Chapter 7: Database Design and E-R Model

Mapping to Relational Model
&
Extended Features

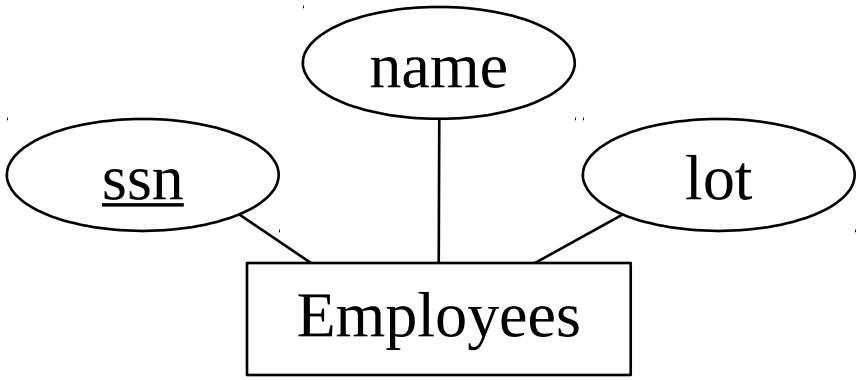
Reduction to Relational Schemas

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.

Logical DB Design: ER to Relational

- Entity sets to tables:

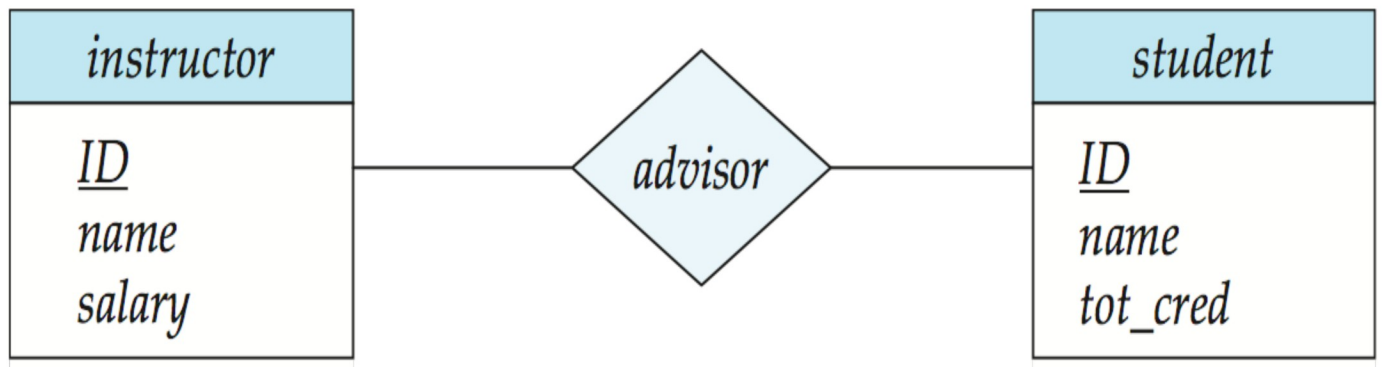



```
CREATE TABLE Employees
        (ssn CHAR(11),
        name CHAR(20),
        lot  INTEGER,
        PRIMARY KEY  (ssn))
```

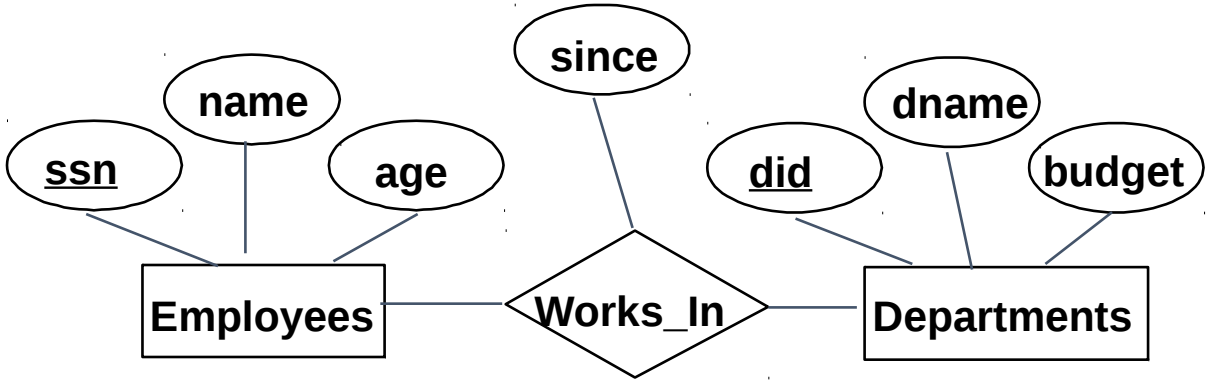
Representing Relationship Sets

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

  advisor = (*s_id, i_id*)



Relationship Sets to Tables



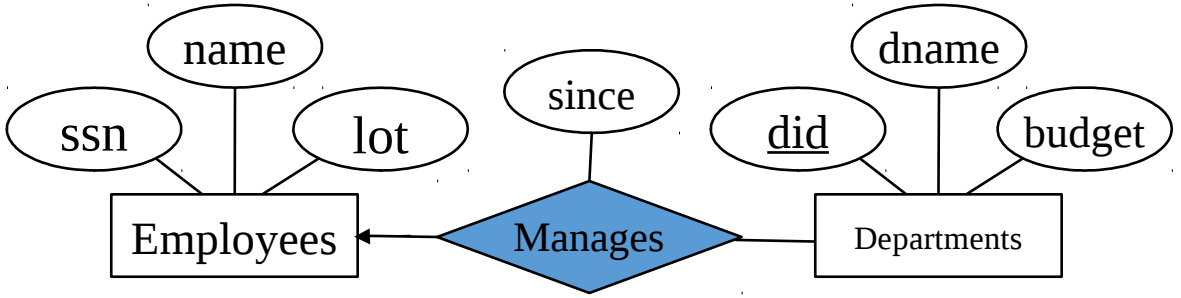
Relationship Sets to Tables

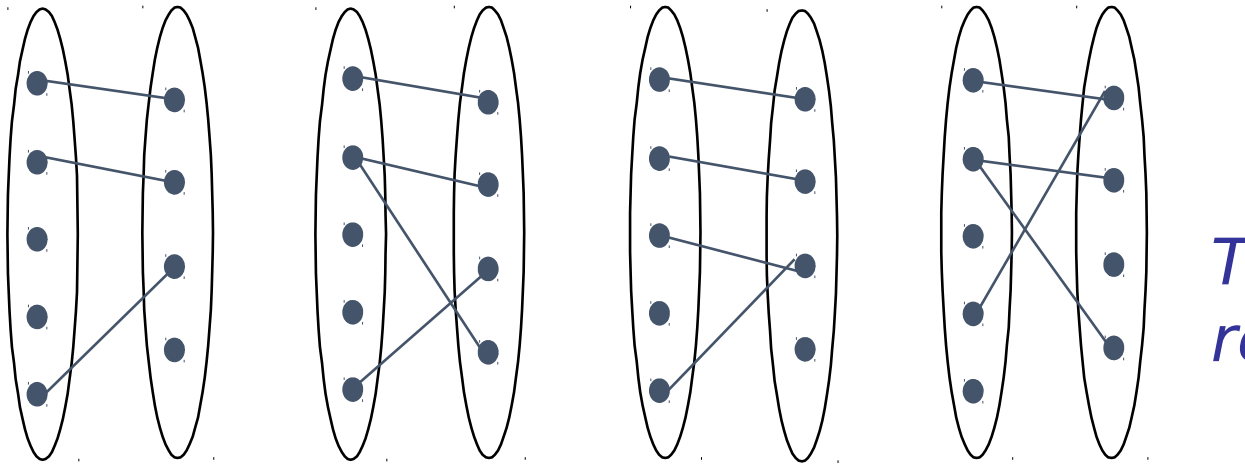
- In translating a relationship set to a relation, attributes of the relation must include:
  - Keys for each participating entity set (declared as foreign keys). This set of keys is at least a *superkey* for the relation.
  - All descriptive attributes.

```
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)
```

Review: Key Constraints

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





1-to-1 | 1-to Many | Many-to-1 | Many-to-Many

Translation to relational model?

Translating ER Diagrams with Key Constraints

- Map relationship to a table:
  - Note that did is the key now! (did and ssn together as the key?)
  - Separate tables for Employees and Departments.

Solution 1:

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

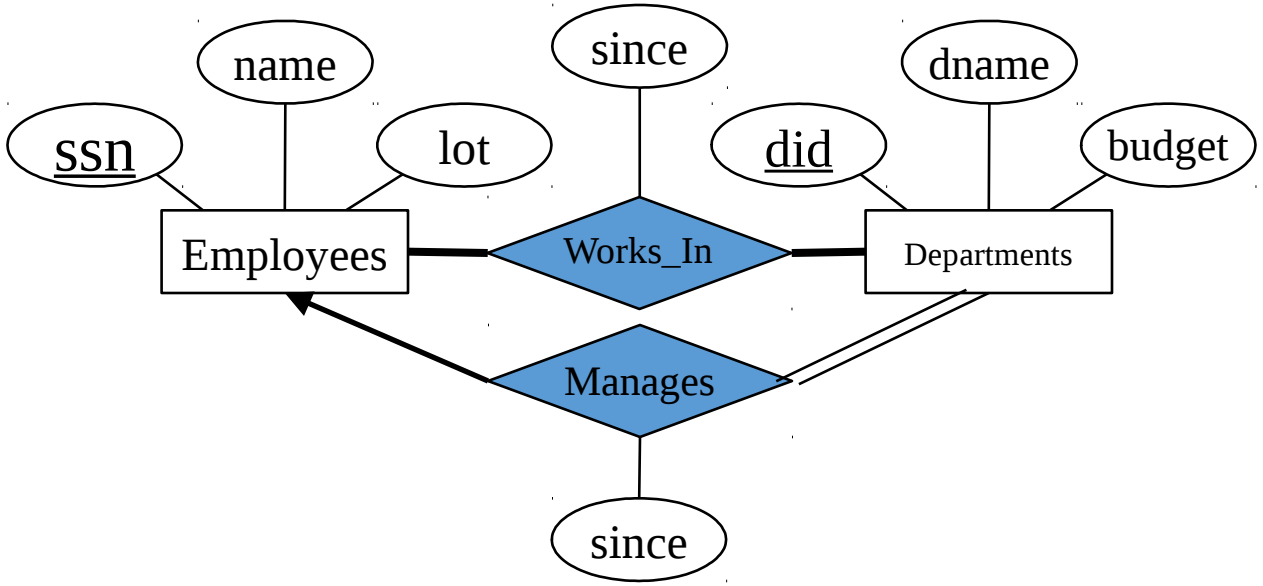
- Since each department has a unique manager, we could instead combine Manages and Departments.
  - Drawback: null value for ssn if a dept. doesn't have a manager.

Solution 2:

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

Review: Participation Constraints

- Does every department have a manager?
  - If so, this is a *participation constraint*: the participation of Departments in Manages is said to be *total* (vs. *partial*).
  - Every *did* value in Departments table must appear in a row of the Manages table (with a non-null *ssn* value)



Participation Constraints in SQL

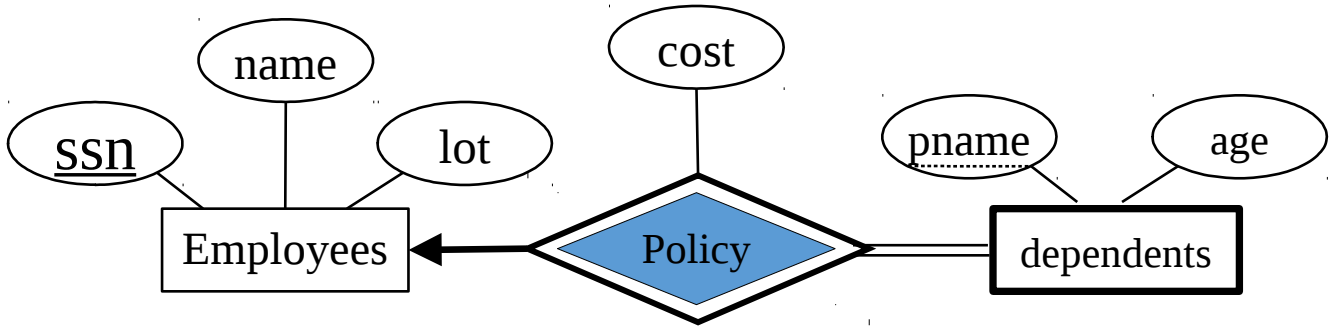
- What is the participation constraint here?

```
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)
```

- What happens if we remove NOT NULL for ssn?

Review: 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 (1 owner, many weak entities).
  - Weak entity set must have total participation in this *identifying* relationship set.

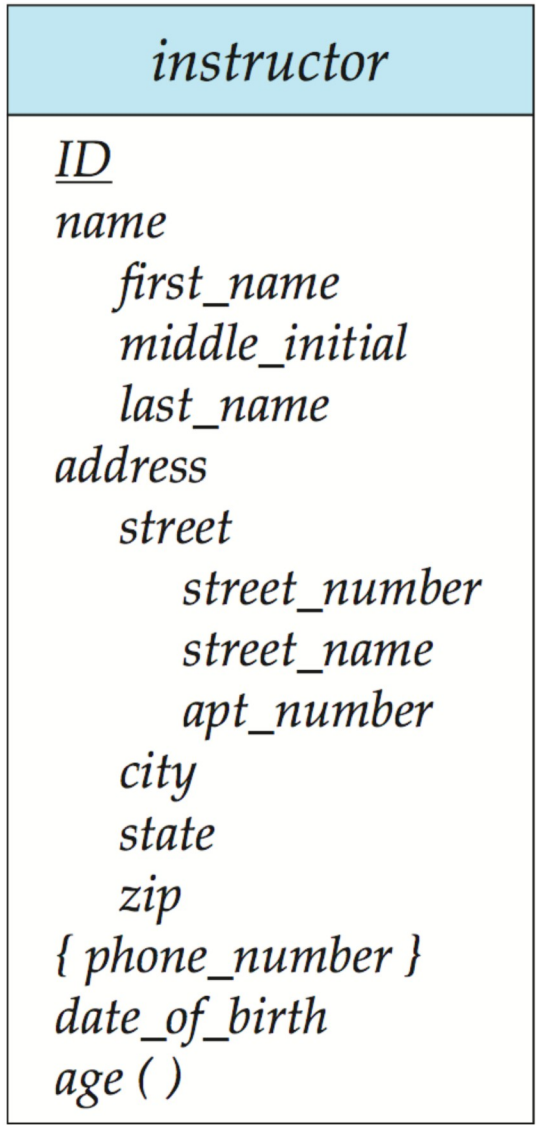


Translating Weak Entity Sets

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

```
CREATE TABLE Dep_Policy (
   pname CHAR(20),
   age INTEGER,
   cost REAL,
   ssn CHAR(11)
   PRIMARY KEY (pname, ssn),
   FOREIGN KEY (ssn) REFERENCES Employees,
      ON DELETE CASCADE)
```

Composite and Multivalued 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*
- 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)*

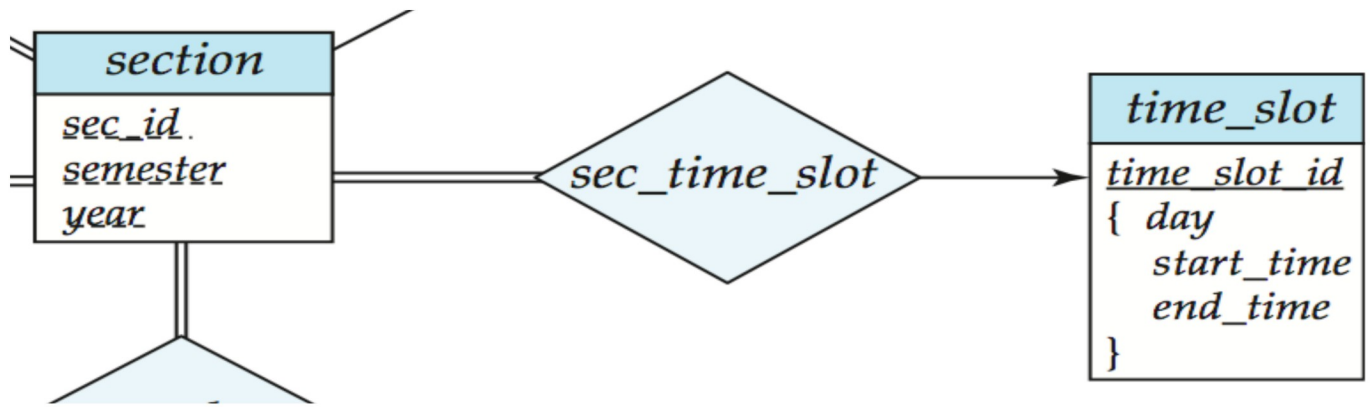
Composite and Multivalued Attributes

```
CREATE TABLE  inst_phone
(
   ID integer,
   Phone_number CHAR(8),
)
```

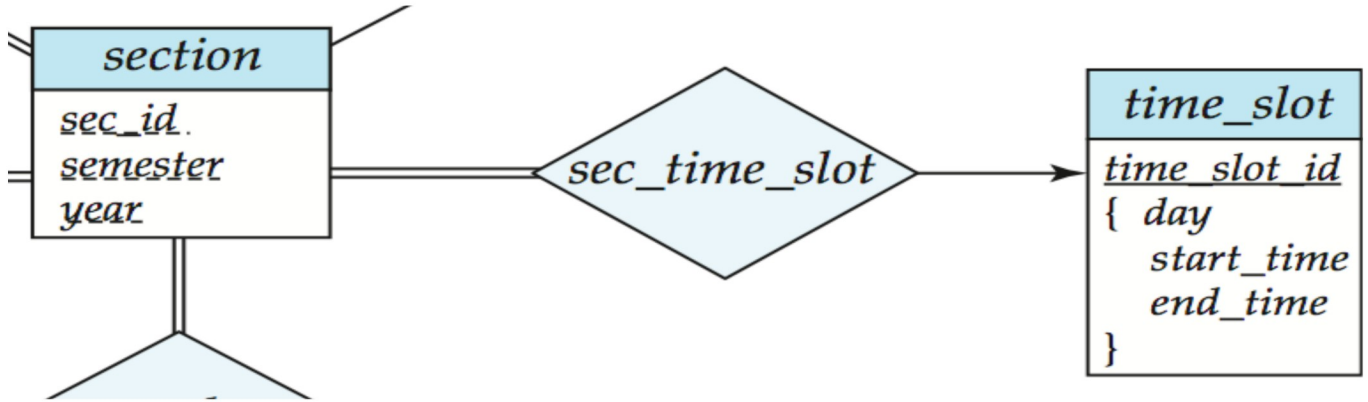
- A multivalued attribute *M* of an entity *E* is represented by a separate schema *EM*
  - Schema *EM* has attributes corresponding to the primary key of *E* and an attribute corresponding to multivalued attribute *M*
  - Example: Multivalued attribute *phone_number* of *instructor* is represented by 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)

Multivalued Attributes (Cont.)

- Special case:entity *time_slot* has only one attribute other than the primary-key attribute, and that attribute is multivalued
  - O m z D ’ g y, j create the one corresponding to the multivalued attribute
  - *time_slot*(*time_slot_id, day, start_time*, *end_time*)
  - Caveat: *time_slot* attribute of *section (*from *sec_time_slot*) cannot be a foreign key due to this optimization



Multivalued Attributes (Cont.)



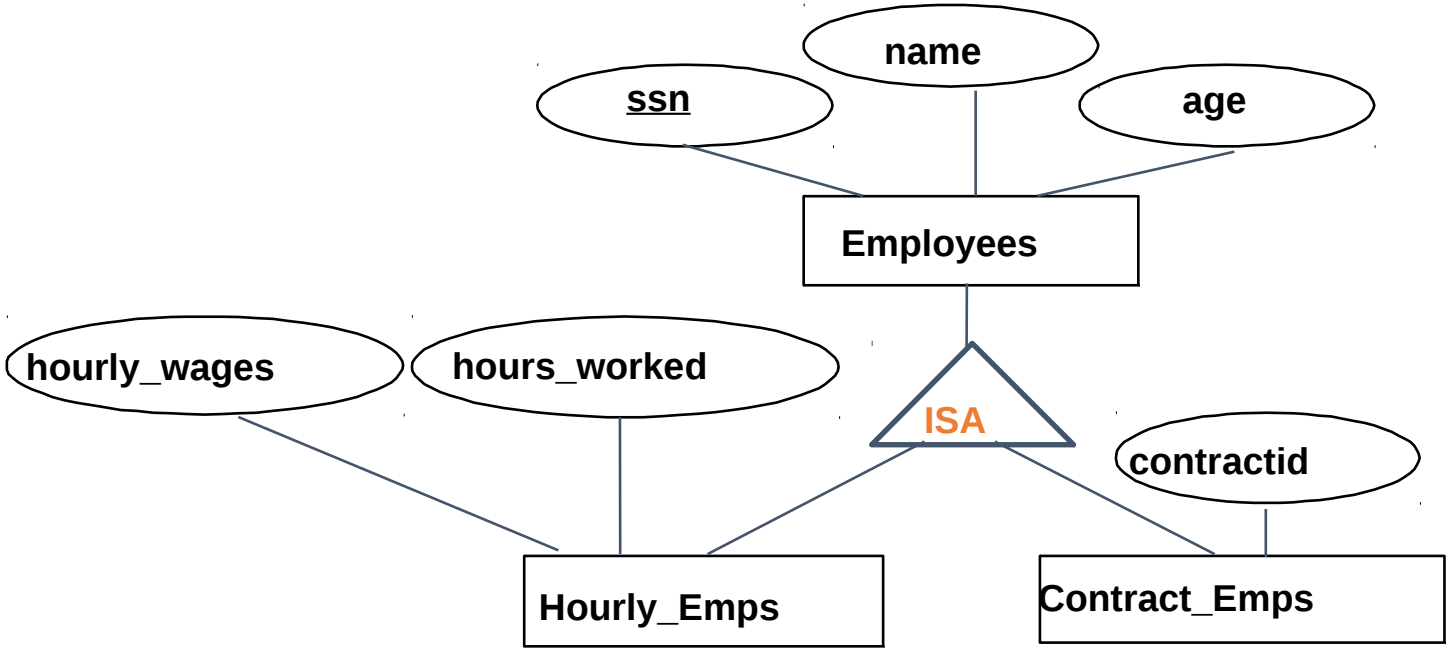
- Originally:
  - sec_time_slot: (22222, 1111)
  - Time_slot: (1111, {(9/5/2017,2:30,3:45), (9/7/2017,2:30,3:45)})
- Optimized:
  - sec_time_slot: (22222, 1111)
  - Time_slot: (1111, 9/5/2017,2:30,3:45), (1111, 9/7/2017,2:30,3:45)

Extended ER Features

Subclasses

- Sometimes, an entity set contains some entities that do share many, but not all properties with the entity set. In this case, we want to define class (entity set) hierarchies.
- A **ISA** B: every A entity is also considered to be a B entity. A *specializes* B, B *generalizes* A.
- A is called *subclass*, B is called *superclass*.
- A subclass *inherits* the attributes of a superclass, and may define additional attributes.

Subclasses



- Hourly_Emps and Contract_Emps inherit the ssn (key!), name and age attributes from Employees.
- They define additional attributes hourly_wages, hours_worked and contractid, resp.

Subclasses

- *Overlap constraints*:
  Can Joe be an Hourly_Emps as well as a Contract_Emps entity?
  (Hourly_Emps OVERLAPS Contract_Emps)

- *Disjoint constraints*:
  Can Joe be an Hourly_Emps as well as a Contract_Emps entity?
  (Hourly_Emps DISJOINTS Contract_Emps)

- *Covering constraints*:
  Does every Employees entity have to be either an Hourly_Emps or a Contract_Emps entity?

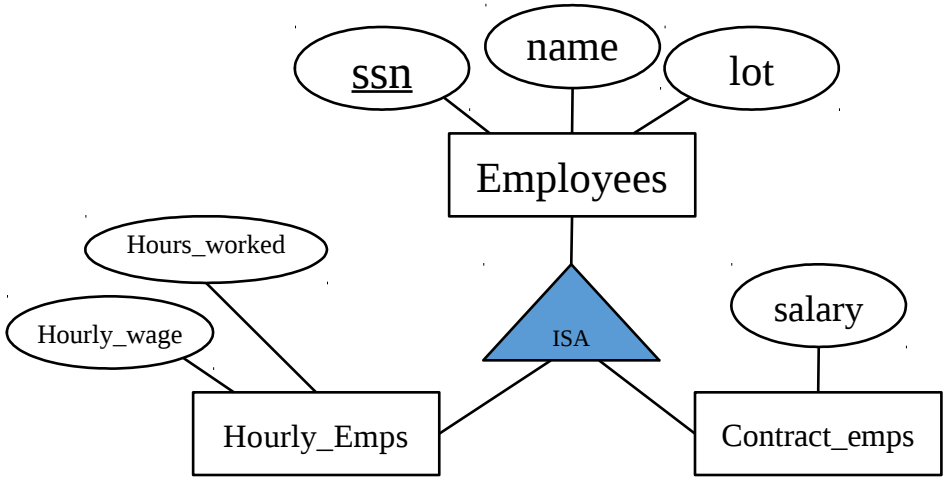
  Hourly_Emps AND Contract_Emps COVER Employees
  (Total Generalization vs Partial Generalization)

Subclasses

- There are several good reasons for using ISA relationships and subclasses:
  - Do not have to redefine all the attributes.
  - Can add descriptive attributes specific to a subclass.
  - To identify entity sets that participate in a relationship set as precisely as possible.
- ISA relationships form a *tree structure* (taxonomy) with one entity set serving as *root*.

Review: ISA Hierarchies

- It is often useful to subdivide entities into classes, like in an OOL
- If we declare A **ISA** B, every A entity is also considered to be a B entity.



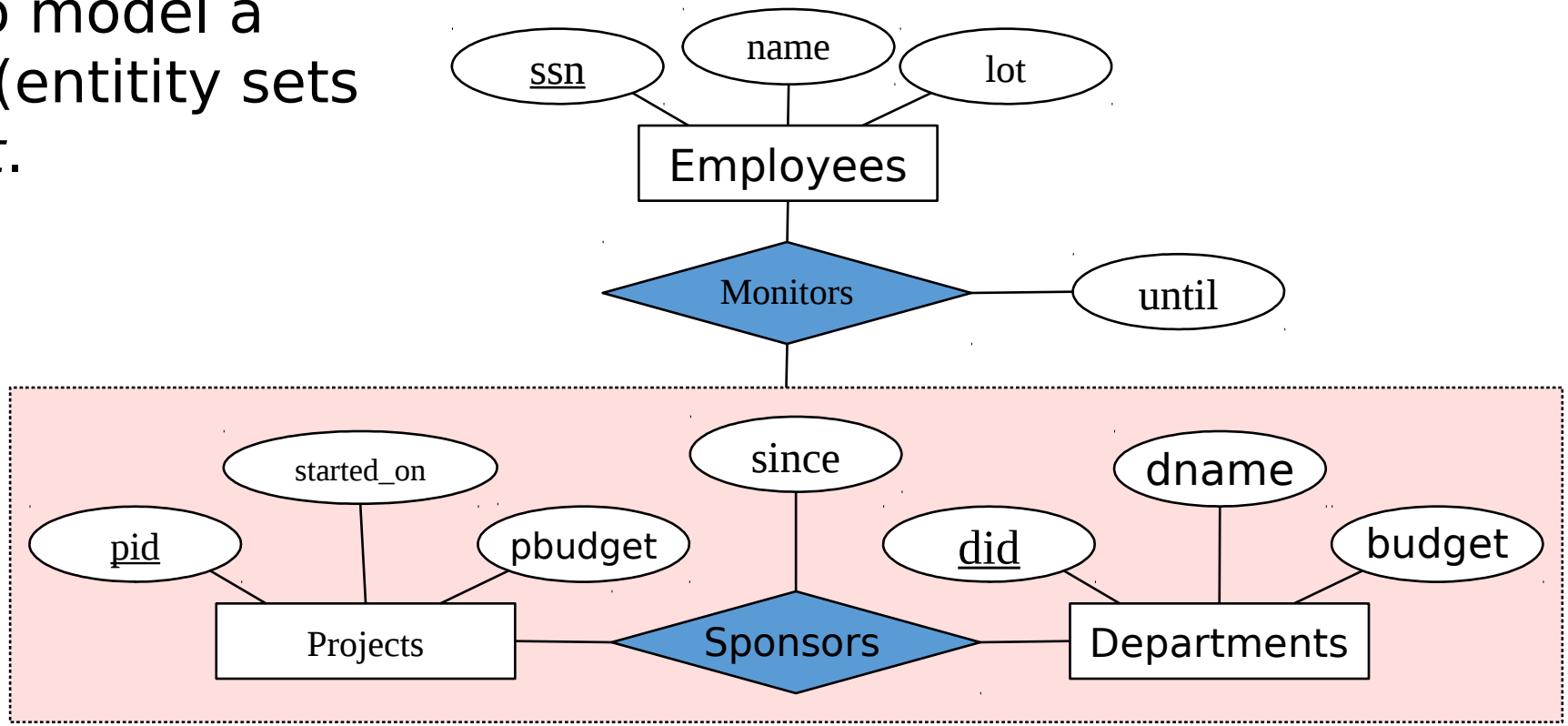
- *Overlap constraints*: Can Joe be an Hourly_Emps as well as a Contract_Emps entity? (*Allowed/disallowed*)
- *Covering constraints*: Does every Employees entity also have to be an Hourly_Emps or a Contract_Emps entity? *(Yes/no)*

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.
  - *Querying Employees requires querying both tables*

Aggregation

- Used when we have to model a relationship involving (entitity sets and) a *relationship set*.
  - *Aggregation* allows a relationship set to be treated as an entity set for purposes of participation in (other) relationships.



- *Aggregation vs. ternary relationship*:
  - Monitors is a distinct relationship, with a descriptive attribute (until).
  - Each sponsorship is monitored by at most one employee.

Schemas Corresponding to Aggregation

- To represent aggregation, create a schema containing
  - primary key of the aggregated relationship,
  - the primary key of the associated entity set
  - any descriptive attributes

Basically treat the aggregation relationship set as a high level entity.

Projects (pid, started_on, pbudget)

Departments (did, dname, budget)

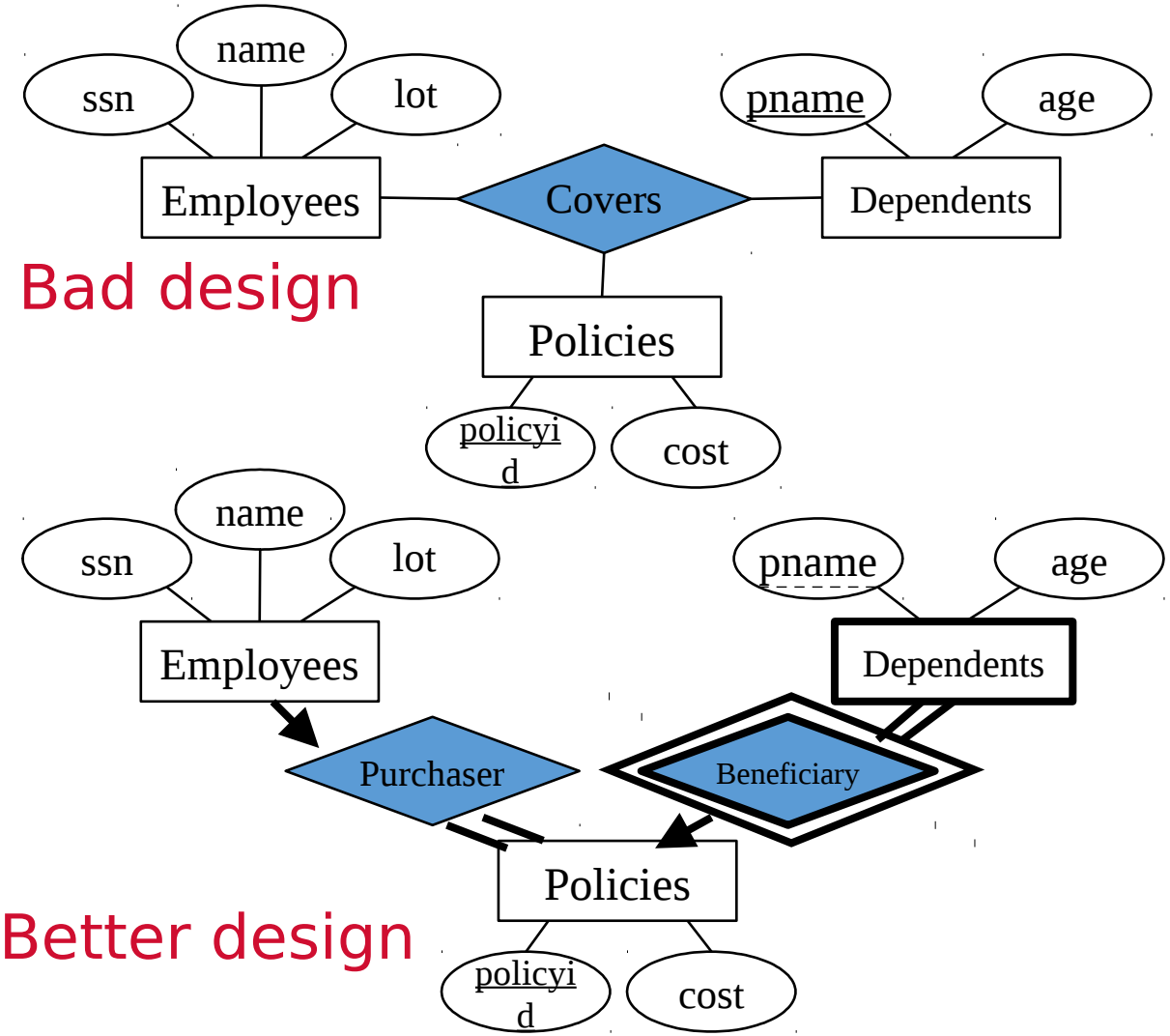
Sponsors (pid, did, since)

Employees (ssn, name, lot)

Monitors (ssn, pid, did, until)

Review: Binary vs. Ternary Relationships

- Recall what were the additional constraints implied by the the better design?
- Better:
  - The same policy cannot be owned jointly by two or more employees.
  - Every policy must be owned by some employee.
  - Dependents is a weak entity set, and each dependent entity is uniquely identified by taking pname in conjunction with the policyid of a policy entity (which, intuitively, covers the given dependent).



Binary vs. Ternary Relationships (Contd.)

- Key constraints allow us to combine Purchaser with Policies, and Beneficiary with Dependents.
- Participation constraints lead to NOT NULL constraints.
- What if Policies is a weak entity set? (generic policy numbers)
  - Make ssn part of the primary key for policies

```
CREATE TABLE  Policies (
   policyid  INTEGER,
   cost  REAL,
   ssn  CHAR(11)  NOT NULL,
   PRIMARY KEY (policyid).
   FOREIGN KEY (ssn) REFERENCES Employees,
      ON DELETE CASCADE)

CREATE TABLE Dependents (
   pname  CHAR(20),
   age  INTEGER,
   policyid  INTEGER,
   PRIMARY KEY (pname, policyid).
   FOREIGN KEY (policyid) REFERENCES Policies,
      ON DELETE CASCADE)
```

Views

- A *view* is just a relation, but we store a *definition*, rather than a set of tuples.

```
CREATE VIEW YoungActiveStudents (login, grade)
    AS SELECT S.login, E.grade
    FROM Students S, Enrolled E
    WHERE S.sid = E.sid and S.age<21
```

  - Views can be dropped using the DROP VIEW command.
    - How to handle DROP TABLE if there's a view on the table?
    - DROP TABLE command has options to let the user specify this.

Views to support ISA relations

- The common elements of an ISA hierarchy can be supported using views.
- For example, consider this implementation for the employee, hourly employee, and contract employee example.

```
CREATE VIEW Employee(ssn, name, lot)
    AS SELECT H.ssn, H.name, H.lot
    FROM Hourly_Emps H
    UNION
    SELECT C.ssn, C.name, C.lot
    FROM Contract_Emps C
```

Views and Security

- Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).

login	grade
smith@cs	C
smith@cs	B
smith@math	A
jones@cs	B

Design Principles

- Faithfulness
  - Design must be faithful to the specification / reality.
  - Relevant aspects of reality must be represented in the model.
- Avoiding redundancy
  - Redundant representation blows up ER diagram and makes it harder to understand.
  - Redundant representation wastes storage.
  - Redundancy may lead to inconsistencies in the database.

Design Principles

- Keep it simple
  - The simpler, the easier to understand for some (external) reader of the ER diagrams.
  - Avoid introducing more elements than necessary.
  - If possible, prefer attributes over entity sets and relationship sets.
- Formulate constraints as far as possible
  - A lot of data semantics can (and should) be captured.
  - But some constraints cannot be captured in ER diagrams.

High-Level Design With E-R Model

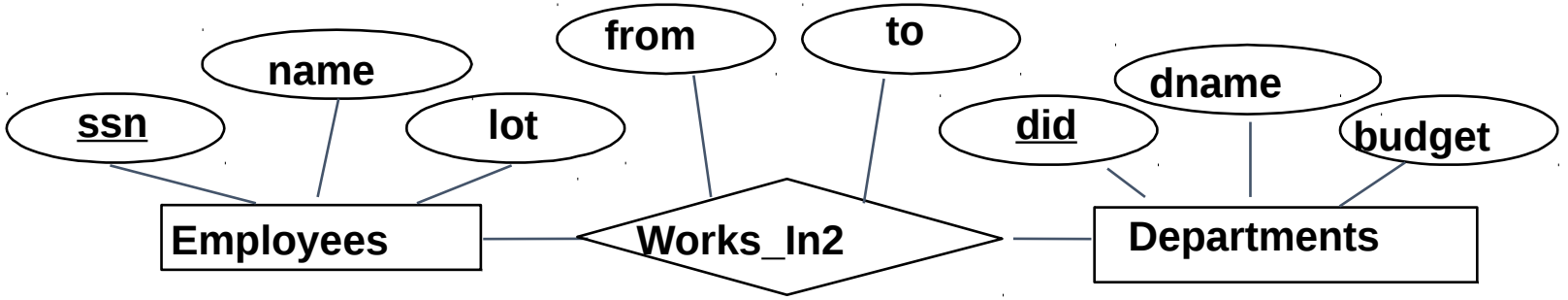
- Major E-R design decisions
  - Should a concept be modeled as an entity or an attribute?
  - Should a concept be modeled as an entity or a relationship?
  - What relationships to use: binary or ternary?
  - The use of a strong or weak entity set.
  - The use of specialization/generalization – contributes 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.

Entity Sets vs. Attributes

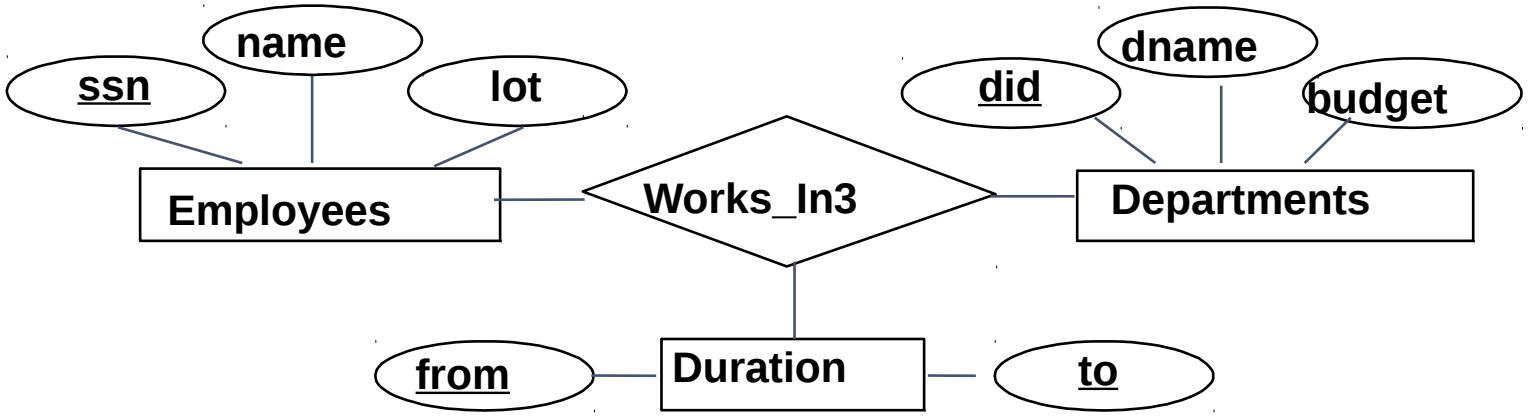
- Should *address* be an attribute of Employees or an entity (connected to Employees by a relationship)?
- Depends upon the use we want to make of address information, and the semantics of the data:
  - If we have several addresses per employee, *address* must be an entity (since attributes cannot be 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).

Entity Sets vs. Attributes

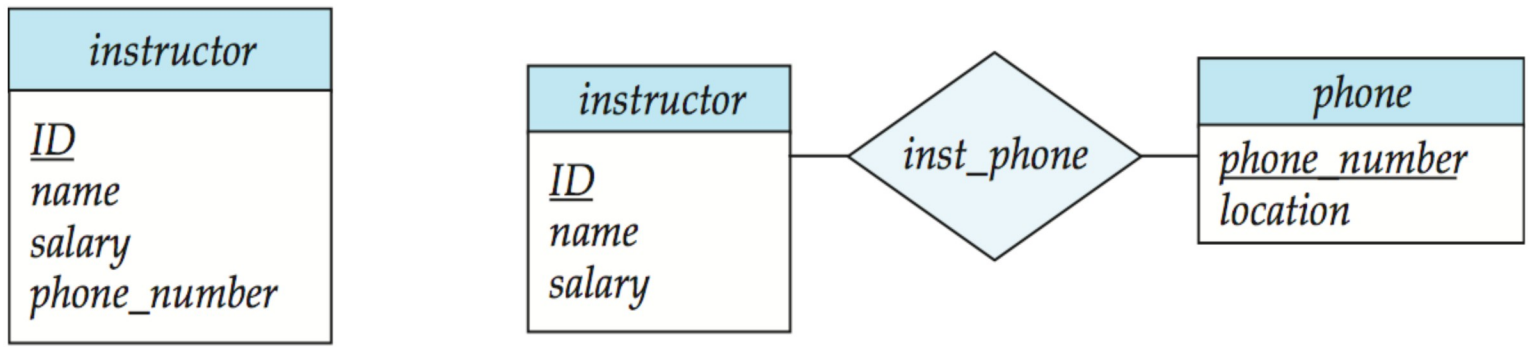
- Works_In2 does not allow an employee to work in the same department for two or more periods (why?).



- We want to record *several values of the descriptive attributes for each instance of this relationship.*



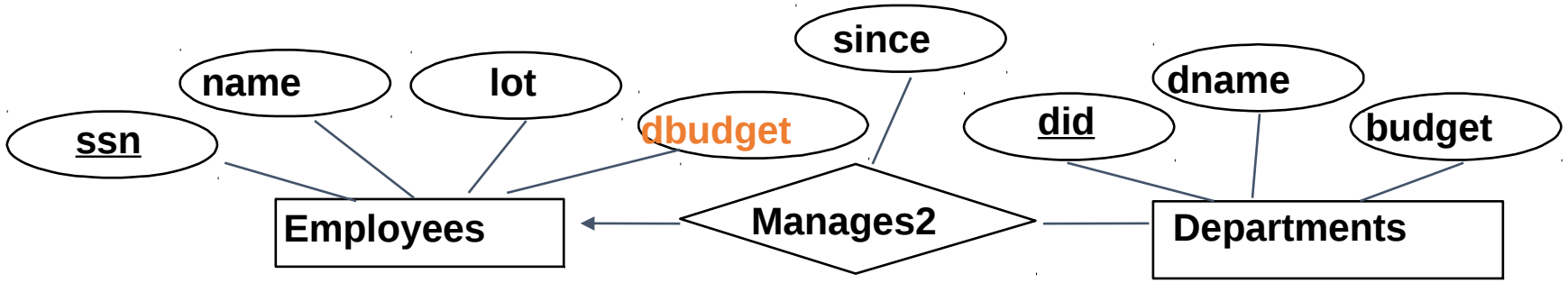
Entity Sets vs. Attributes



- Use of phone as an entity allows extra information about phone numbers (plus multiple phone numbers)

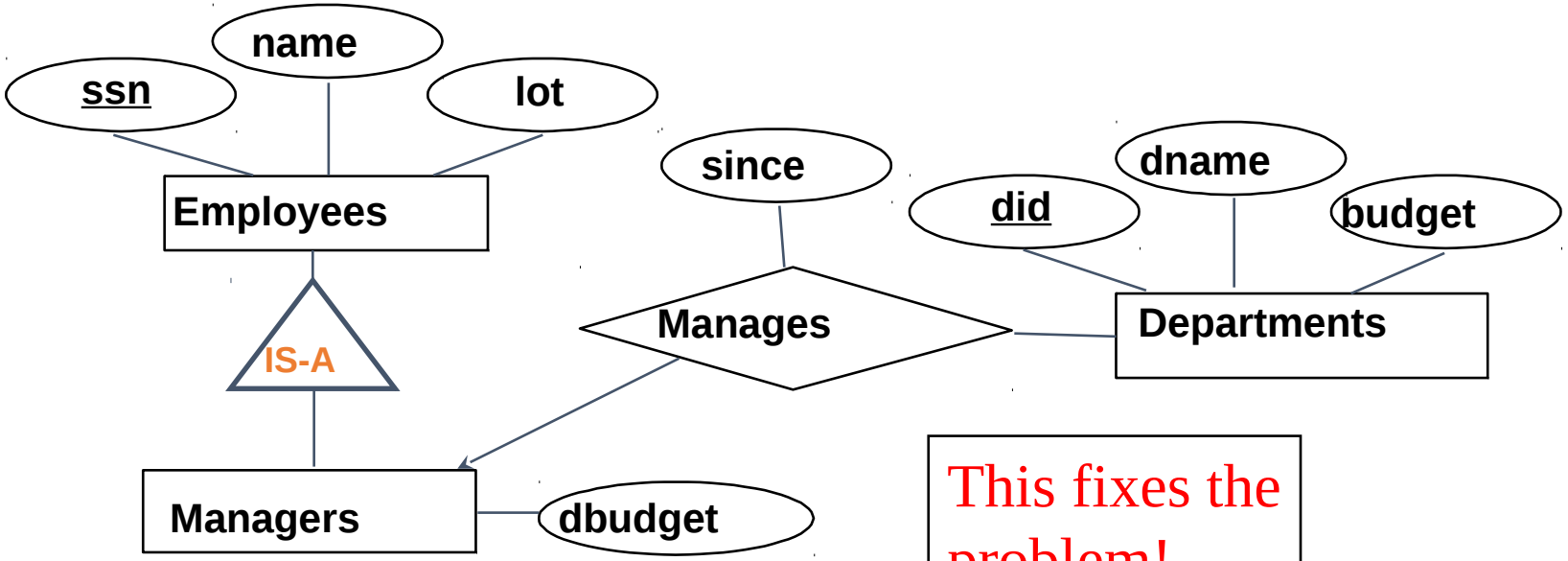
Entity vs. Relationship

- What about this diagram?



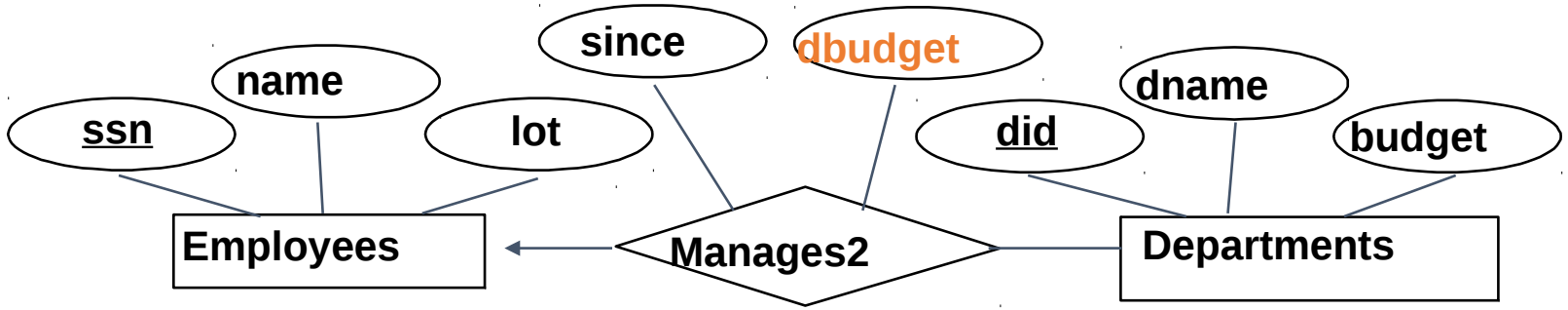
Not every employee is a manager!!

- The following ER diagram is more appropriate and avoids the above problems!



This fixes the problem!

Entity vs. Relationship



- This ER diagram o.k. if a manager gets a *separate* discretionary budget for each dept.
- But what if a manager gets a discretionary budget that covers *all* managed depts?
  - Redundancy of *dbudget,* which is stored for each dept managed by the manager.
  - Misleading: suggests *dbudget* tied to managed dept.

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 people think about their applications.
- Basic constructs: *entities*, *relationships*, and *attributes* (of entities and relationships).
- Some additional constructs: *weak entities*, *ISA hierarchies*, and *aggregation*.
- Note: There are many variations on ER model.

Summary of ER (Contd.)

- Several kinds of integrity constraints can be expressed in the ER model: *key constraints*, *participation constraints*, and *overlap/covering constraints* for ISA hierarchies. Some *foreign key constraints* are also implicit in the definition of a relationship set.
  - Some constraints (notably, *functional dependencies*) cannot be expressed in the ER model.
  - Constraints play an important role in determining the best database design for an enterprise.

Summary of ER (Contd.)

- ER design is *subjective*. There are often many ways to model a given scenario! Analyzing alternatives can be tricky, especially for a large enterprise. Common choices include:
  - Entity vs. attribute, entity vs. relationship, binary or n-ary relationship, whether or not to use ISA hierarchies, and whether or not to use aggregation.
- Ensuring good database design: resulting relational schema should be analyzed and refined further. FD information and normalization techniques are especially useful.

Next:

- Introduction to SQL