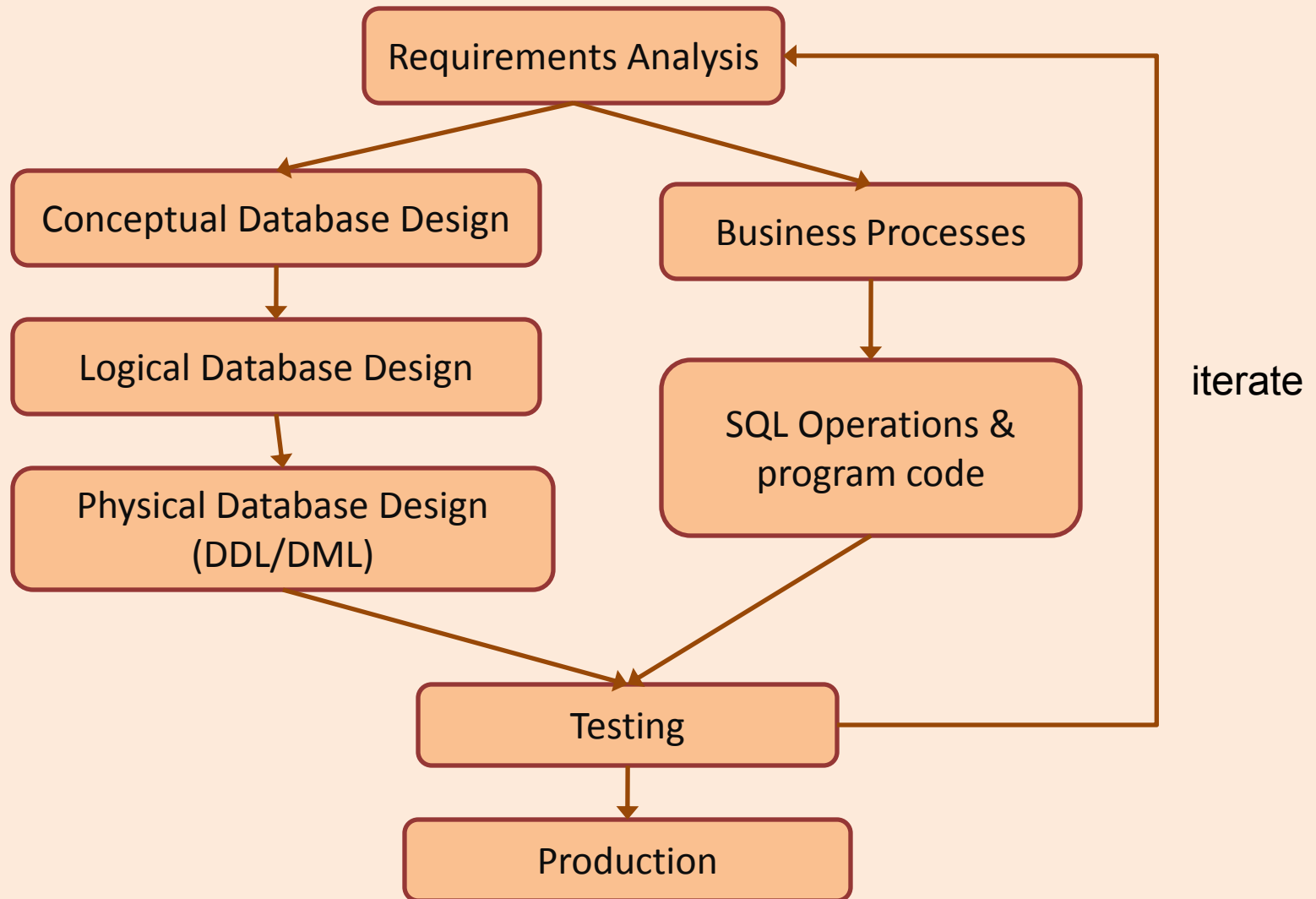


ICS 321 Fall 2013

# High Level Database Models

Asst. Prof. Lipyeow Lim  
Information & Computer Science Department  
University of Hawaii at Manoa

# Database Design & Deployment

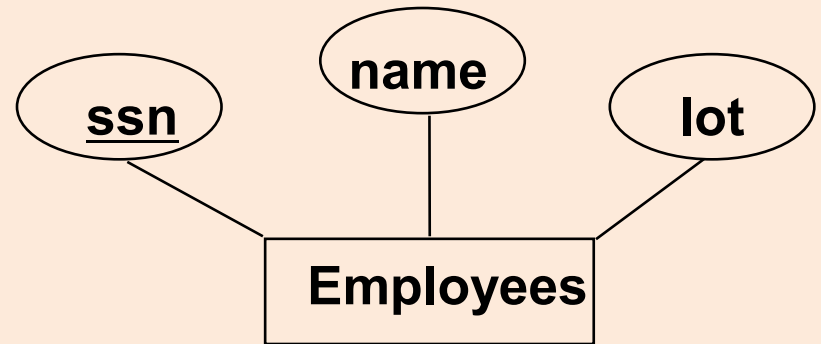


# Overview Database Design

- Conceptual Design
  - Use **entity-relationship** (aka ER) model represented pictorially as ER diagrams
  - Map ER model to relational schema
- Questions to ask yourself
  - What are the **entities** and **relationships** in the application?
  - What information about these entities and relationships should we store in the database?
  - What are the integrity constraints or business rules that hold?

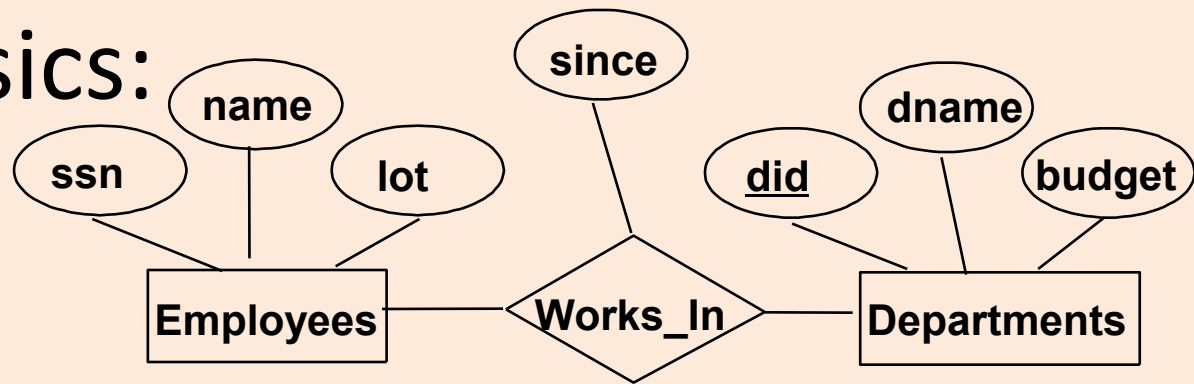
# ER Model Basics:

## Entities



- **Entity**: Real-world object distinguishable from other objects. An entity is described (in DB) using a set of **attributes**.
- **Entity Set**: A collection of similar entities. E.g., all employees.
  - All entities in an entity set have the same set of attributes. (Until we consider ISA hierarchies, anyway!)
  - Each entity set has a **key**.
  - Each attribute has a **domain**.

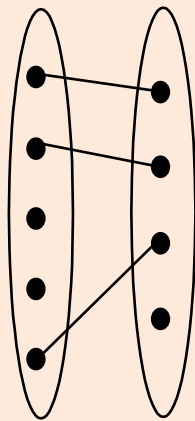
# ER Model Basics: Relationships



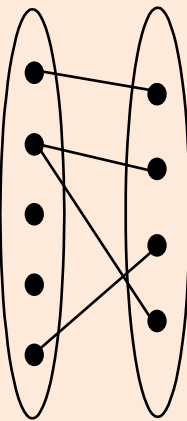
- *Relationship*: Association among two or more entities.
- *Relationship Set*: Collection of similar relationships.
  - An n-ary relationship set  $R$  relates  $n$  entity sets  $E_1 \dots E_n$ ; each relationship in  $R$  involves entities  $e_1 \in E_1, \dots, e_n \in E_n$
  - Same entity set could participate in different relationship sets, or in different “roles” in same set.

# Cardinality Ratios of Relationships

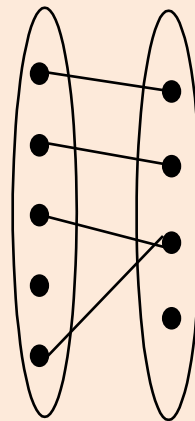
- Consider binary relationships, i.e., between two entity sets
- Alternate notation: 1:1, 1:M, M:1, M:N



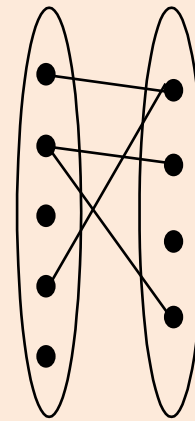
1-to-1



1-to Many

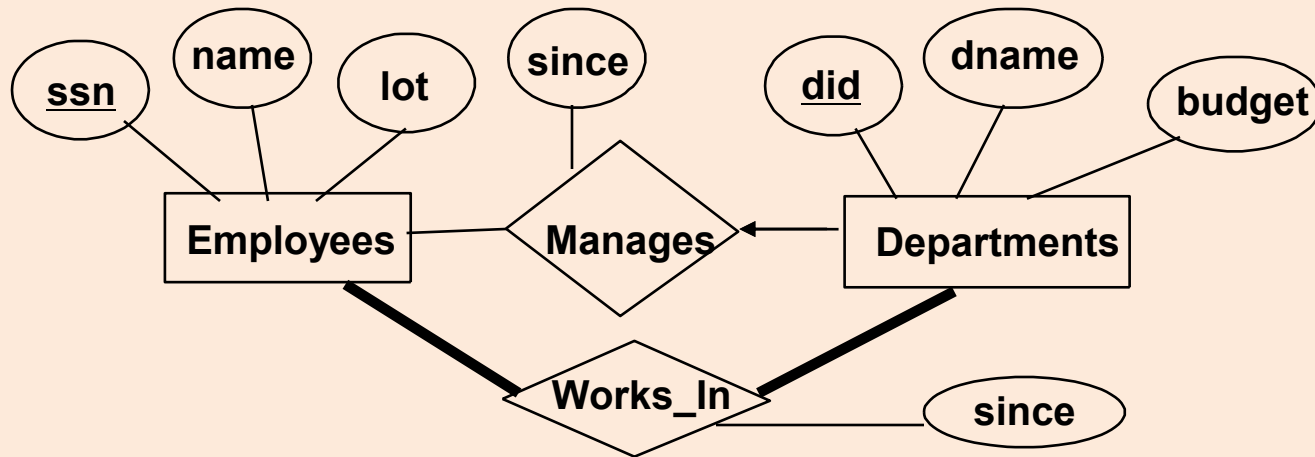


Many-to-1



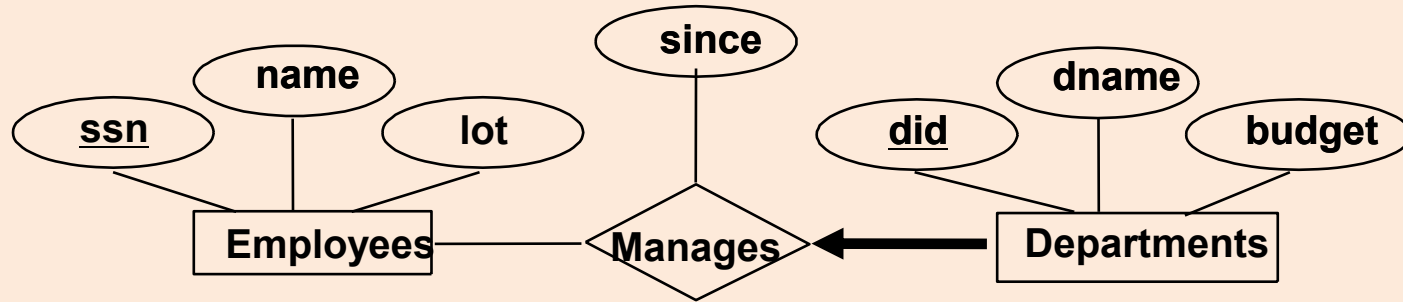
Many-to-Many

# Key Constraints



- Consider Works\_In: An employee can work in many depts; a dept can have many employees : m-to-m
- Consider Manages: each dept has at most one manager
- Dept has a key constraint on Manages: each instance of dept appears in at most one instance of manages
- Denoted by an arrow: given a dept entity we can uniquely identify the manages relationship in which it appears

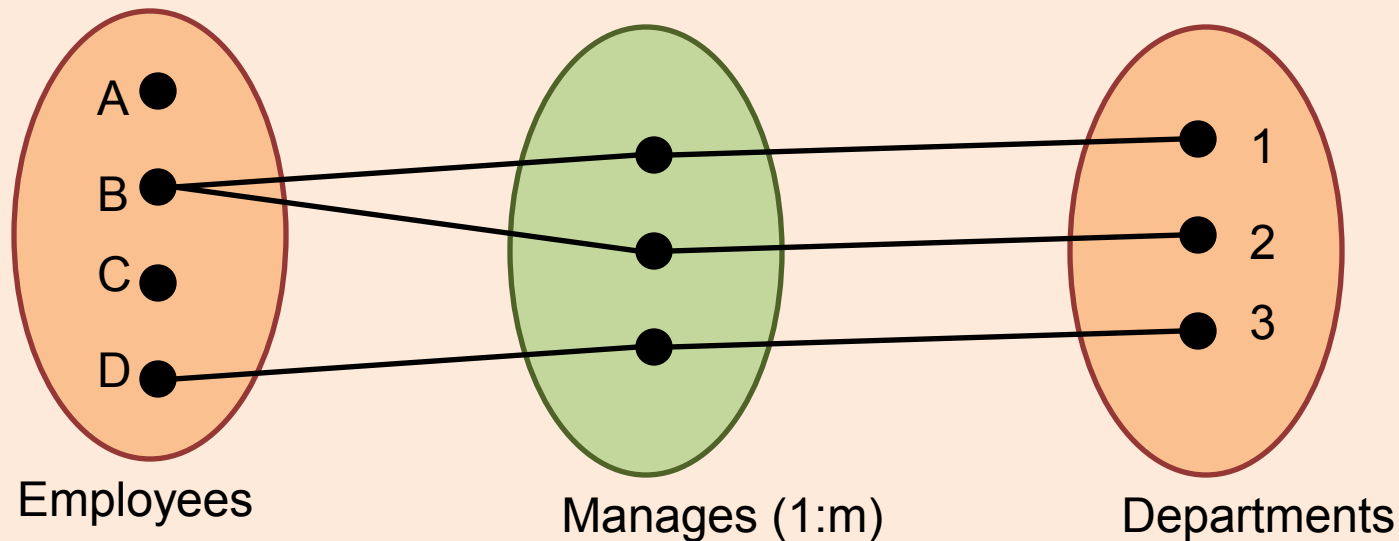
# Participation constraints



- Does every dept have a manager?
- If so, this is a participation constraint: the participation of dept in Manages is said to be *total* (*vs. partial*). Denoted by thick/double line
- Meaning that every Dept entity must appear in an instance of the Manages relationship

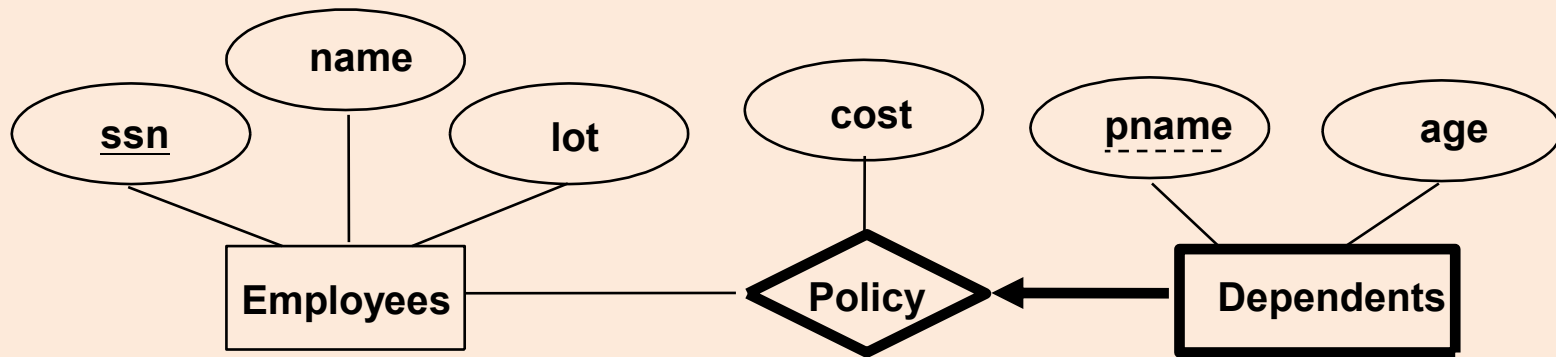


# Set Theoretic Formulation



- **Partial Participation**: Not all members of the Employees entity set take part in the manages relations
- **Total Participation**: All members of the Dept entity set take part in the manages relationship
- Dept has a **key constraint** on Manages: each member of the dept entity set takes part in at most one member of the manages relationship set

# Weak Entities

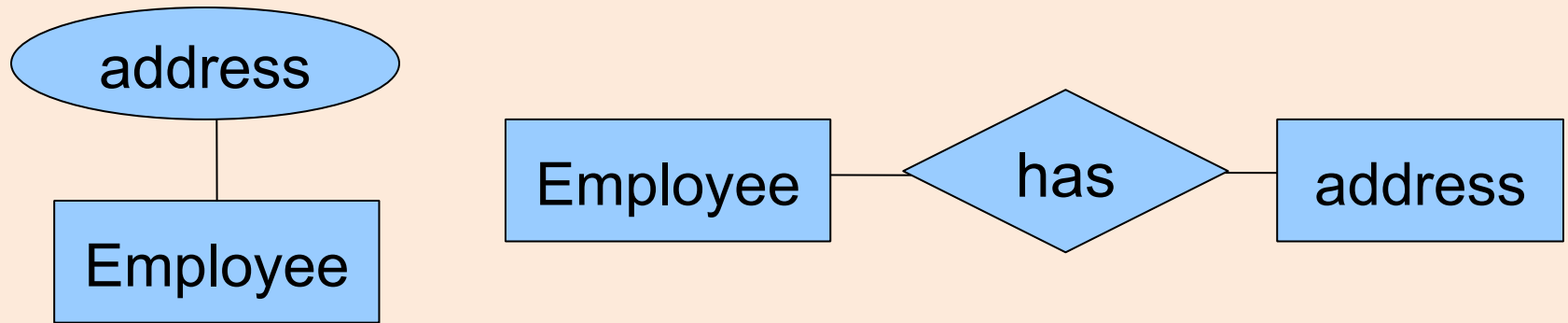


- A *weak entity* can be identified uniquely only by considering the primary key of another (*owner*) entity.
- Owner entity set and weak entity set must participate in a one-to-many relationship set (one owner, many weak entities).
- Weak entity set must have total participation in this *identifying* relationship set.
- Denoted by a box with double or thick lines

# Design Choices

- Should a concept be modeled as an entity or an attribute?
- Should a concept be modeled as an entity or a relationship?
- Identifying relationships: Binary or ternary?  
Aggregation?
- How much semantics to capture in the form of constraints ?

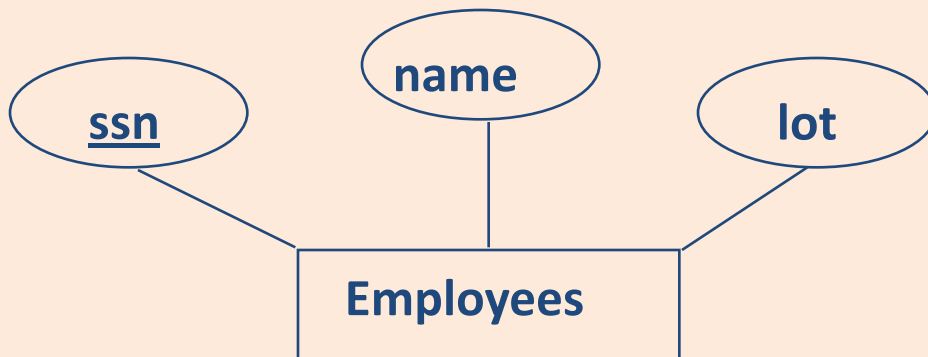
# Entity vs. Attribute



- Depends upon how we want to use the address information, and the semantics of the data:
  - If we have several addresses per employee, *address* must be an entity (since attributes cannot be set-valued).
  - If the structure (city, street, etc.) is important, e.g., we want to retrieve employees in a given city, *address* must be modeled as an entity (since attribute values are atomic).

# Logical DB Design: ER to Relational

- Entity sets to tables:

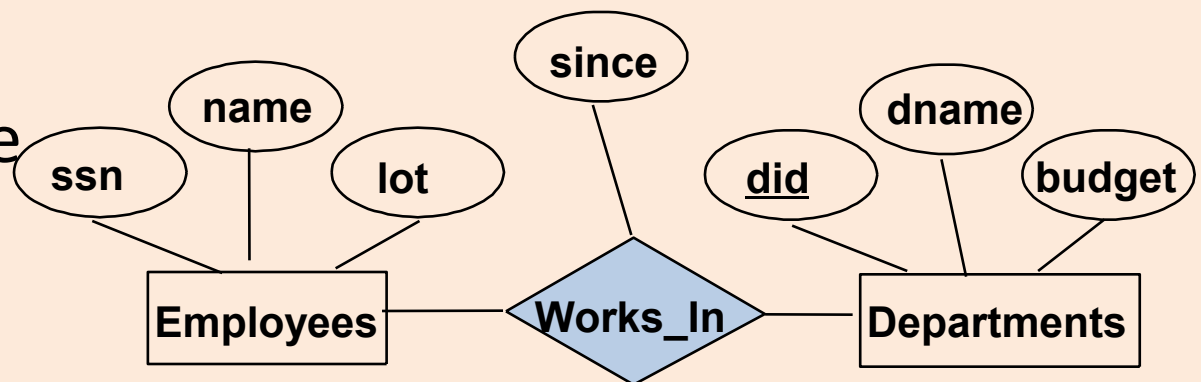


```
CREATE TABLE Employees  
  (ssn CHAR(11),  
   name CHAR(20),  
   lot INTEGER,  
   PRIMARY KEY (ssn))
```

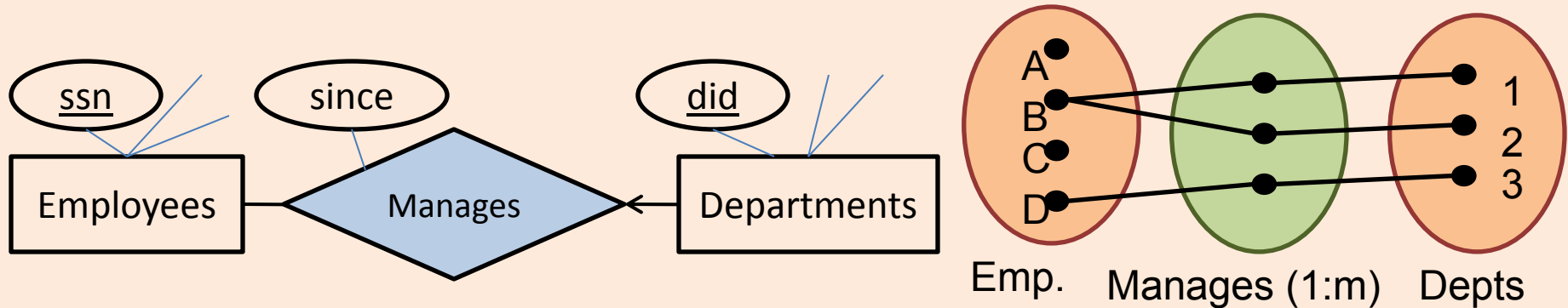
# Relationship Sets to Tables

- Attributes of the relation must include:
  - Keys for each participating entity set (as foreign keys).
    - This set of attributes forms a *superkey* for the relation.
  - All descriptive attributes.

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



# Translating ER Diagrams with Key Constraints



- Map relationship to a table:
  - Note that **did** is the key now!
- Since each department has a unique manager, we could instead combine Manages and Departments.

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

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

# Participation Constraints in SQL

- We can capture participation constraints involving one entity set in a binary relationship, but little else (without resorting to CHECK constraints).

```
CREATE TABLE Dept_Mgr(  
  did INTEGER,  
  dname CHAR(20),  
  budget REAL,  
  ssn CHAR(11) NOT NULL,  
  since DATE,  
  PRIMARY KEY (did),  
  FOREIGN KEY (ssn) REFERENCES Employees,  
  ON DELETE NO ACTION)
```



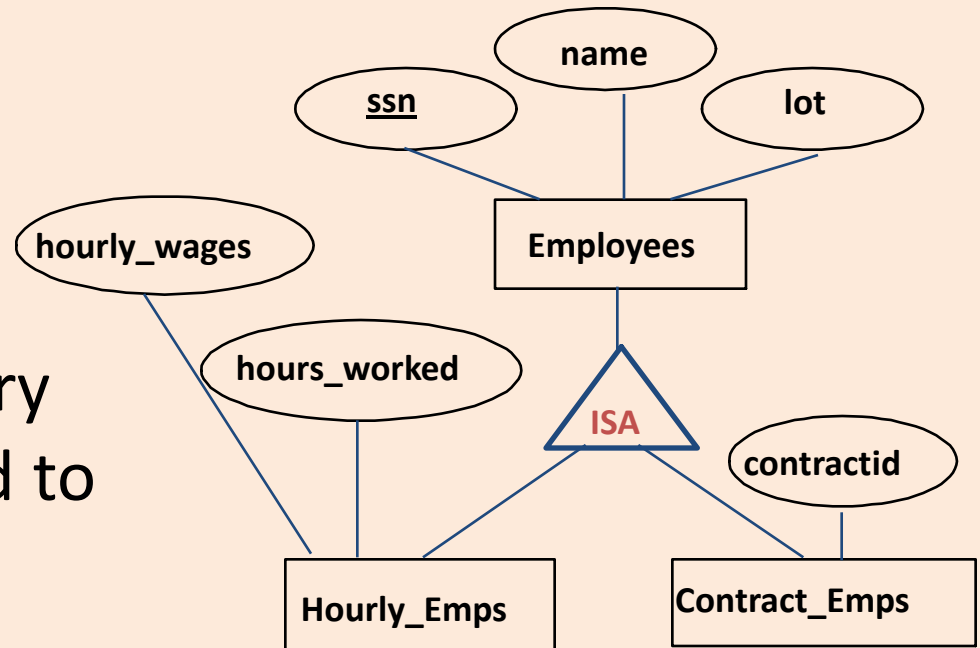
# Translating Weak Entity Sets

- 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)
```

# ISA Hierarchies

- As in C++, or other PLs, attributes are inherited.
- If we declare A **ISA** B, every A entity is also considered to be a B entity.



- *Overlap constraints*: Can Joe be an Hourly\_Emps as well as a Contract\_Emps entity? (*Allowed/disallowed*)
- *Covering constraints*: Does every Employees entity also have to be an Hourly\_Emps or a Contract\_Emps entity? (*Yes/no*)

# Translating ISA Hierarchies to Relations

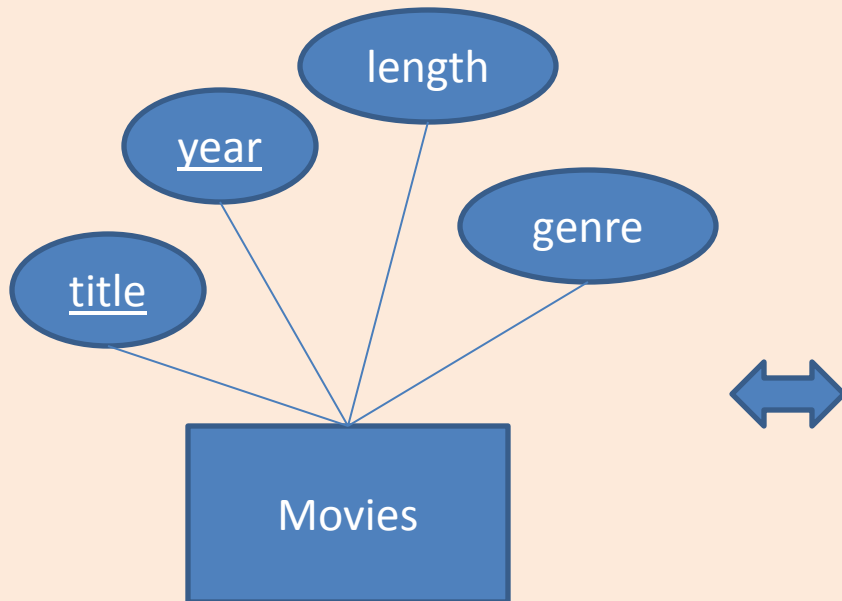
- **General approach:**
  - 3 relations: Employees, Hourly\_Emps and Contract\_Emps.
    - *Hourly\_Emps*: Every employee is recorded in Employees. For hourly emps, extra info recorded in Hourly\_Emps (*hourly\_wages*, *hours\_worked*, *ssn*); must delete Hourly\_Emps tuple if referenced Employees tuple is deleted).
    - Queries involving all employees easy, those involving just Hourly\_Emps require a join to get some attributes.
- **Alternative: Just Hourly\_Emps and Contract\_Emps.**
  - *Hourly\_Emps*: *ssn*, *name*, *lot*, *hourly\_wages*, *hours\_worked*.
  - Each employee must be in one of these two subclasses.

# Unified Modeling Language

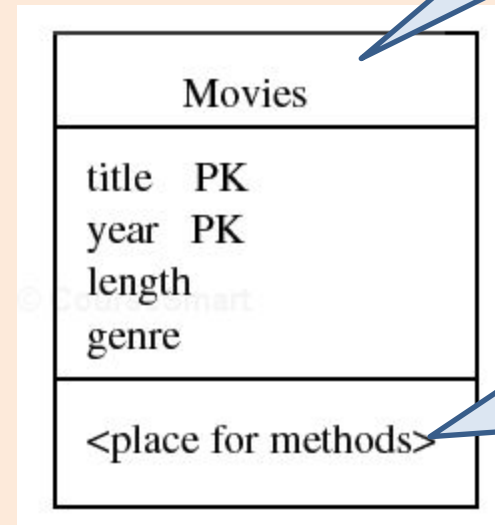
- Standardized general-purpose modeling language for software design
- Based on object-oriented model
- Class diagrams

UML	E/R Model
Class	Entity set
Association	Binary relationship
Association Class	Attributes on a relationship
Subclass	Isa hierarchy
Aggregation	Many-one relationship
Composition	Many-one relationship with referential integrity

# UML Classes



ER Entity Set

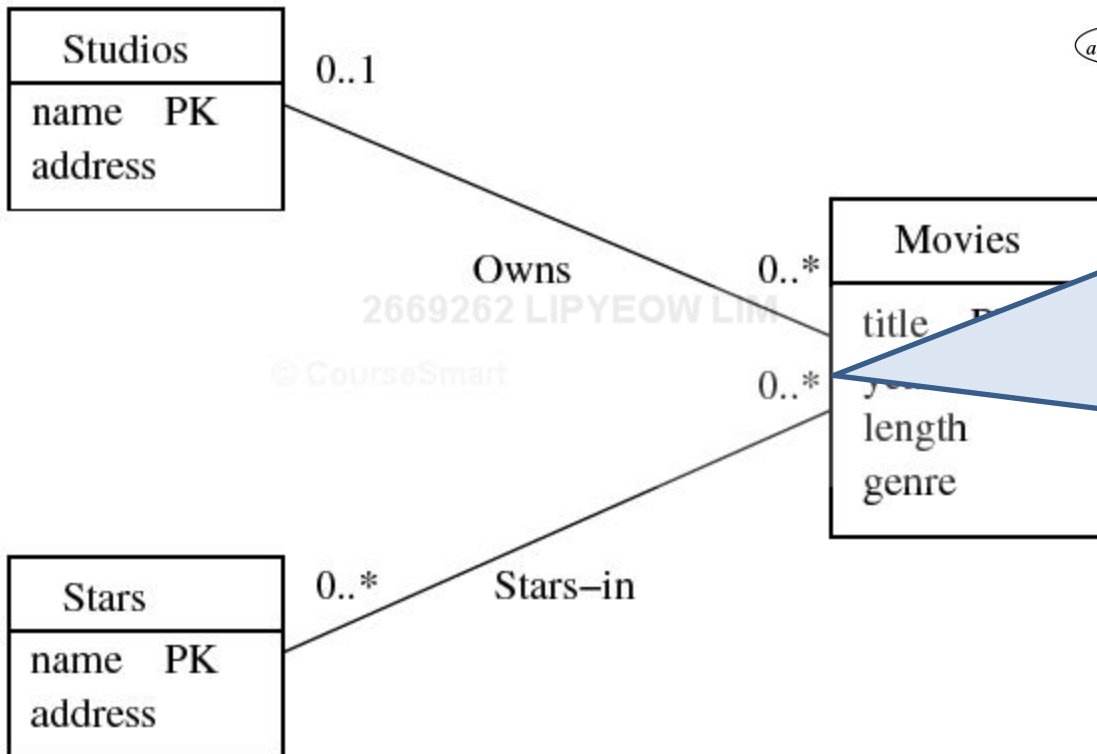
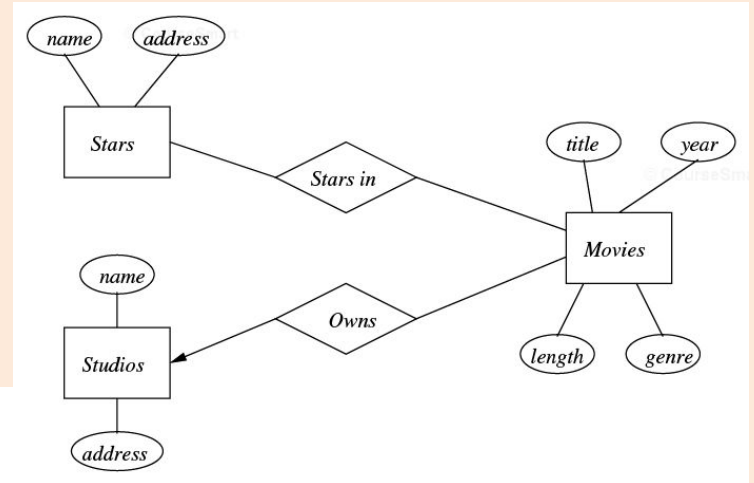


Class name

Methods section typically not used in data modeling

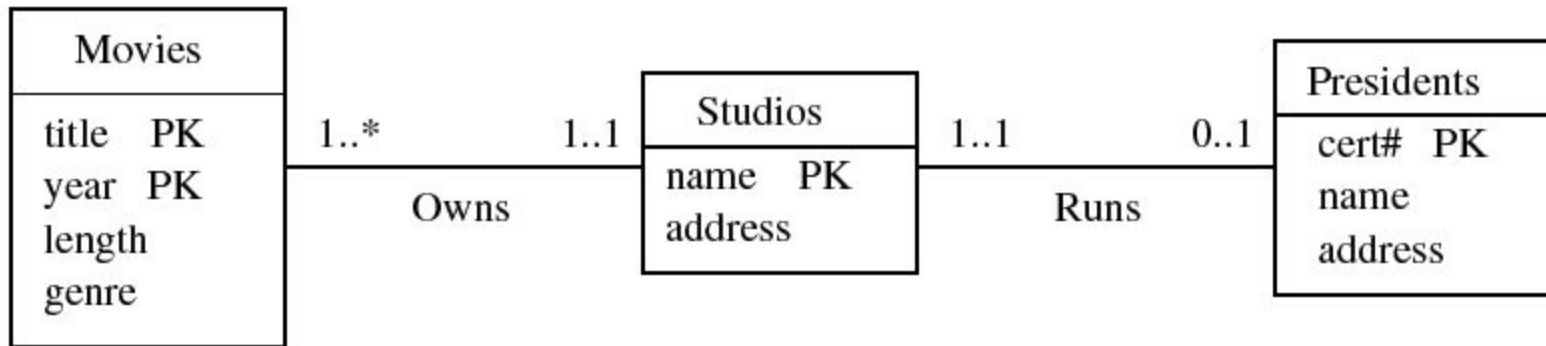
UML Class

# Associations



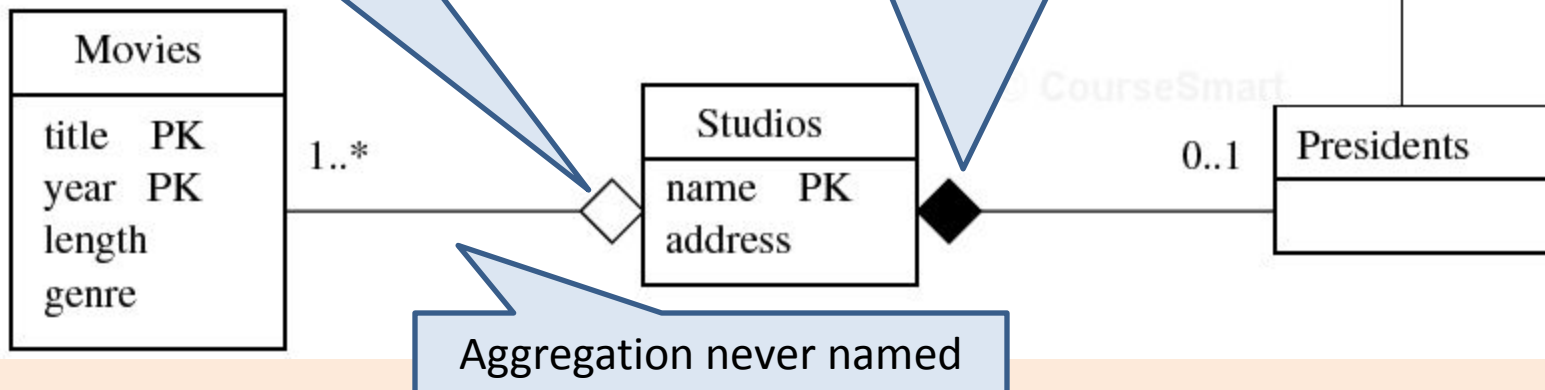
Cardinality constraints : one instance of Stars can be connected to at least 0 instance of movies and at most infinite instances of movies

# Referential Integrity

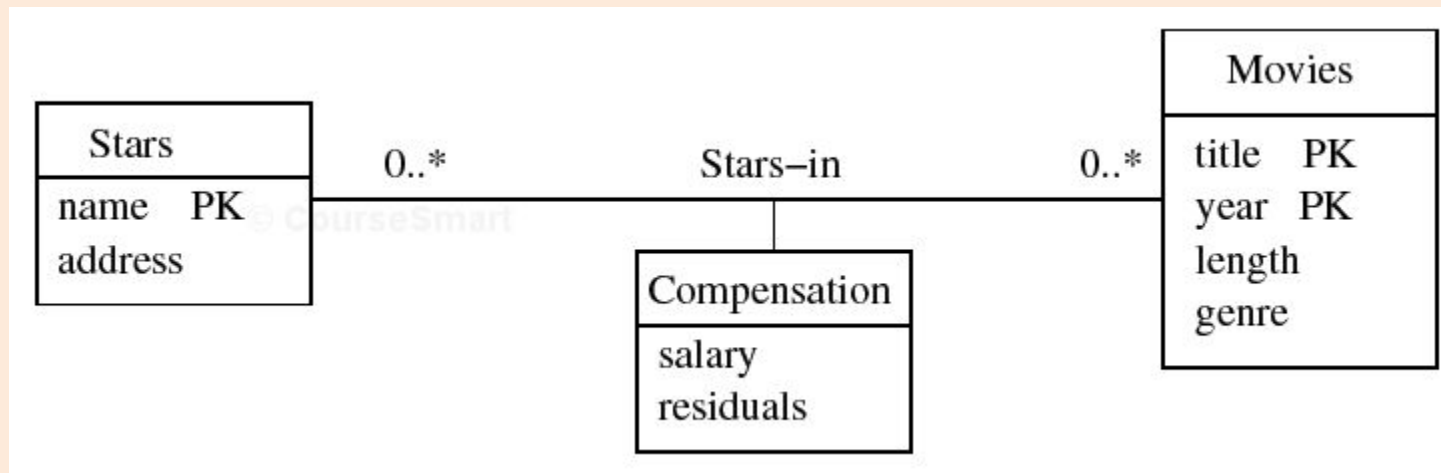


Aggregation: Must be 0..1 (includes 1..1)

Composition : Must be 1..1  
Every president runs exactly one studio

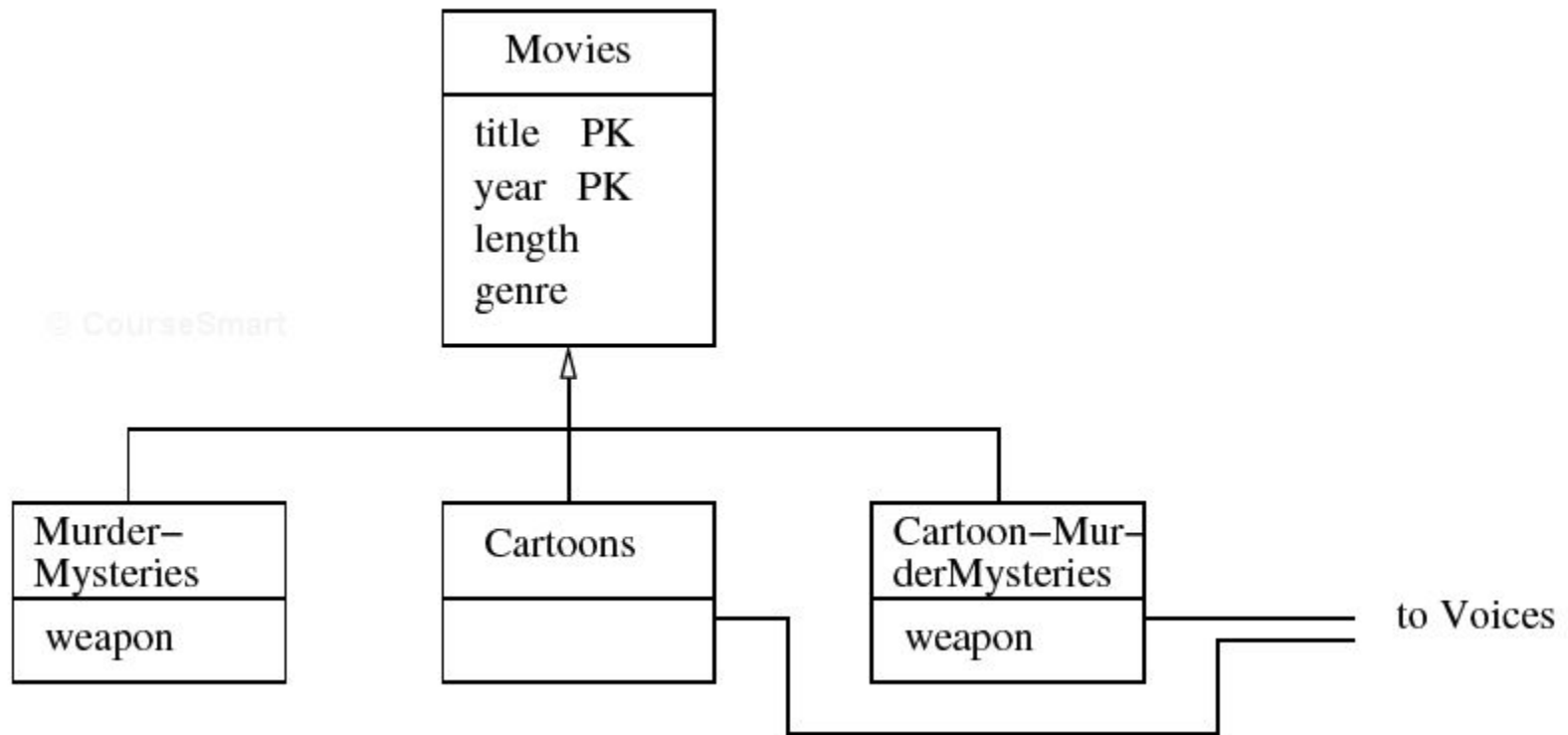


# Association Classes





# Sub-Class Hierarchies



# Modeling Tips

- Faithful to the semantics of the application
- Model only what is needed in the application
- Minimize redundancy (why?)
- Simple is good
- If the model is getting too complicated, take a step back and ask
  - Am i conceptualizing the right entities ?
  - Am i thinking of the right relationships ?
  - Should some relationships become entities ? Vice versa ?
  - Should some attributes become entities ? Vice versa ?