Introduction to Database Systems

2023-Fall

- **ER Model and ER Diagram**
- **Database Design Method**

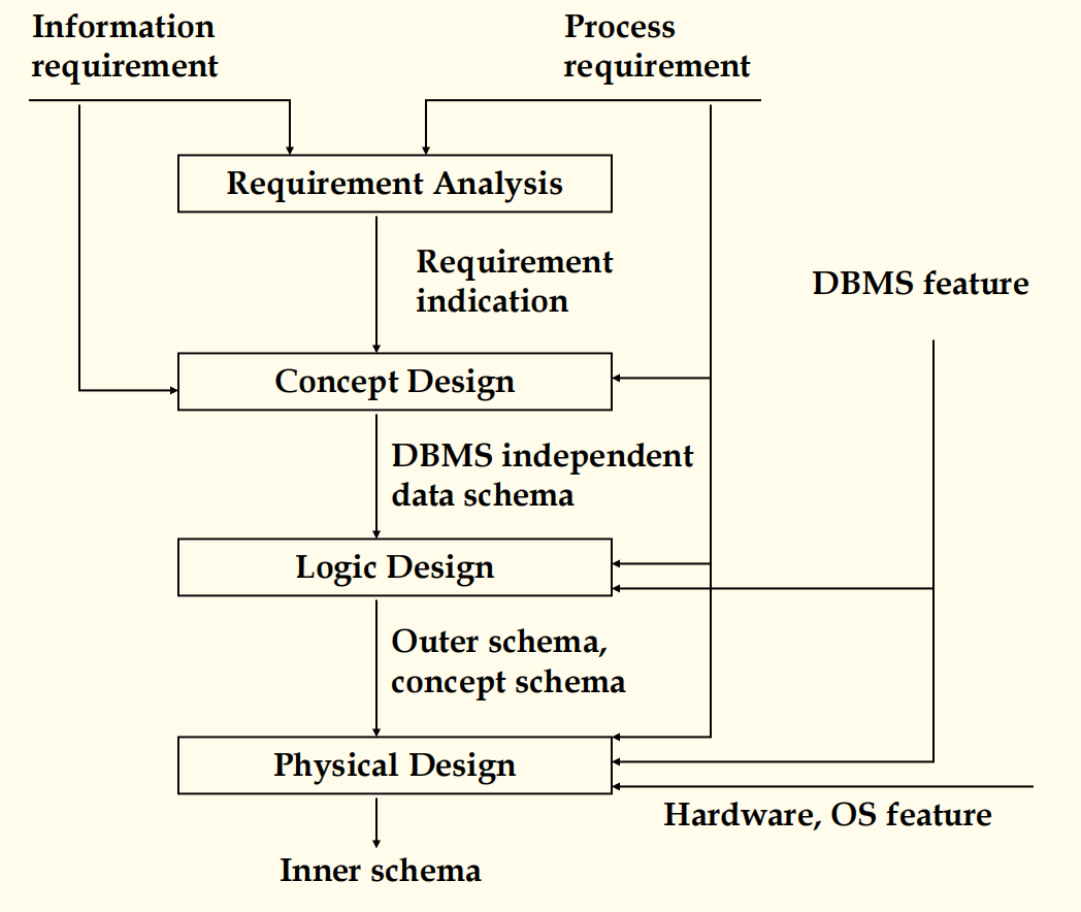
Database Design Method

- **Procedure oriented method**
- This method takes business procedures as center, the database schema is designed basically in accordance directly with the vouchers, receipts, reports, etc. in business. Because of no detailed analysis on data and inner relationships between data, although it is fast at the beginning of the project, it is hard to ensure software quality and the system will be hard to fit future changes in requirement and environment. So this method is not suitable for the development of a large, complex system.
- **Data oriented method**
- This method design the database schema based on the detailed analysis on data and inner relationships between data which are involved in business procedures. It takes data as center, not procedures. It can not only fulfill the current requirements, but also some potential requirements. It is liable to fit future changes in requirement and environment. It is recommended in the development of large, complex systems

Steps in Database Design

- **Requirements Analysis**
  - user needs; what must database do?
- **Conceptual Design**
  - *high level description (often done w/ER model)*
  - Object-Relational Mappings (ORMs: Hibernate, Rails, Django, etc) encourage you to program here
- **Logical Design**
  - translate ER into DBMS data model
  - ORMs often require you to help here too
- **Schema Refinement**
  - consistency, normalization
- **Physical Design** - indexes, disk layout
- **Security Design** - who accesses what, and how

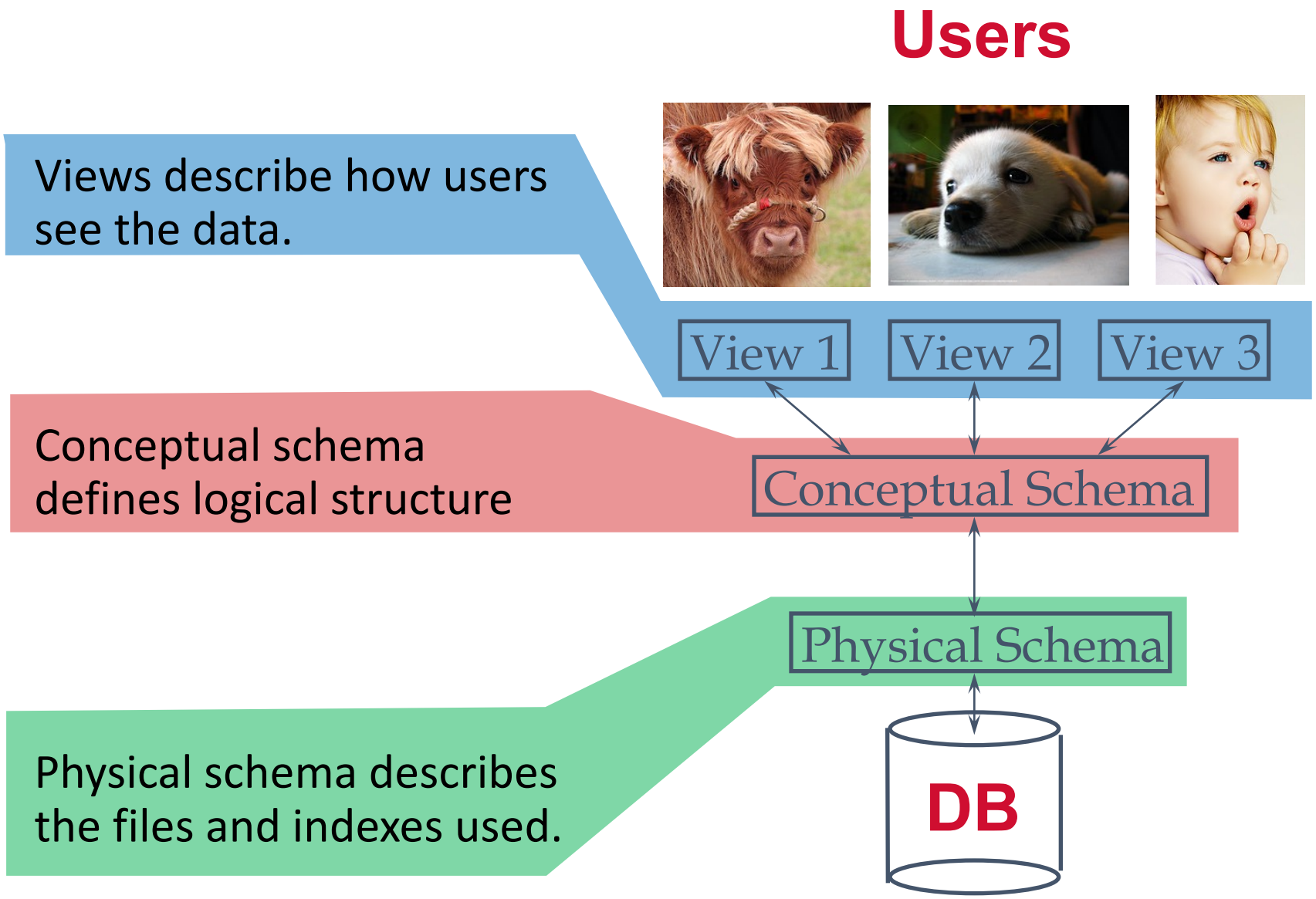
Database Design Flow



Describing Data: Data Models

- **Data model :** collection of concepts for describing data.
- **Schema:** description of a particular collection of data, using a given data model.
- **Relational model of data**
  - Main concept: relation (table), rows and columns
  - Every relation has a schema
    - describes the columns
    - column names and domains

Levels of Abstraction



Example: University Database

- **Conceptual schema:**
  - `Students(sid text, name text, login text, age integer, gpa float)`
  - `Courses(cid text, cname text, credits integer)`
  - `Enrolled(sid text, cid text, grade text)`
- **Physical schema:**
  - Relations stored as unordered files.
  - Index on first column of Students.
- **External Schema** (View):
  - Course_info(cid text, enrollment integer)

Data Independence

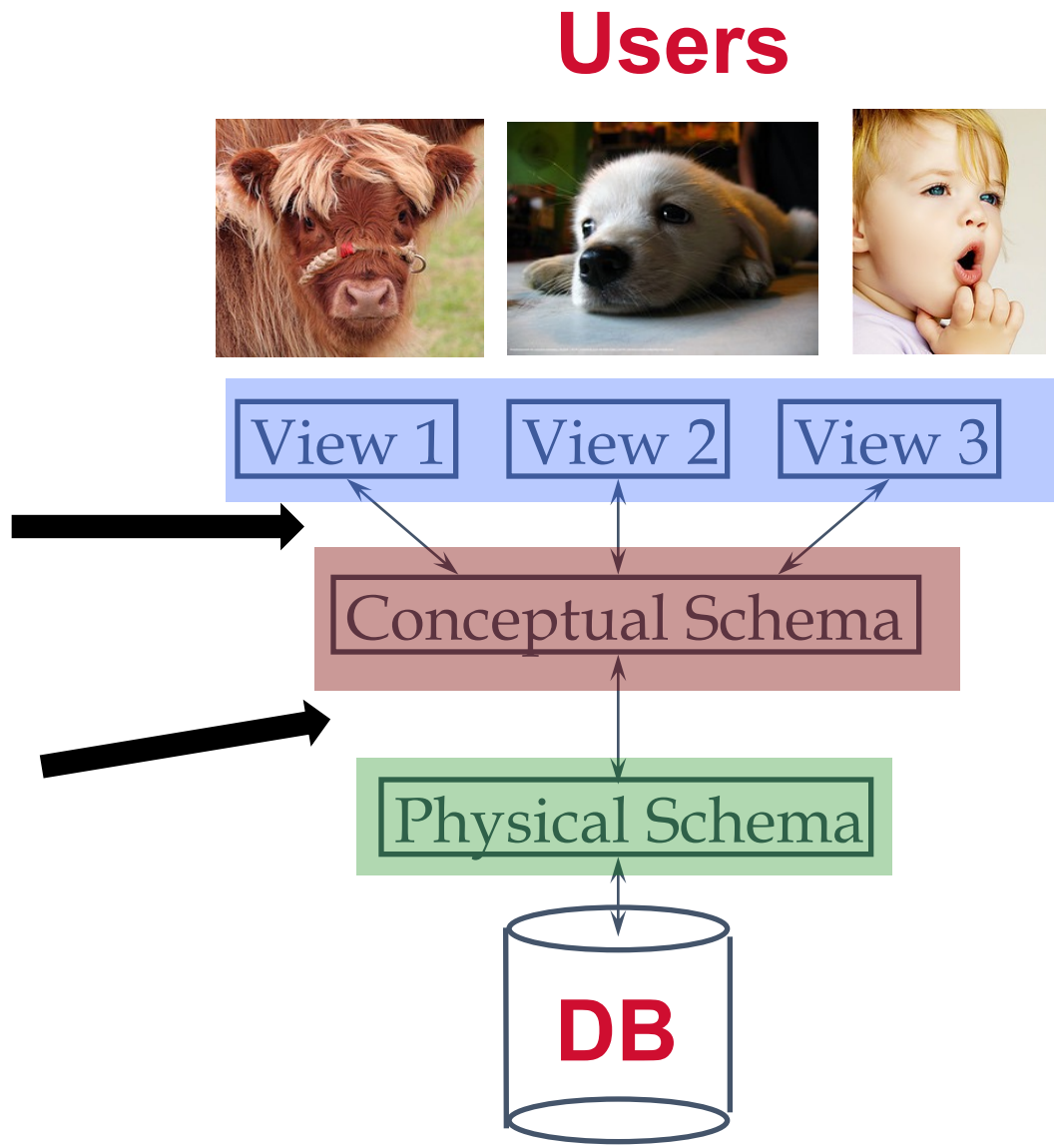
- Insulate apps from structure of data

- **Logical data independence:**
  - Maintain views when logical structure changes
- **Physical data independence:**
  - Maintain logical structure when physical structure changes

Levels of Abstraction, cont

Logical data independence

Physical data independence



Data Independence, cont

- Insulate apps from structure of data

- **Logical data independence:**
  - Maintain views when logical structure changes
- **Physical data independence:**
  - Maintain logical structure when physical structure changes

- Q: Why particularly important for DBMS?
  - Because databases and their associated applications persist

Requirement Analysis

- A very important part of system requirement analysis. In requirement analysis phase, the data dictionary and DFD (or UML) diagrams are the most important to database design.
- **Dictionary and DFD**
  - Name conflicts
  - Homonym(the same name with different meanings)
  - Synonym(the same meaning in different names)
  - Concept conflicts
  - Domain conflicts
- **About coding**
  - Standardization of information
  - Identifying entities
  - Compressing information
- **Through requirement analysis, all information must be with unique source and unique responsibility.**

Concept Design

- Based on data dictionary and DFD, analyze and classify the data in data dictionary, and refer to the processing requirement reflected in DFD, identify entities, attributes, and relationships between entities. Then we can get concept schema of the database.
  - Identify Entities
  - Define the relationships between entities
  - Draw ER diagram and discuss it with user
- It is proposed to use ER design tools such as ERWin, Rose, etc.

Entity-Relationship Model

- Relational model is a great formalism
  - But a bit detailed for design time
  - Too fussy for brainstorming
  - Hard to communicate to “customers”
- Entity-Relationship model: a graph-based model
  - can be viewed as a graph, or a veneer over relations
    - “feels” more flexible, less structured
  - corresponds well to “Object-Relational Mapping”
    - (ORM) SW packages
    - Ruby-on-Rails, Django, Hibernate, Sequelize, etc.

Steps in Database Design, again

- Requirements Analysis
  - user needs; what must database do?
- **Conceptual Design**
  - ***high level description (often done w/ER model)***
  - **ORM encourages you to program here**
- Logical Design
  - translate ER into DBMS data model
  - ORMs often require you to help here too
- Schema Refinement
  - consistency, normalization
- Physical Design - indexes, disk layout
- Security Design - who accesses what, and how

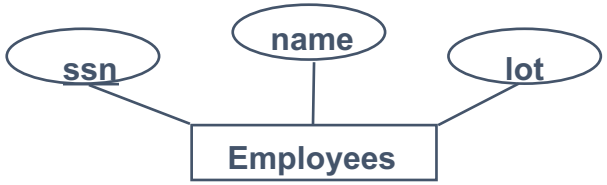
Conceptual Design

- What are the entities and relationships?
  - And what info about E's & R's should be in DB?
- What integrity constraints ("business rules") hold?

- ER diagram is the "schema"
- Can map an ER diagram into a relational schema.

- Conceptual design is where the data engineering begins

ER Model Basics: Entities



- 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.

Entity Sets -- *instructor* and *student*

76766	Crick
45565	Katz
10101	Srinivasan
98345	Kim
76543	Singh
22222	Einstein

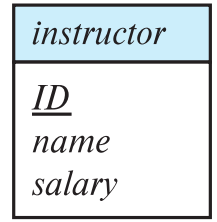
instructor

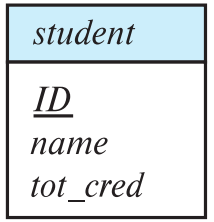
98988	Tanaka
12345	Shankar
00128	Zhang
76543	Brown
76653	Aoi
23121	Chavez
44553	Peltier

student

Representing Entity sets in ER Diagram

- Entity sets can be represented graphically as follows:
  - Rectangles represent entity sets.
  - Attributes listed inside entity rectangle
  - Underline indicates primary key attributes





Relationship Sets

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

  Example:

  44553 (Peltier) — *advisor* — 22222 (Einstein)

  student entity — relationship set — *instructor* entity

- A **relationship set** is a mathematical relation among $n \geq 2$ entities, each taken from entity sets

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

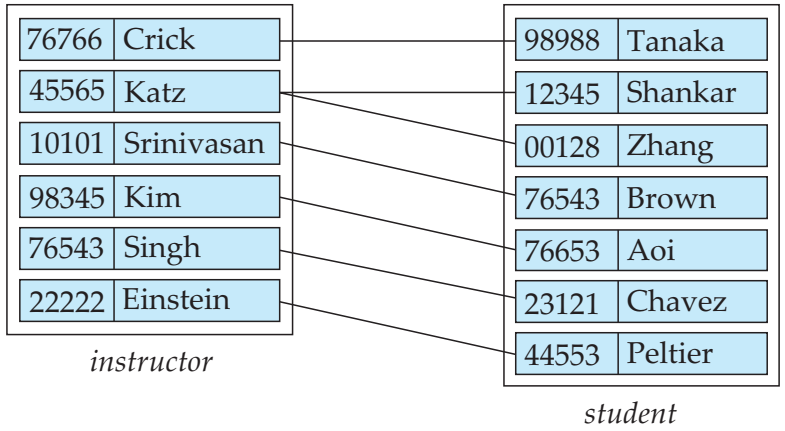
  where $(e_1, e_2, \ldots, e_n)$ is a relationship

  - Example:

    $(44553,22222) \in advisor$

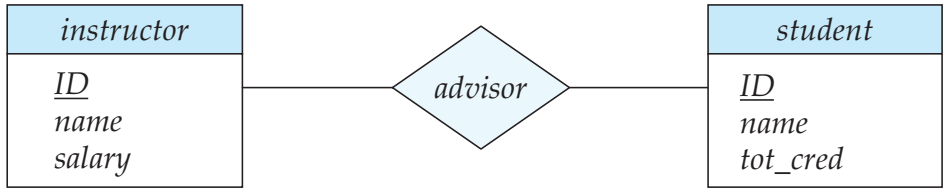
Relationship Sets (Cont.)

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



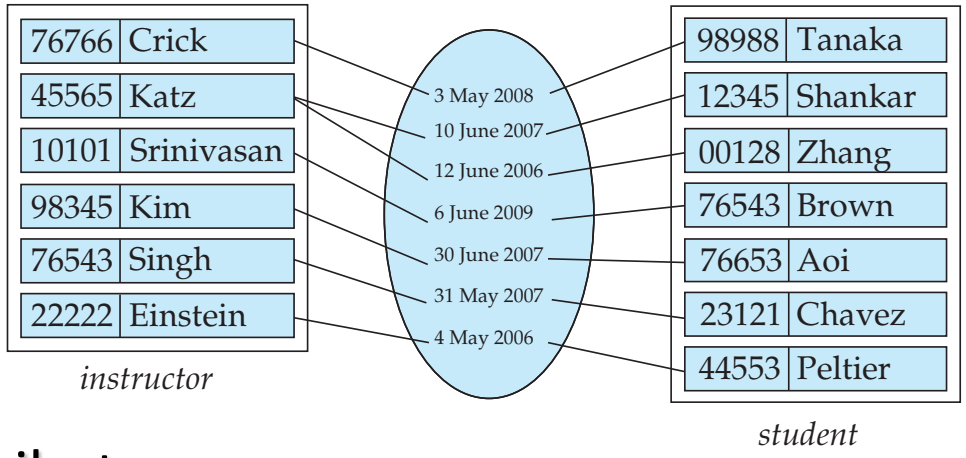
Representing Relationship Sets via ER Diagrams

- Diamonds represent relationship sets.

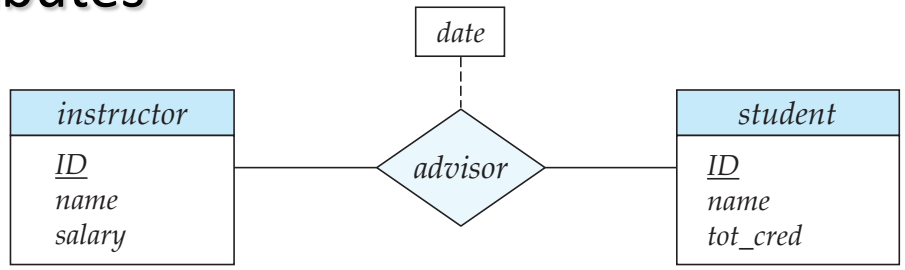


Relationship Sets (Cont.)

- An attribute can also be associated with a relationship set.
- For instance, the *advisor* relationship set between entity sets *instructor* and *student* may have the attribute *date* which tracks when the student started being associated with the advisor

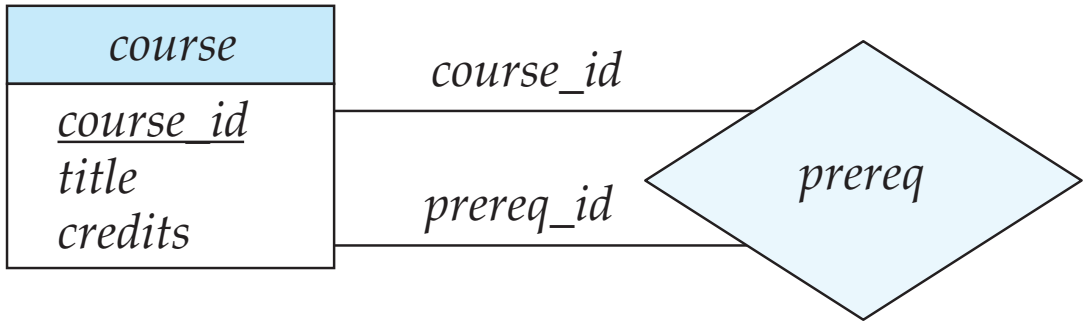


Relationship Sets with Attributes



Roles

- Entity sets of a relationship need not be distinct
  - Each occurrence of an entity set plays a “role” in the relationship
- The labels “*course_id*” and “*prereq_id*” are called **roles**.



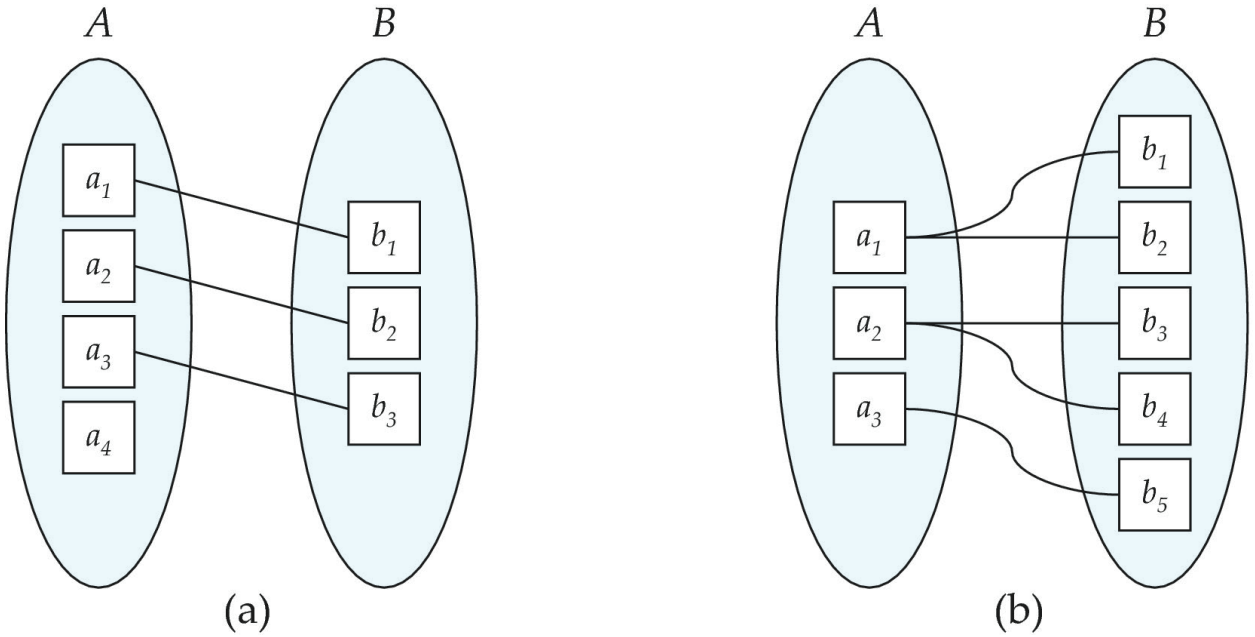
Degree of a Relationship Set

- Binary relationship
  - involve two entity sets (or degree two).
  - most relationship sets in a database system are binary.
- Relationships between more than two entity sets are rare. Most relationships are binary. (More on this later.)
  - Example: *students* work on research *projects* under the guidance of an *instructor*.
  - relationship *proj_guide* is a ternary relationship between *instructor, student,* and *project*

Mapping Cardinality Constraints

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

Mapping Cardinalities

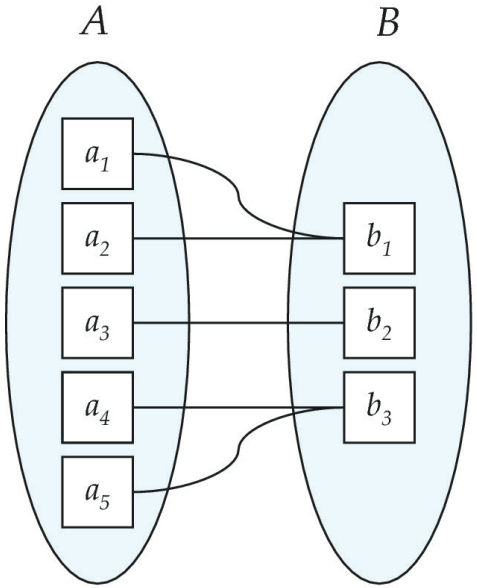


One to one

One to many

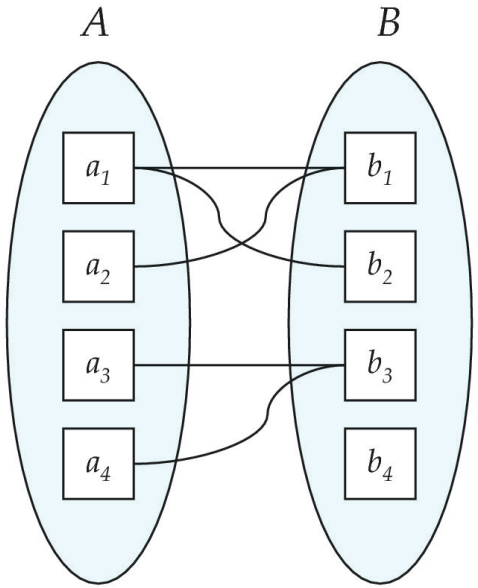
Note: Some elements in *A* and *B* may not be mapped to any elements in the other set

Mapping Cardinalities



(a)

Many to one



(b)

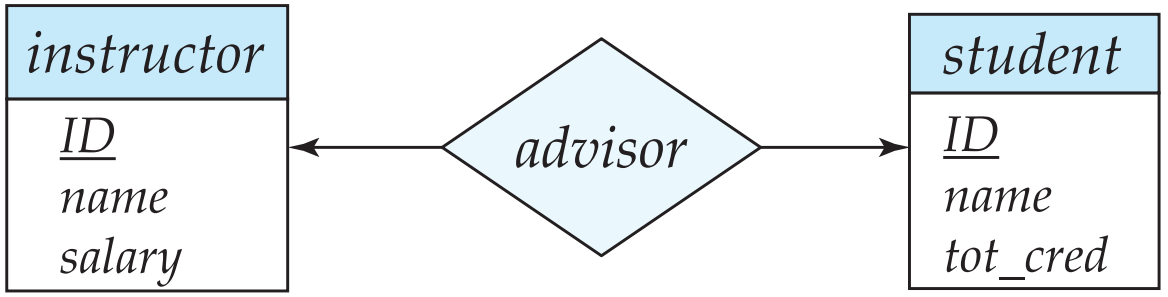
Many to many

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

Representing Cardinality Constraints in ER Diagram

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

- One-to-one relationship between an *instructor* and a *student* :
  - A student is associated with at most one *instructor* via the relationship *advisor*
  - A *student* is associated with at most one *department* via *stud_dept*



Relationships

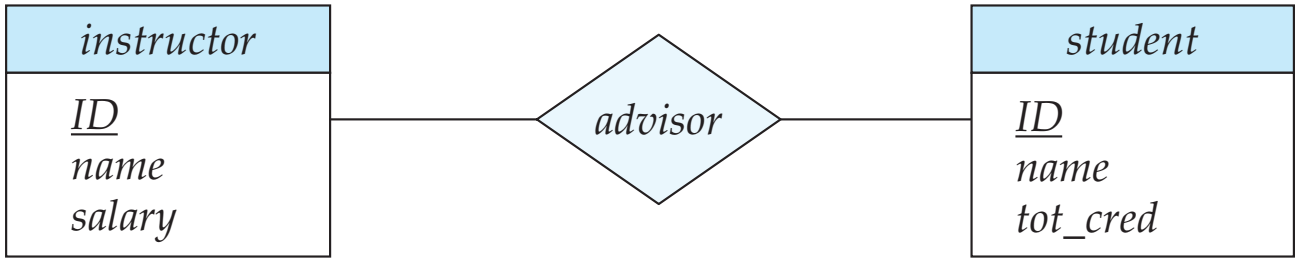
- In a many-to-one relationship between an *instructor* and a *student,*
  - an instructor is associated with at most one student via *advisor*,
  - and a student is associated with several (including 0) instructors via *advisor*

One-to-Many Relationship

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

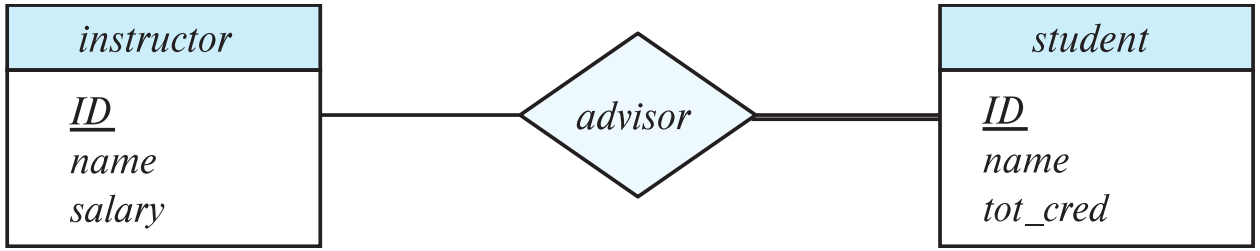
Many-to-Many Relationship

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



Total and Partial Participation

- **Total participation** (indicated by double line): every entity in the entity set participates in at least one relationship in the relationship set



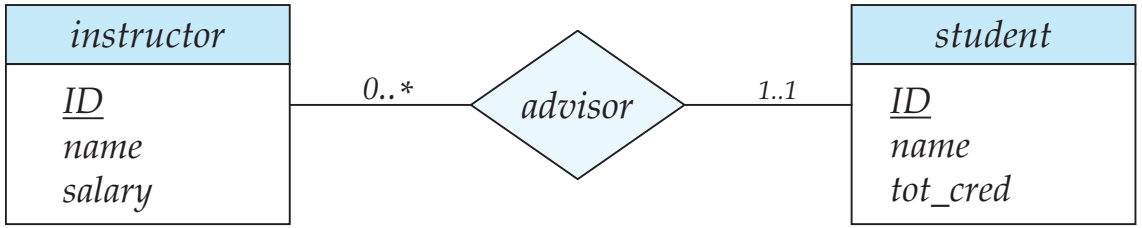
participation of *student* in *advisor* relation is total

  - every *student* must have an associated instructor

- **Partial participation**: some entities may not participate in any relationship in the relationship set
  - Example: participation of *instructor* in *advisor* is partial

Notation for Expressing More Complex Constraints

- A line may have an associated minimum and maximum cardinality, shown in the form *l..h*, where *l* is the minimum and *h* the maximum cardinality
  - A minimum value of 1 indicates total participation.
  - A maximum value of 1 indicates that the entity participates in at most one relationship
  - A maximum value of * indicates no limit.
- Example



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

Primary Key

- Primary keys provide a way to specify how entities and relations are distinguished. We will consider:
  - Entity sets
  - Relationship sets.
  - Weak entity sets

- 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 is consists of the union of the primary keys of entity sets E1, E2, ..En
  - If the relationship set *R* has attributes a1, a2, .., am associated with it, then the primary key of *R* also includes the attributes a1, a2, .., am
- Example: relationship set “advisor”.
  - The primary key consists of *instructor.ID* and s*tudent.ID*
- The choice of the primary key for a relationship set depends on the mapping cardinality of the relationship set.

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 s*ec_course*; however, by doing so the relationship between *section* and *course* becomes implicit in an attribute, which is not desirable.

Weak Entity Sets (Cont.)

- An alternative way to deal with this redundancy is to not store the attribute *course_id* in the *section* entity and to only store the remaining attributes *section_id*, *year*, and *semester.*
  - However, the entity set *section* then does not have enough attributes to identify a particular *section* entity uniquely
- To deal with this problem, we treat the relationship *sec_course* as a special relationship that provides extra information, in this case, the *course_id*, required to identify *section* entities uniquely.
- A **weak entity set** is one whose existence is dependent on another entity, called its **identifying entity**
- Instead of associating a primary key with a weak entity, we use the identifying entity, along with extra attributes called **discriminator** to uniquely identify a weak entity.

Weak Entity Sets (Cont.)

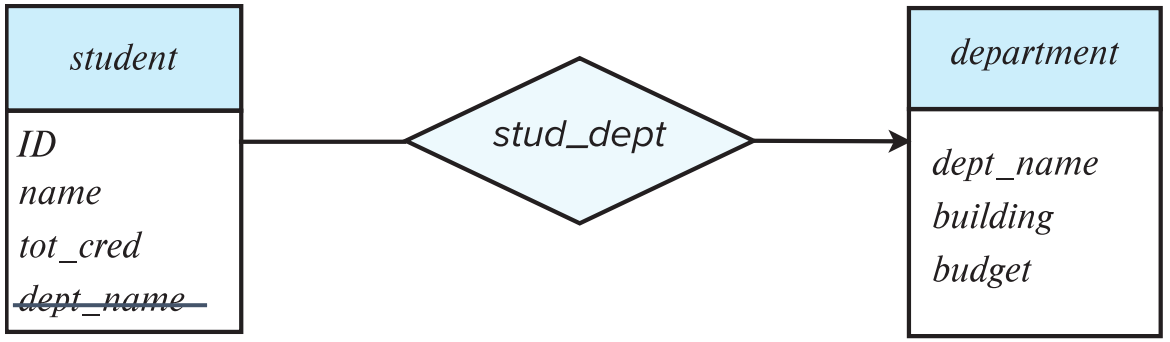
- An entity set that is not a weak entity set is termed a **strong entity set**.
- Every weak entity must be associated with an identifying entity; that is, the weak entity set is said to be **existence dependent** on the identifying entity set.
- The identifying entity set is said to **own** the weak entity set that it identifies.
- The relationship associating the weak entity set with the identifying entity set is called the **identifying relationship**.

Expressing Weak Entity Sets

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

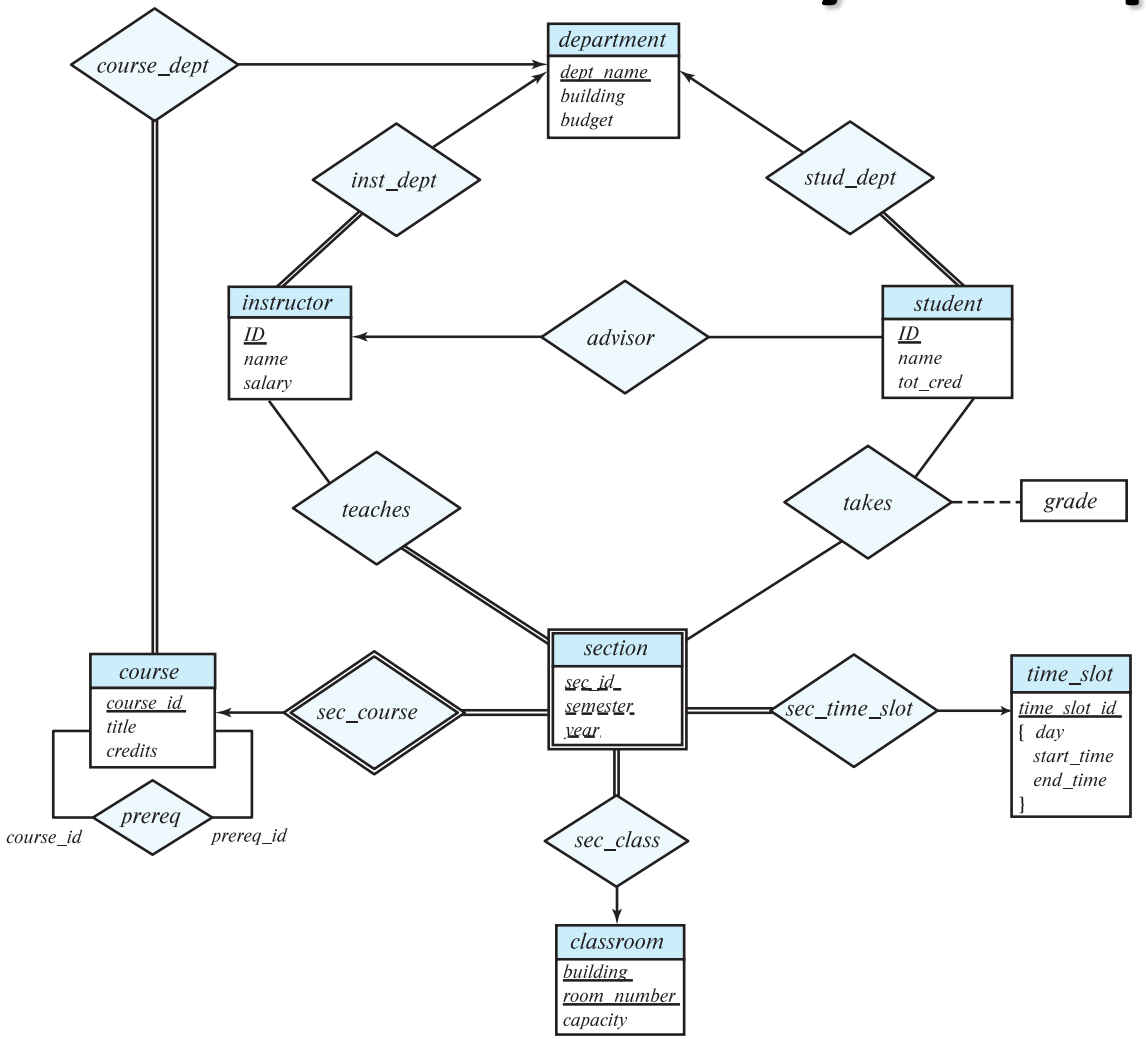
Redundant Attributes

- Suppose we have entity sets:
  - *student*, with attributes: *ID*, *name, tot_cred*, *dept_name*
  - *department,* with attributes: *dept_name, building, budget*
- We model the fact that each student has an associated department using a relationship set *stud_dept*
- The attribute *dept_name* in *student* below replicates information present in the relationship and is therefore redundant
  - and needs to be removed.
- BUT: when converting back to tables, in some cases the attribute gets reintroduced, as we will see later.



(a) Incorrect use of attribute

E-R Diagram for a University Enterprise

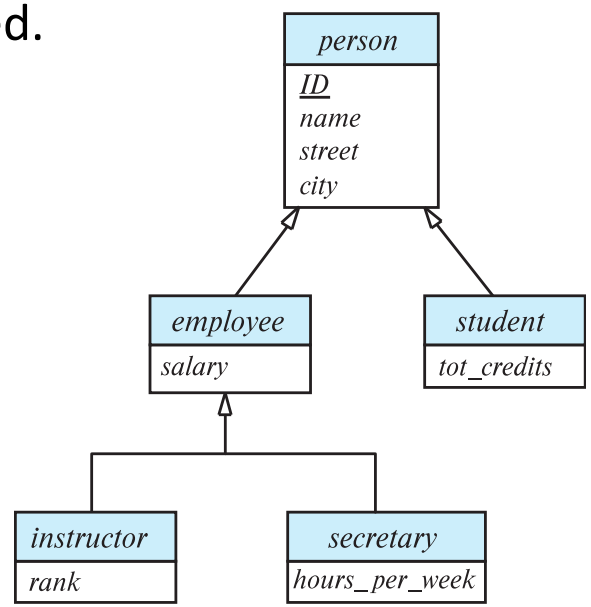


Extended E-R Features

Specialization

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

Overlapping – *employee* and *student*
Disjoint – *instructor* and *secretary*
Total and partial



Representing Specialization via Schemas

- Method 1:
  - Form a schema for the higher-level entity
  - Form a schema for each lower-level entity set, include primary key of higher-level entity set and local attributes

schema	attributes
person	ID, name, street, city
student	ID, tot_cred
employee	ID, salary

  - Drawback: getting information about, an *employee* requires accessing two relations, the one corresponding to the low-level schema and the one corresponding to the high-level schema

Representing Specialization as Schemas (Cont.)

- Method 2:
  - Form a schema for each entity set with all local and inherited attributes

schema	attributes
person	ID, name, street, city
student	ID, name, street, city, tot_cred
employee	ID, name, street, city, salary

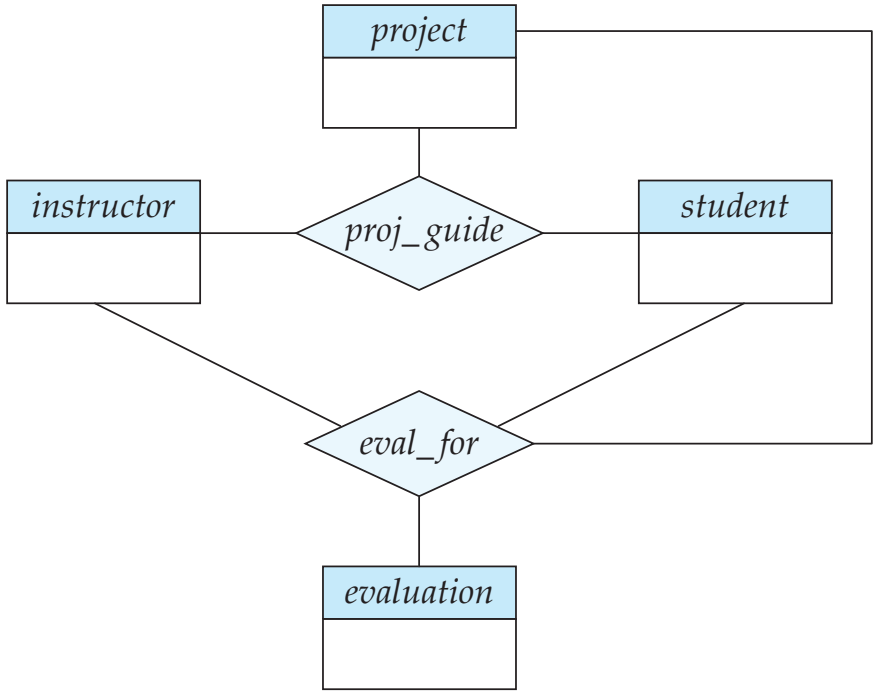
  - Drawback: *name, street* and *city* may be stored redundantly for people who are both students and employees

Generalization

- **A bottom-up design process** – combine a number of entity sets that share the same features into a higher-level entity set.
- Specialization and generalization are simple inversions of each other; they are represented in an E-R diagram in the same way.
- The terms specialization and generalization are used interchangeably.

Aggregation

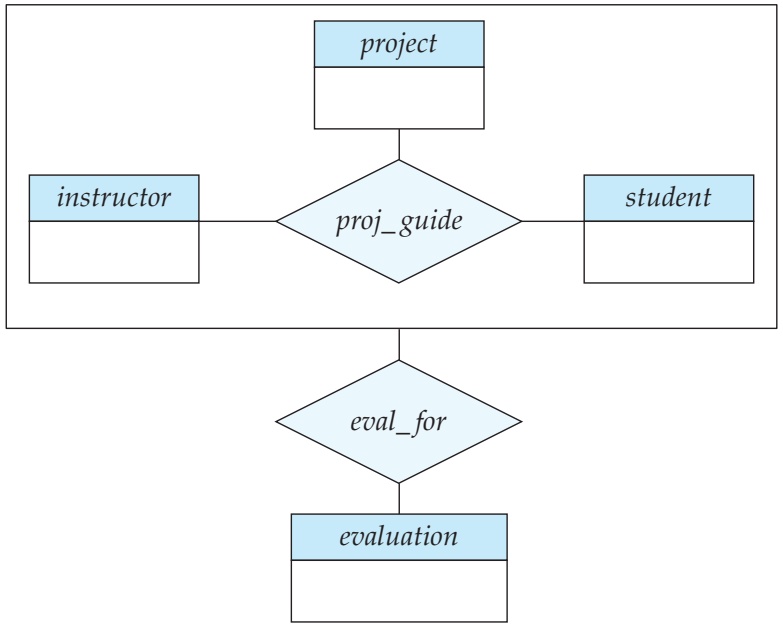
- Consider the ternary relationship *proj_guide*, which we saw earlier
- Suppose we want to record evaluations of a student by a guide on a project



- Relationship sets *eval_for* and *proj_guide* represent overlapping information
  - Every *eval_for* relationship corresponds to a *proj_guide* relationship
  - However, some *proj_guide* relationships may not correspond to any *eval_for* relationships
    - So we can't discard the *proj_guide* relationship
- Eliminate this redundancy via *aggregation*
  - Treat relationship as an abstract entity
  - Allows relationships between relationships
  - Abstraction of relationship into new entity

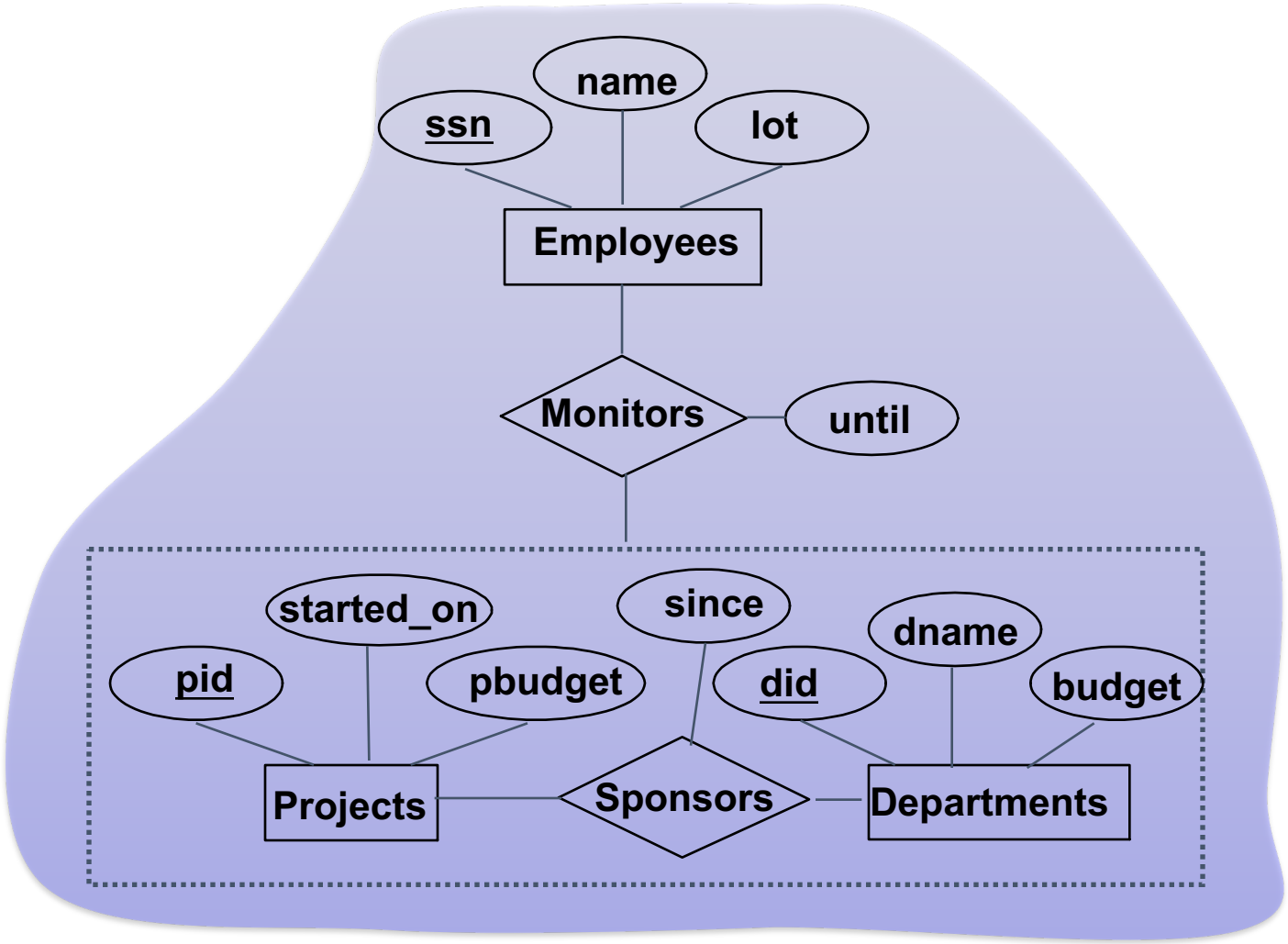
Aggregation (Cont.)

- Eliminate this redundancy via *aggregation* without introducing redundancy, the following diagram represents:
  - A student is guided by a particular instructor on a particular project
  - A student, instructor, project combination may have an associated evaluation

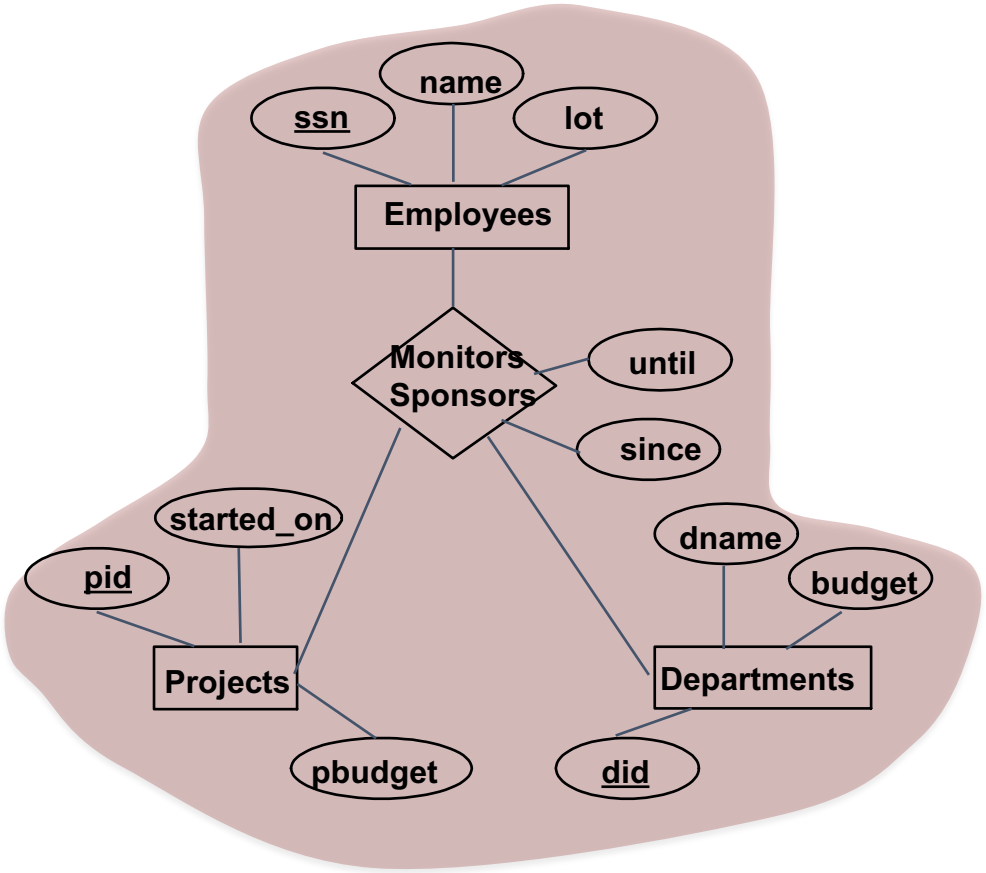


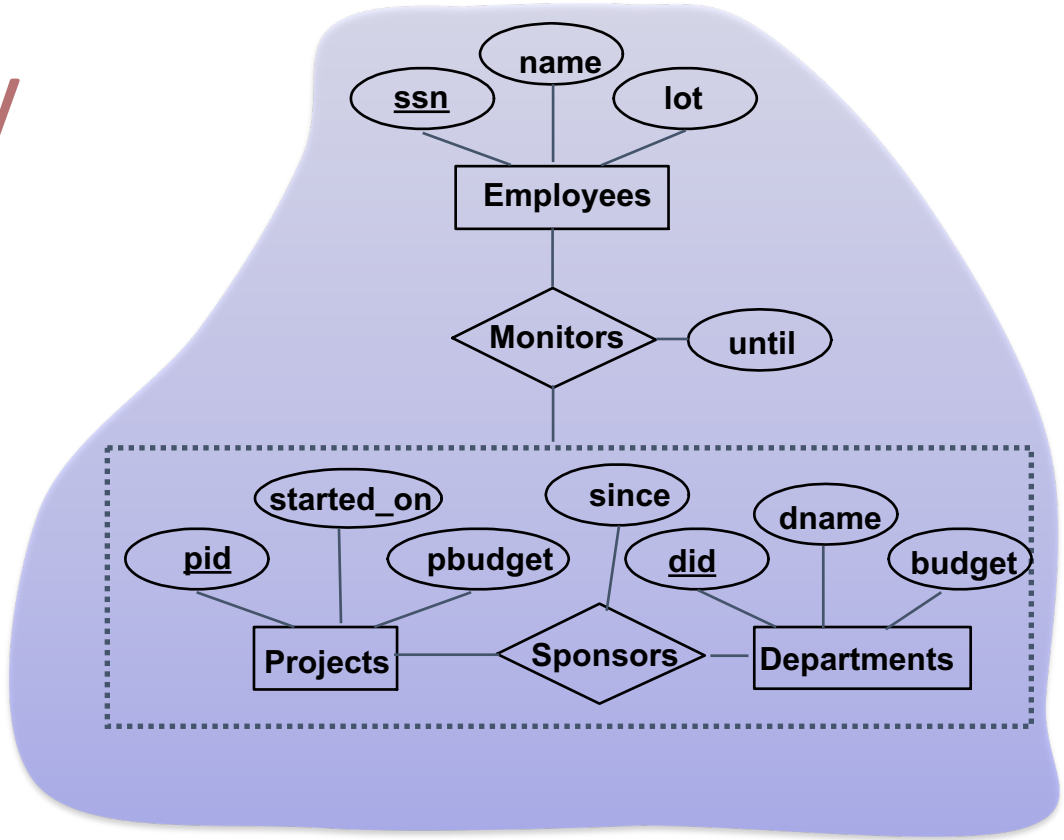
Aggregation

Allows relationships to have relationships.



Aggregation vs. Ternary





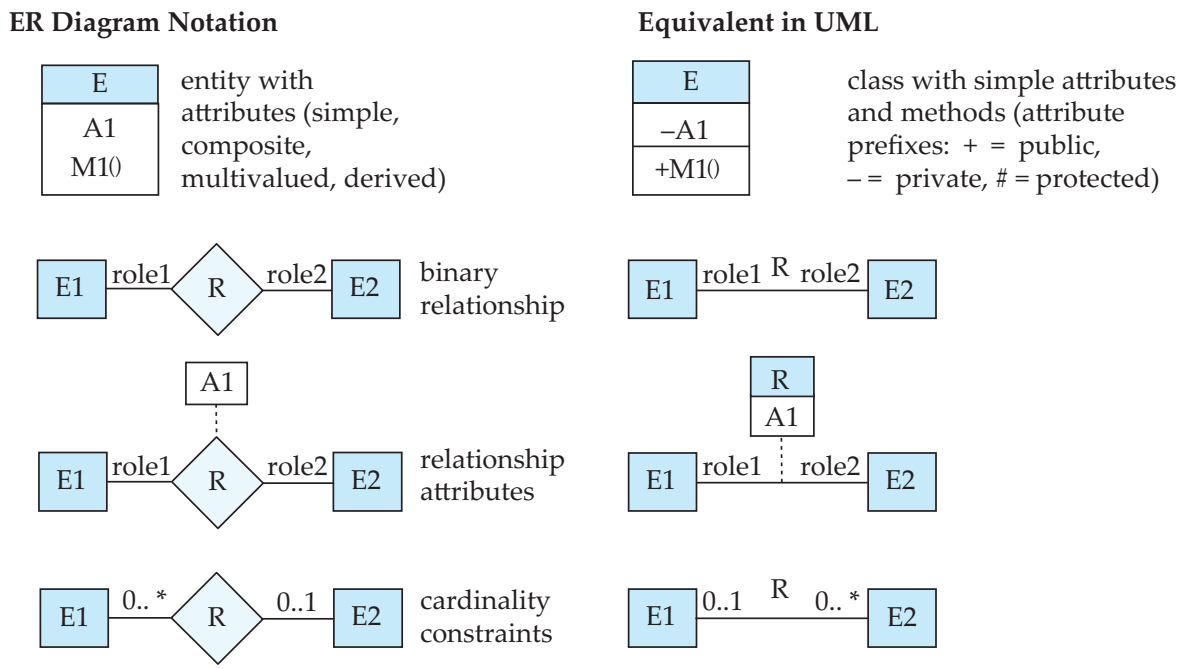
E-R Design Decisions

- The use of an attribute or entity set to represent an object.
- Whether a real-world concept is best expressed by an entity set or a relationship set.
- The use of a ternary relationship versus a pair of binary relationships.
- The use of a strong or weak entity set.
- The use of specialization/generalization – contributes to modularity in the design.
- The use of aggregation – can treat the aggregate entity set as a single unit without concern for the details of its internal structure.

UML

- **UML**: Unified Modeling Language
- UML has many components to graphically model different aspects of an entire software system
- UML Class Diagrams correspond to E-R Diagram, but several differences.

ER vs. UML Class Diagrams



* Note reversal of position in cardinality constraint depiction

Logic Design

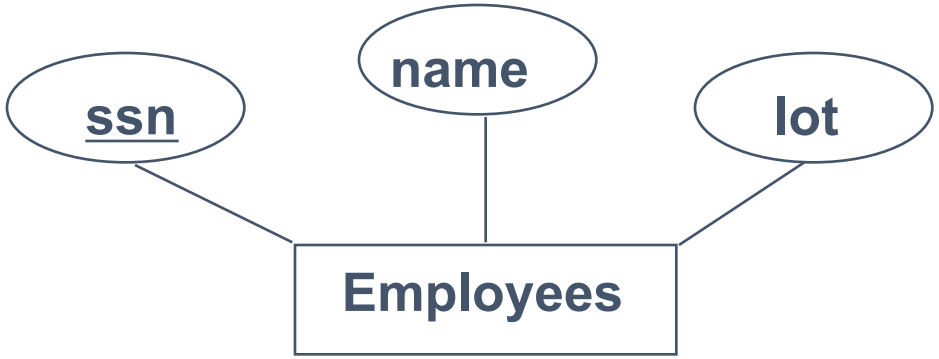
- According to the entities and relationships in ER diagram, define tables and views in target DBMS. Basic standard is 3NF.
  - Translate entities and relationships in ER diagram to tables
  - Naming rule of table and attribute
  - Define the type and domain of every attribute
  - Suitable denormalization
  - Necessary view
  - Consider the tables in legacy system
  - Interface tables

Converting ER to Relational

- Fairly analogous structure
- But many simple concepts in ER are subtle to specify in relations

Logical DB Design: ER to Relational

- Entity sets to tables. Easy.



ssn	name	lot
123-22-3666	Attishoo	48
231-31-5368	Smiley	22
131-24-3650	Smethurst	35

```
CREATE TABLE Employees
  (ssn CHAR(11),
   name CHAR(20),
   lot  INTEGER,
   PRIMARY KEY  (ssn))
```

Relationship Sets to Tables

In translating a **many-to-many** relationship set to a relation, attributes of the relation must include:

1) Keys for each participating entity set (as foreign keys). This set of attributes forms a ***superkey*** for the relation.

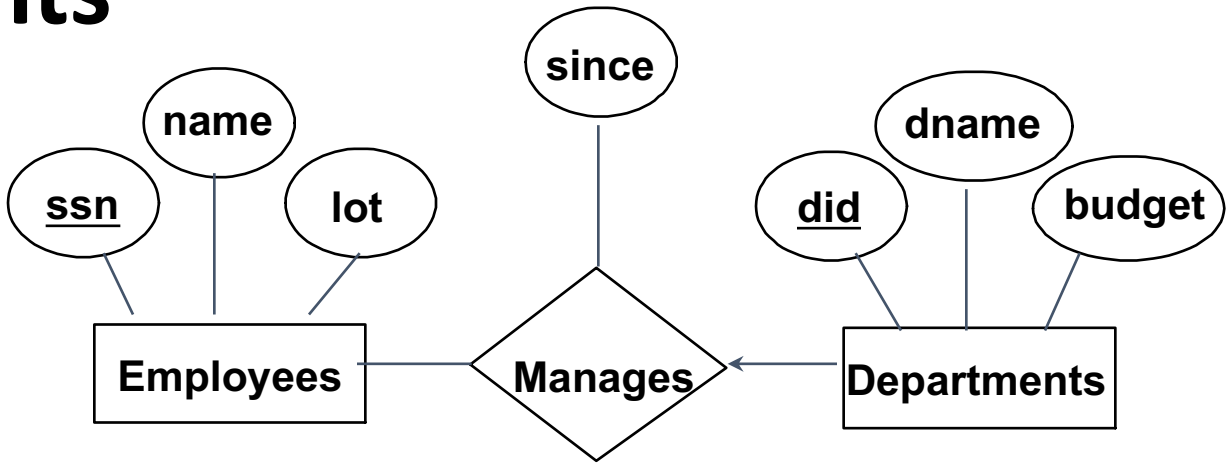
2) All descriptive attributes.

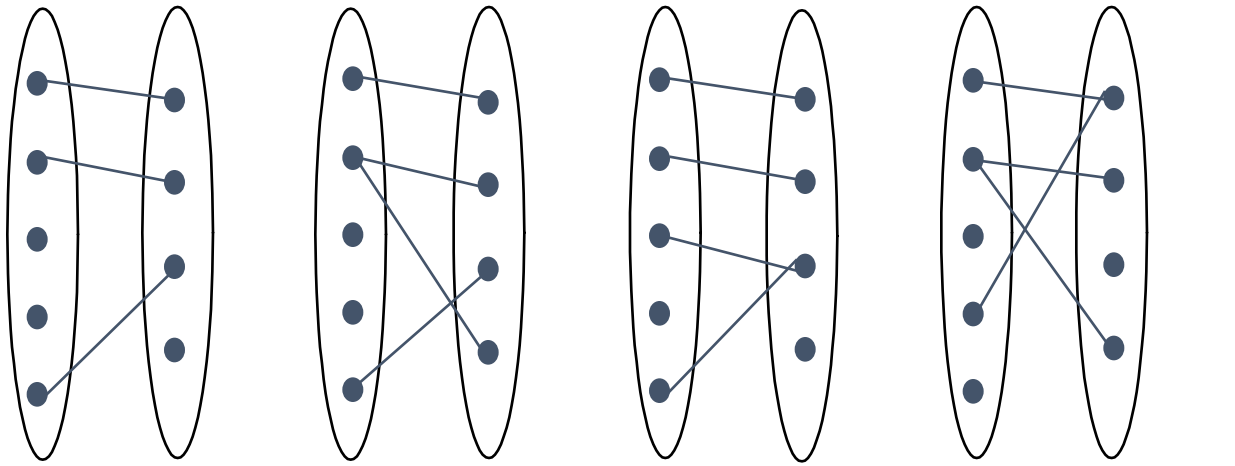
ssn	did	since
123-22-3666	51	1/1/91
123-22-3666	56	3/3/93
231-31-5368	51	2/2/92

```
CREATE TABLE Works_In(
  ssn  CHAR(1),
  did  INTEGER,
  since  DATE,
  PRIMARY KEY (ssn, did),
  FOREIGN KEY (ssn)
    REFERENCES Employees,
  FOREIGN KEY (did)
    REFERENCES Departments)
```

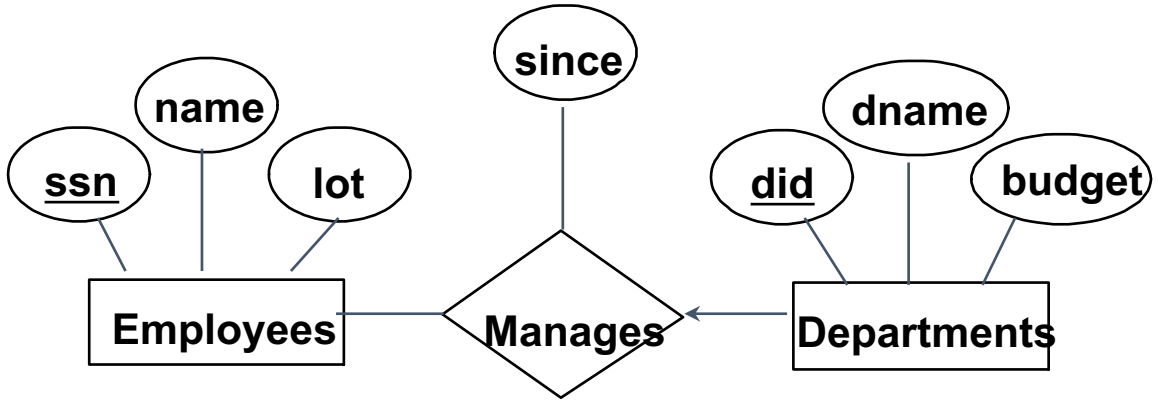
Review: Key Constraints

Each dept has at most one manager, according to the **key constraint** on Manages.



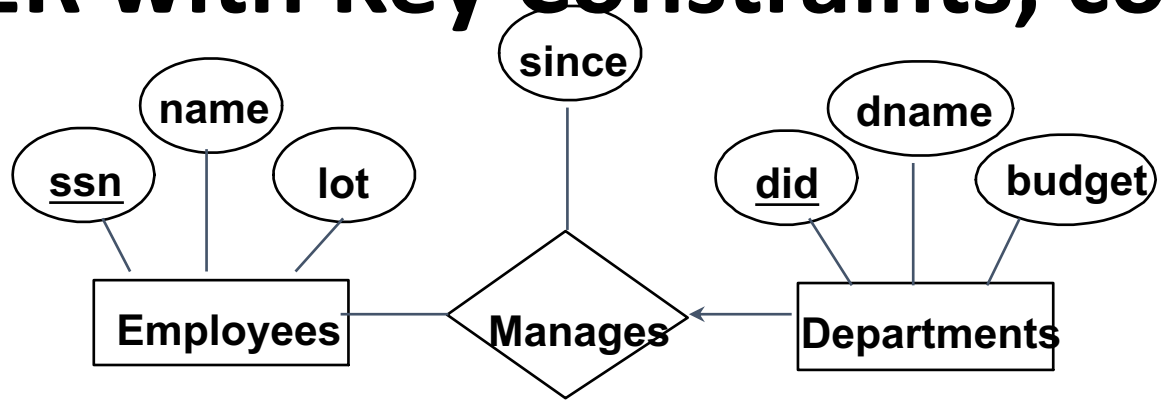


Translating ER with Key Constraints




```
CREATE TABLE  Manages(
 ssn  CHAR(11),
 did  INTEGER,
 since  DATE,
 PRIMARY KEY  (did),
 FOREIGN KEY (ssn)
   REFERENCES Employees,
 FOREIGN KEY (did)
   REFERENCES Departments)
```

Translating ER with Key Constraints, cont




```
CREATE TABLE  Manages(
 ssn  CHAR(11),
 did  INTEGER,
 since  DATE,
 PRIMARY KEY  (did),
 FOREIGN KEY (ssn)
   REFERENCES Employees,
 FOREIGN KEY (did)
   REFERENCES Departments)
```

Vs.

```
CREATE TABLE  Dept_Mgr(
 did  INTEGER,
 dname  CHAR(20),
 budget  REAL,
 ssn  CHAR(11),
 since  DATE,
 PRIMARY KEY  (did),
 FOREIGN KEY (ssn)
   REFERENCES Employees)
```

- Since each department has a unique manager, we could instead combine Manages and Departments.

Translating Weak Entity Sets

- Weak entity set and identifying relationship set are translated into a single table.
  - **When the owner entity is deleted, all owned weak entities must also be deleted.**

```
CREATE TABLE  Dep_Policy (
   pname  CHAR(20),
   age  INTEGER,
   cost  REAL,
   ssn  CHAR(11) NOT NULL,
   PRIMARY KEY  (pname, ssn),
   FOREIGN KEY  (ssn) REFERENCES Employees
      ON DELETE CASCADE)
```

Summary of Conceptual Design

- **Conceptual design** follows requirements analysis
  - Yields a high-level description of data to be stored
- ER model popular for conceptual design
  - Constructs are expressive, close to the way we think about applications.
  - Note: There are many variations on ER model
    - Both graphically and conceptually
- Basic constructs: **entities**, **relationships**, and **attributes** (of entities and relationships).
- Some additional constructs**: weak entities**, ISA hierarchies (see text if you're curious), and aggregation.

Summary of ER (Cont.)

- Basic integrity constraints
  - **key constraints**
  - **participation constraints**
- Some **foreign key** constraints are also implicit in the definition of a relationship set.
- Many other constraints (notably, **functional dependencies**) cannot be expressed.
- Constraints play an important role in determining the best database design for an enterprise.

Summary of ER (Cont….)

- ER design is **subjective**. Many ways to model a given scenario!
- Analyzing alternatives can be tricky! Common choices include:
  - Entity vs. attribute, entity vs. relationship, binary or n-ary relationship, whether or not to use aggregation
- For good DB design: resulting relational schema should be analyzed and refined further.
  - Functional Dependency information
    + normalization coming in subsequent lecture.

Physical Design

- For relational database, the main task in this phase is to consider creating necessary indexes according to the processing requirements, including single attribute indexes, multi attributes indexes, cluster indexes, etc. Generally, the attribute often as query conditions should have index.
- **Other problems:**
  - Partition design
  - Stored procedure
  - Trigger
  - Integrity constraints