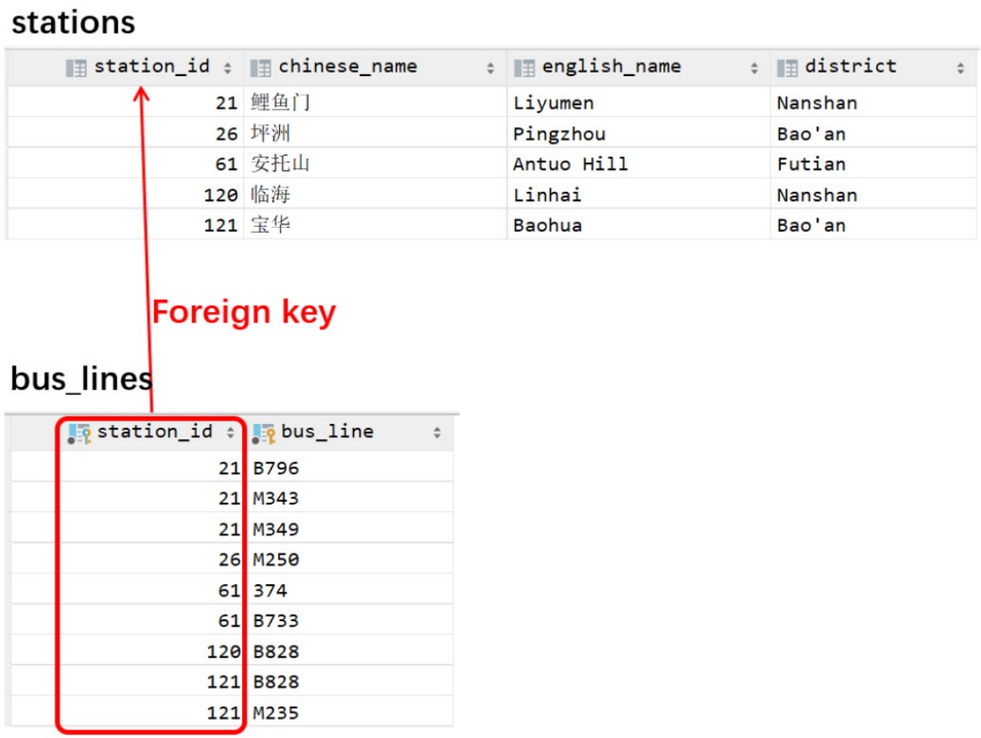
bus_line	station_id	chinese_name	english_name	district
B796	21	鲤鱼门	Liyumen	Nanshan
M343	21	鲤鱼门	Liyumen	Nanshan
M349	21	鲤鱼门	Liyumen	Nanshan
M250	26	坪洲	Pingzhou	Bao'an
374	61	安托山	Antuo Hill	Futian
B733	61	安托山	Antuo Hill	Futian
B828	120	临海	Linhai	Nanshan



- Problem when not meeting 3NF:
  - Data redundancy: as you can see in the table, the attributes for a station have been stored multiple times
  - Insertion and deletion anomaly: inserting a new bus line with no station becomes impossible without NULLs; deleting a station/bus line may also delete corresponding bus lines/stations.

Third Normal Form (3NF)

- Fix it by:
  - Create a new table with *station_id* as the primary key
    - i.e., the column which *chinese_name*, *english_name*, and *district* depend on
  - Move all columns which depend on the new primary key into the new table
    - … and, only leave the primary key of the new table (*station_id*) in the original table
  - (*In practice, if necessary) Add a foreign-key constraint
    - Not related to relational database modeling, only in implementations



Normalization

- In practice, we usually just satisfy 1NF, 2NF and 3NF

	UNF (1970)	1NF (1970)	2NF (1971)	3NF (1971)	EKNF (1982)	BCNF (1974)	4NF (1977)	ETNF (2012)	5NF (1979)	DKNF (1981)	6NF (2003)
Primary key (no duplicate tuples)[4]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Atomic columns (cells cannot have tables as values)[5]	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Every non-trivial functional dependency either does not begin with a proper subset of a candidate key or ends with a prime attribute (no partial functional dependencies of non-prime attributes on candidate keys)[5]	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
Every non-trivial functional dependency either begins with a superkey or ends with a prime attribute (no transitive functional dependencies of non-prime attributes on candidate keys)[5]	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
Every non-trivial functional dependency either begins with a superkey or ends with an elementary prime attribute	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	N/A
Every non-trivial functional dependency begins with a superkey	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	N/A
Every non-trivial multivalued dependency begins with a superkey	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	N/A
Every join dependency has a superkey component[8]	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	N/A
Every join dependency has only superkey components	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	N/A
Every constraint is a consequence of domain constraints and key constraints	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
Every join dependency is trivial	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓

https://en.wikipedia.org/wiki/Database_normalization

Normalization

Every non key attribute must provide a fact about the key, the whole key, and nothing but the key.

William Kent (1936 – 2005)

William Kent. "A Simple Guide to Five Normal Forms in Relational Database Theory", Communications of the ACM 26 (2), Feb. 1983, pp. 120–125.

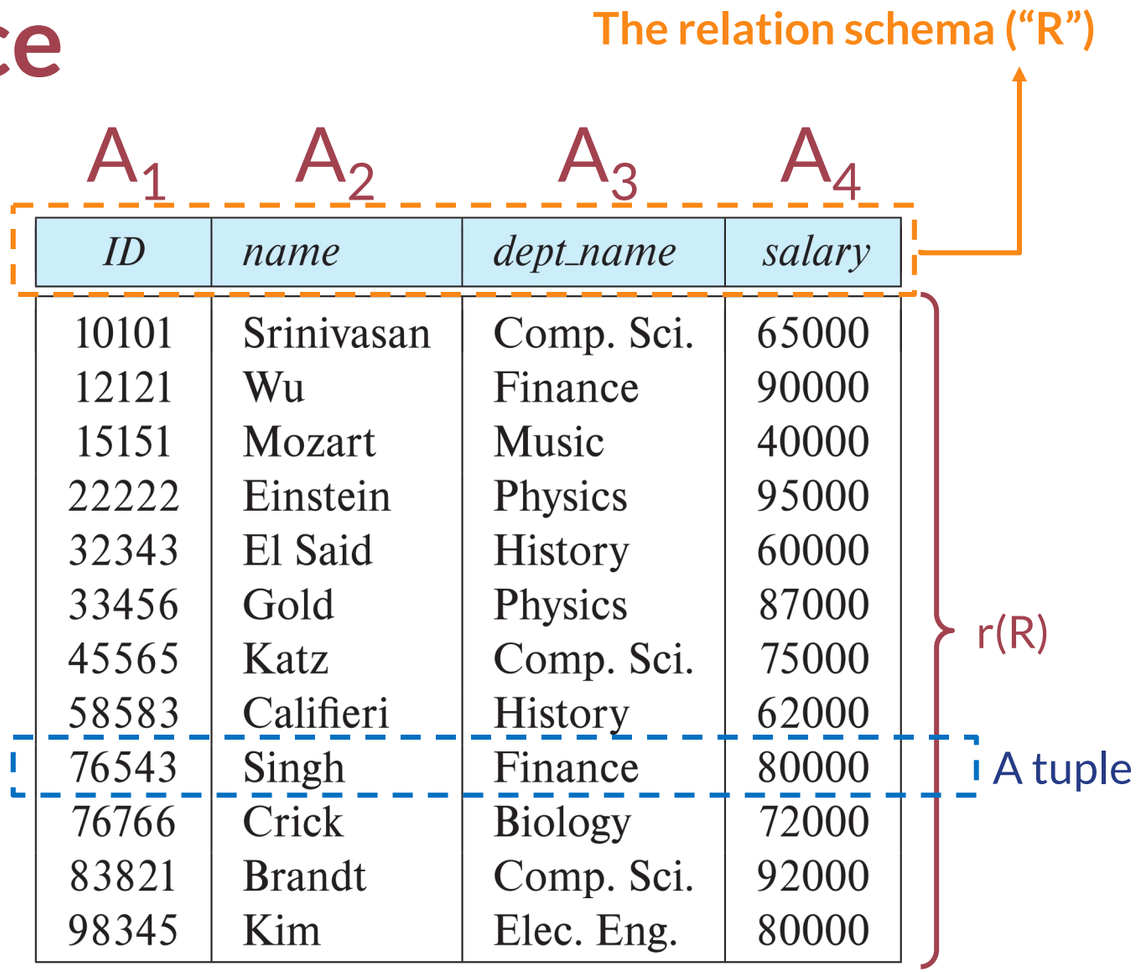
Normalization: A Deeper Look

Prerequisites

Relation Schema and Instance

- $A_1, A_2, \ldots, A_n$ are attributes
- $R = (A_1, A_2, \ldots, A_n)$ is a **relation schema**
  - Example on the right side:

    instructor = (*ID*, *name*, *dept_name*, *salary*)
- $r(R)$ denotes a relation instance r defined over schema R
  - Or to say, the entire table on the right side
- An element t of relation r is called a tuple
  - … and is represented by a row in a table



ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Relation Schema and Instance

- An analogy to programming languages:
  - Relation - Variables
  - Relation schema – Variable types
  - Relation instance – Value(s) stored in the variable

Database Schema

- Database schema is the logical structure of the database
  - It contains a set of relation schemas
  - … and a set of integrity constraints
- Database instance is a snapshot of the data in the database at a given instant in time

Keys

- Let K $\subseteq$ *R*
  - K is a **superkey** of *R* if values for K are sufficient to identify a unique tuple of each possible relation *r*(*R*)
    - E.g., {*ID*} and {*ID*, *name*} are both superkeys of instructor
    - If K is a superkey, any superset K' of K where K' $\subseteq$ *R* is a superkey as well
  - Superkey K is a **candidate key** if K is minimal, i.e., no proper subset of K is a superkey
    - E.g., {ID} is a candidate key for *instructor*
  - One of the candidate keys is selected to be the **primary key**
    - We mark the primary key with an underline:

      instructor = (*ID*, *name*, *dept_name*, *salary*)

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

instructor

Normalization: A Deeper Look

Features of Good Relational Designs

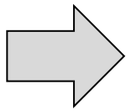
- Suppose we combine *instructor* and *department* into *in_dep*, which represents the natural join on the relations *instructor* and *department*
  - There is repetition of information
  - Need to use nulls (if we add a new department with no instructors)

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

instructor

dept_name	*building*	*budget*
Physics	Watson	70000
Finance	Painter	120000
History	Painter	50000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Biology	Watson	90000
Comp. Sci.	Taylor	100000
History	Painter	50000
Comp. Sci.	Taylor	100000
Music	Packard	80000
Physics	Watson	70000
Finance	Painter	120000

department



ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

in_dep

Decomposition

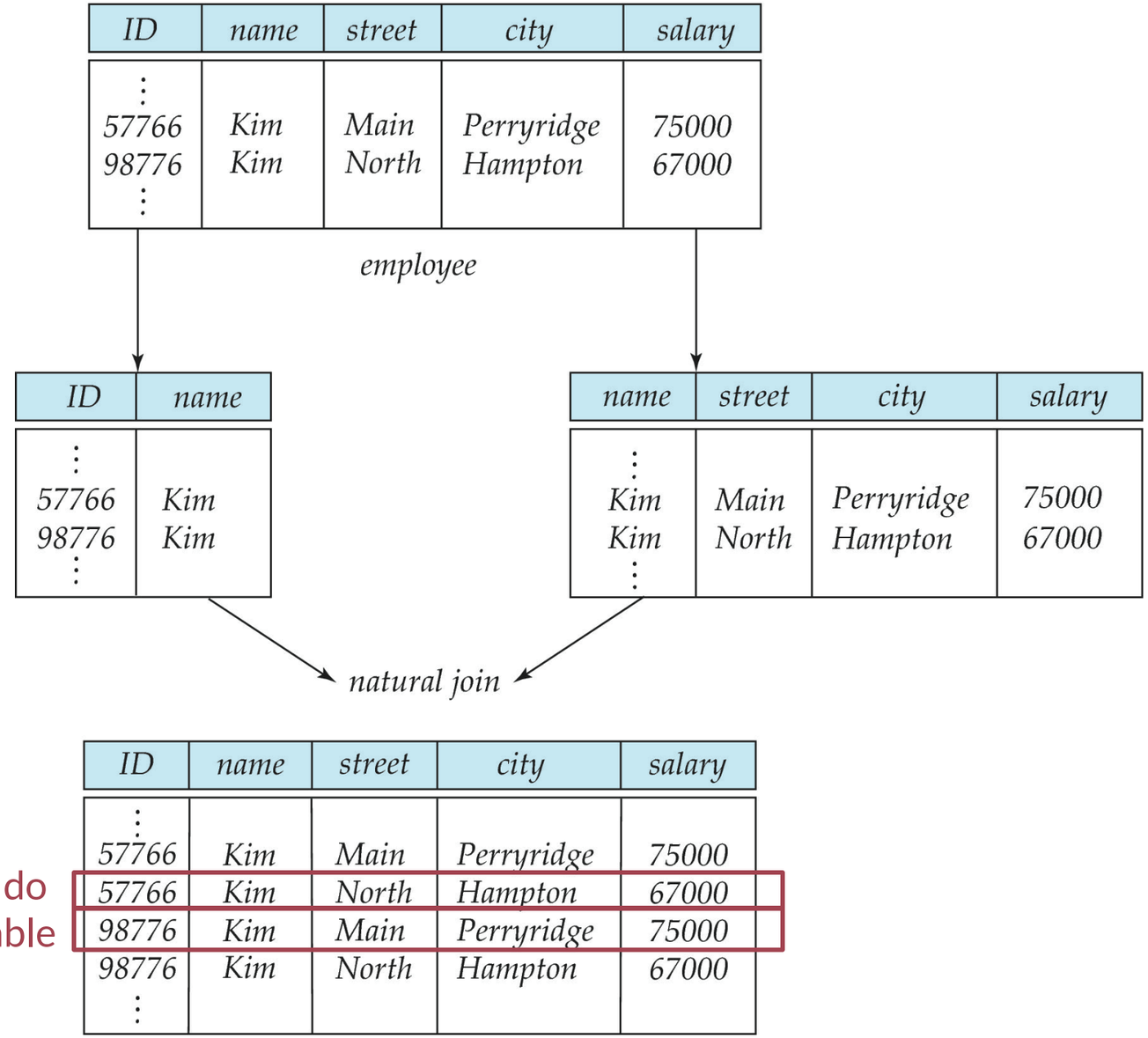
- Avoid the repetition-of-information problem
  - Decompose *in_dep* into two schemas: *instructor* and *department*

- However, not all decompositions are good
  - E.g., decompose *employee*(ID, name, street, city, salary) into:
    - employee1(ID, name)
    - employee2(name, street, city, salary)

  The problem arises when we have two employees with the same name

A Lossy Decomposition

- (Continue) we cannot reconstruct the original employee relation with the join operation
  - We call it a **lossy decomposition**



Two "ghost" records that do not exist in the original table

Lossless Decomposition

- Let R be a relation schema and let R_1 and R_2 form a decomposition of R
  - That is, $R = R_1 \cup R_2$
  - The decomposition is a **lossless decomposition** if there is no loss of information by replacing R with the two relation schemas $R = R_1 \cup R_2$

- Formally, $\prod_{R_1}(r) \bowtie \prod_{R_2}(r) = r$
  - … and a decomposition is lossy if $r \subset \prod_{R_1}(r) \bowtie \prod_{R_2}(r)$
    proper subset

- Or to say, the two SQL queries on the right side generate identical results:

```
select * -- 1
from (select R1 from r)
    natural join
     (select R2 from r);

select * from R; -- 2
```

Normalization Theory

- Decide whether a particular relation *R* is in “good” form

- In the case that a relation *R* is not in “good” form, decompose it into a set of relations $\{R_1, R_2, ..., R_n\}$ such that
  - Each relation is in good form
  - The decomposition is a lossless decomposition

- Our theory is based on:
  - Functional dependencies
  - * Multivalued dependencies (self study)

Functional Dependencies

- There are usually a variety of constraints (rules) on the data in the real world

- For example, some of the constraints that are expected to hold in a university database are:
  - Students and instructors are uniquely identified by their ID
  - Each student and instructor has only one name
  - Each instructor and student is (primarily) associated with only one department
  - Each department has only one value for its budget, and only one associated building

Functional Dependencies

- An instance of a relation that satisfies all such real-world constraints is called a **legal instance** of the relation
  - A legal instance of a database is one where all the relation instances are legal instances
- Constraints on the set of legal relations
  - Require that the value for a certain set of attributes determines uniquely the value for another set of attributes
- A functional dependency is a generalization of the notion of a key

Definition of Functional Dependencies

- Let R be a relation schema, and $\alpha \subseteq R$ *and* $\beta \subseteq R$,

  the functional dependency

  $$\alpha \rightarrow \beta$$

  holds on R if and only if for any legal relations $r(R)$, whenever any two tuples t_1 and t_2 of r agree on the attributes α, they also agree on the attributes β. That is,

  $$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$

- Example: Consider $r(A, B)$ with the following instance of r,
  - On this instance, $A \rightarrow B$ does **NOT** hold, but $B \rightarrow A$ does hold

A	B
1	4
1	5
3	7

Closure of a Set of Functional Dependencies

- Given a set *F* set of functional dependencies, there are certain other functional dependencies that are logically implied by *F*:
  - If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$

- The set of all functional dependencies logically implied by *F* is the **closure** of *F*
  - We denote the closure of *F* by F^+
  - F^+ is a superset of *F*

Keys and Functional Dependencies

- K is a superkey for relation schema R if and only if $K \rightarrow R$
- K is a candidate key for R if and only if
  - $K \rightarrow R$, and
  - for **no** $\alpha \subset K$, $\alpha \rightarrow R$
    proper subset
- Functional dependencies allow us to express constraints that cannot be expressed using superkeys
  - Consider the schema: *inst_dept*(*ID*, *name*, *salary*, *dept_name*, *building*, *budget*)
  - We expect these functional dependencies to hold:

    dept_name*→ *building*, *ID* → *building

    … but would not expect the following to hold:

    dept_name* → *salary

Use of Functional Dependencies

- We use functional dependencies to
  - To test relations to see if they are legal under a given set of functional dependencies
    - If a relation *r* is legal under a set *F* of functional dependencies, we say that *r* satisfies *F*
  - To specify constraints on the set of legal relations
    - We say that *F* holds on *R* if all legal relations on *R* satisfy the set of functional dependencies *F*

Use of Functional Dependencies

- Example: List some functional dependencies that the table satisfies

A	B	C	D
a1	b1	c1	d1
a1	b2	c1	d2
a2	b2	c2	d2
a2	b3	c2	d3
a3	b3	c2	d4

Use of Functional Dependencies

- Example: List some functional dependencies that the table satisfies
  - $A \rightarrow C$

A	B	C	D
a1	b1	c1	d1
a1	b2	c1	d2
a2	b2	c2	d2
a2	b3	c2	d3
a3	b3	c2	d4

Use of Functional Dependencies

- Note: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances.
- Example: we see that *room_number* → *capacity* is satisfied.
  - However, in real world, two classrooms in different buildings can have the same room number but with different room capacity
  - We prefer {*building*, *room_number*} → *capacity*

building	*room_number*	*capacity*
Packard	101	500
Painter	514	10
Taylor	3128	70
Watson	100	30
Watson	120	50

Trivial Functional Dependencies

- A functional dependency is **trivial** if it is satisfied by all relations

- Example:
  - ID, name $\rightarrow$ ID
  - name $\rightarrow$ name

- In general,
  - $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$

Lossless Decomposition

- We can use functional dependencies to show when certain decomposition are lossless
  - For the case of $R = (R_1, R_2)$, we require that for all possible relations r on schema R

$$r = \prod_{R1}(r) \bowtie \prod_{R2}(r)$$

  - A decomposition of R into R_1 and R_2 is a lossless decomposition if at least one of the following dependencies is in F^+:
    - $R_1 \cap R_2 \rightarrow R_1$
    - $R_1 \cap R_2 \rightarrow R_2$

In other words, if $R_1 \cap R_2$ forms a superkey for either R_1 or R_2, the decomposition of R is a lossless decomposition

Lossless Decomposition

- Example:
  - *in_dep (ID, name, salary, dept_name, building, budget)*
  - ... and the decomposed schemas, *instructor* and *department*:
    - *instructor(ID, name, dept_name, salary)*
    - *department(dept_name, building, budget)*

$instructor \cap department = dept\_name$

$dept\_name \rightarrow dept\_name, building, budget$

(… which means the decomposition is lossless)

Lossless Decomposition

- Another example:

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C)$
- $R_1 = (A, B), \quad R_2 = (B, C)$
  - Lossless decomposition:

    $R_1 \cap R_2 = \{B\}$ and $B \rightarrow BC$
- $R_1 = (A, B), \quad R_2 = (A, C)$
  - Lossless decomposition:

    $R_1 \cap R_2 = \{A\}$ and $A \rightarrow AB$
- *Note:*
  - $B \rightarrow BC$

    is a shorthand notation for
  - $B \rightarrow \{B, C\}$

Lossless Decomposition

- Note: the above functional dependencies are a sufficient condition for lossless join decomposition
  - The dependencies are a necessary condition only if all constraints are functional dependencies

*(There are other types of constraints, e.g., **multivalued dependencies**, that can ensure that a decomposition is lossless even if no functional dependencies are present)*

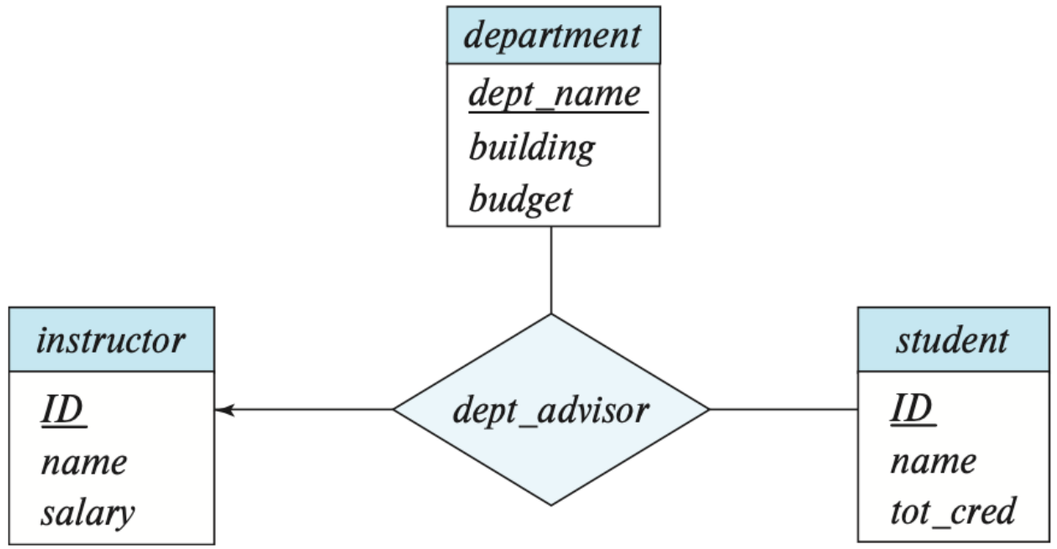
Dependency Preservation

- Testing functional dependency constraints each time the database is updated can be costly
  - It is useful to design the database in a way that constraints can be tested efficiently.

- If a functional dependency in the original relation *R* does not exist in any of the decomposed relations, we say it is not dependency-preserving
  - In the dependency preservation, at least one decomposed table must satisfy every dependency

Dependency Preservation

- Consider a new E-R design for relationships between students, instructors, and departments
  - An instructor can only be associated with one department
  - A student can have multiple advisors but not more than one from a given department
    - Think about double-major students



Dependency Preservation

- Consider a schema
  - *dept_advisor(s_ID, i_ID, dept_name)*
  - … with function dependencies: (1) *i_ID* → *dept_name* (2) *s_ID*, *dept_name* → *i_ID*

In the above design, we are forced to repeat the department name once for each time an instructor participates in a *dept_advisor* relationship.

Dependency Preservation

- Consider a schema
  - *dept_advisor*(*s_ID*, *i_ID*, *dept_name*)
  - … with function dependencies: (1) *i_ID* → *dept_name* (2) *s_ID*, *dept_name* → *i_ID*

In the above design, we are forced to repeat the department name once for each time an instructor participates in a *dept_advisor* relationship.

- To fix this problem, we need to decompose *dept_advisor*
  - However, any decomposition will not include all the attributes in

    s_ID, *dept_name* → *i_ID*
  - Thus, the decomposition will NOT be dependency-preserving

Dependency Preservation

- Problem when not meeting dependency preservation
  - Every time the database wants to check the integrity of the functional dependency

    $$s\_ID, dept\_name \rightarrow i\_ID,$$

    the decomposed tables must be joined

  - … where the computational cost could be very high with join operations

Normal Forms: Revisited

- Boyce-Codd Normal Form (BCNF)
- 3NF
- Higher-order normal forms

Boyce-Codd Normal Form

- A relation schema *R* is in **BCNF** with respect to a set *F* of functional dependencies if for all functional dependencies in F^+ of the form

$$\alpha \rightarrow \beta$$

where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
  - α is a superkey for R

- * A database design is in BCNF if each member of the set of relation schemas that constitutes the design is in BCNF

Boyce-Codd Normal Form

- Example schema that is **not** in BCNF:
  - *in_dep* (*ID*, *name*, *salary*, *dept_name*, *building*, *budget)*

  Because,

  $$dept\_name \rightarrow building, budget$$

  - holds in *in_dep*, however, *dept_name* is not a superkey
    - … where {ID, dept_name} is

  - When decompose *in_dept* into *instructor* and *department*
    - *instructor* is in BCNF
    - *department* is in BCNF

Decomposing a Schema into BCNF

- Let *R* be a schema *R* that is not in BCNF
- Let $\alpha \rightarrow \beta$ be the functional dependency that causes a violation of BCNF
  - We decompose *R* into:
    - $(\alpha \cup \beta)$
    - $(R - (\beta - \alpha))$

- Example: *in_dep (<u>ID</u>, name, salary, <u>dept_name</u>, building, budget)*
  - α = *dept_name*, β = *building, budget*
  - Thus, *in_dep* is replaced by:
    - $(\alpha \cup \beta)$ = (*dept_name, building, budget*)
    - $(R - (\beta - \alpha))$ = (*ID, name, dept_name, salary*)

Decomposing a Schema into BCNF

- Another example:
  - $R = (A, B, C)$
    $F = \{A \rightarrow B, B \rightarrow C\}$
  - $R_1 = (A, B), \quad R_2 = (B, C)$
    - Lossless-join decomposition:

      $$R_1 \cap R_2 = \{B\} \quad \text{and } B \rightarrow BC$$

    - Dependency preserving
  - $R_1 = (A, B), \quad R_2 = (A, C)$
    - Lossless-join decomposition:

      $$R_1 \cap R_2 = \{A\} \text{ and } A \rightarrow AB$$

    - Not dependency preserving
      (cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

BCNF and Dependency Preservation

- It is not always possible to achieve both BCNF and dependency preservation
- Consider the schema (that we have visited before)
  - *dept_advisor(s_ID, i_ID, dept_name)*
  - … with function dependencies: (1) $i\_ID \rightarrow dept\_name$ (2) $s\_ID, dept\_name \rightarrow i\_ID$
  - *dept_advisor* is not in BCNF since for $i\_ID \rightarrow dept\_name$, i_ID is not a superkey
    - (where {*s_ID*, *i_ID*, *dept_name*} is)
- To fix this problem, we need to decompose *dept_advisor*
  - However, any decomposition will **not** include all the attributes in $s\_ID, dept\_name \rightarrow i\_ID$
  - Thus, the decomposition will NOT be dependency-preserving

Third Normal Form (3NF)

- A relation schema R is in **third normal form (3NF)** if for all

  $$\alpha \rightarrow \beta \text{ in } F^+$$

  at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
  - α is a superkey for R
  - Each attribute A in $\beta - \alpha$ is contained in a candidate key for R

- Notes
  - Each attribute A may be in a different candidate key
  - If a relation is in BCNF, it is in 3NF (… since in BCNF, one of the first two conditions above must hold)
  - The third condition above is a minimal relaxation of BCNF to ensure dependency preservation

3NF Example

- Consider the schema (that we have visited before)
  - *dept_advisor(s_ID, i_ID, department_name)*
  - … with function dependencies: (1) *i_ID* $\rightarrow$ *dept_name* (2) *s_ID*, *dept_name* $\rightarrow$ *i_ID*

  - We have two candidate keys: {*s_ID*, *dept_name*} and {*s_ID*, *i_ID*}

- *dept_advisor* is not in BCNF, but it can be in 3NF
  - {s_ID, dept_name} is a superkey
  - *i_ID* $\rightarrow$ *dept_name* and *i_ID* is NOT a superkey (which violates BCNF), but:
    - α is *i_ID*, β is *dept_name*
    - {*dept_name*} – {*i_ID*} = {*dept_name*}
    - *dept_name* is contained in a candidate key (-> {*s_ID*, *dept_name*})

Redundancy in 3NF

- Consider the schema R below, which is in 3NF
  - $R = (J, K, L)$, $F = \{JK \rightarrow L, L \rightarrow K\}$, and an instance table:

J	*L*	*K*
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
null	l_2	k_2

- Problems in this table:
  - Repetition of information
    - Row 1-3: L and K
  - Need to use nulls
    - Row 4: Represent the relationship l_2, k_2 with no corresponding value for J

Comparison of BCNF and 3NF

- Advantages to 3NF over BCNF
  - It is always possible to obtain a 3NF design without sacrificing *losslessness* or *dependency preservation*
- Disadvantages to 3NF
  - We may have to use nulls to represent some of the possible meaningful relationships among data items
  - There is a problem of potential repetition of information

Goals of Normalization

- Let *R* be a relation scheme with a set *F* of functional dependencies
  - Decide whether a relation scheme *R* is in "good" form.
  - In the case that a relation scheme R is not in "good" form, need to decompose it into a set of relation scheme $\{R_1, R_2, ..., R_n\}$ such that:
    - Each relation scheme is in good form
    - The decomposition is a lossless decomposition
    - *Preferably*, the decomposition should be dependency preserving

How good is BCNF?

- There are database schemas in BCNF that do not seem to be sufficiently normalized
  - Consider a relation *inst_info*(*ID*, *child_name*, *phone*)
    - … where an instructor may have more than one phone and can have multiple children
      - Actually, we would better use two relations: (*ID*, *child_name*) and (*ID*, *phone*)

An instance of *inst_info*:

(99999,	David,	512-555-1234)
(99999,	David,	512-555-4321)
(99999,	William,	512-555-1234)
(99999,	William,	512-555-4321)

How good is BCNF?

- inst_info is in BCNF
  - since *ID* $\rightarrow$ *child_name*, *ID* $\rightarrow$ *phone*, and *ID* is the superkey

- However,
  - Insertion anomalies
    - If we add a phone number 981-992-3443 to the instructor 99999, we need to add two tuples:

      (99999, David, 981-992-3443)
      (99999, William, 981-992-3443)

    - If we only add one of the two tuples above, it will imply that only David or William corresponds to 981-992-3443, which is not the functional dependency we need to keep

Higher Normal Forms

- It is better to decompose *inst_info* into *inst_child* and *inst_phone*:

ID	*child_name*
99999	David
99999	William

ID	*phone*
99999	512-555-1234
99999	512-555-4321

  - This suggests a need for higher normal forms, such as Fourth Normal Form (4NF) that resolves such kind of multivalued dependencies

Self Study

- Database System Concepts , 7th Edition
  - Chapter 7.4 “Functional Dependency Theory”
  - Chapter 7.5 “Algorithms for Decomposition Using Functional Dependencies”
  - Chapter 7.6 “Decomposition Using Multivalued Dependencies”
  - Chapter 7.7 “More Normal Forms”

Wait, Where are 1NF and 2NF?

- 1NF is about the attribute domains but not decompositions
  - … and hence not quite related to dependencies we have learned in this section

Wait, Where are 1NF and 2NF?

- 2NF: Partial dependency
  - A functional dependency $\alpha \rightarrow \beta$ is called a partial dependency if there is a proper subset γ of α such that $\gamma \rightarrow \beta$
    - We say that β is partially dependent on α
  - A relation schema R is in second normal form (2NF) if each attribute A in R meets one of the following criteria:
    - It appears in a candidate key
    - It is not partially dependent on a candidate key

Wait, Where are 1NF and 2NF?

- 2NF: Partial dependency
  - You can try to prove that a relation meeting 3NF also satisfies 2NF
    - Exercise 7.19 in “Database System Concepts, 7th Edition”
  - In practice, we usually choose to satisfy 3NF or BCNF

Summary

Overall Database Design Process

- We have assumed schema *R* is given
  - *R* could have been generated when converting E-R diagram to a set of tables
  - *R* could have been a single relation containing all attributes that are of interest (called universal relation)
  - Normalization breaks *R* into smaller relations
  - *R* could have been the result of some ad-hoc design of relations, which we then test/convert to normal form

E-R Model and Normalization

- When an E-R diagram is carefully designed, identifying all entities correctly, the tables generated from the E-R diagram should not need further normalization.
  - However, in a real (imperfect) design, there can be functional dependencies from non-key attributes of an entity to other attributes of the entity
  - Example: an *employee* entity with attributes *department_name* and *building*
    - … but with functional dependency: *department_name* $\rightarrow$ *building*
    - Good design would have made department an entity

Denormalization for Performance

- We may want to use non-normalized schema for performance
- For example, displaying *prereqs* along with *course_id*, and *title* requires join of *course* with *prereq*
  - Alternative 1: Use denormalized relation containing attributes of *course* as well as *prereq* with all above attributes
    - faster lookup
    - extra space and extra execution time for updates
    - extra coding work for programmer and possibility of error in extra code
  - Alternative 2: use a materialized view defined a *course* ⋈ *prereq*
    - Benefits and drawbacks same as above, except no extra coding work for programmer and avoids possible errors

Other Design Issues

- Some aspects of database design are not caught by normalization
  - Examples of bad database design, to be avoided: Instead of *earnings* (*company_id*, *year*, *amount*), use
    - *earnings_2004*, *earnings_2005*, *earnings_2006*, etc., all on the schema (*company_id*, *earnings*).
      - Above are in BCNF, but make querying across years difficult and needs new table each year
    - *company_year* (*company_id*, *earnings_2004*, *earnings_2005*, *earnings_2006*)
      - Also in BCNF, but also makes querying across years difficult and requires new attribute each year
      - It is an example of a **crosstab**, where values for one attribute become column names
      - Such crosstabs are widely used in spreadsheets and data analysis tools