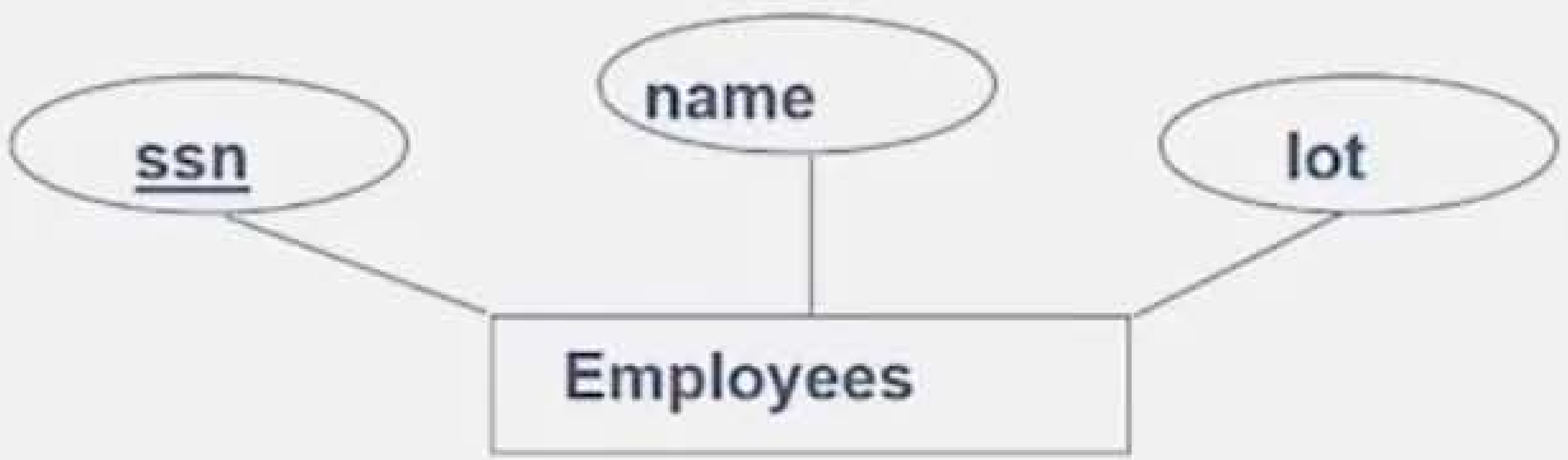
ER to Relational

This is what we're going to talk about now.

Logical DB Design: ER to Relational

- Entity sets to tables:




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

So, again, remember the ER diagram.

Logical DB Design: ER to Relational

- Entity sets to tables:

ER

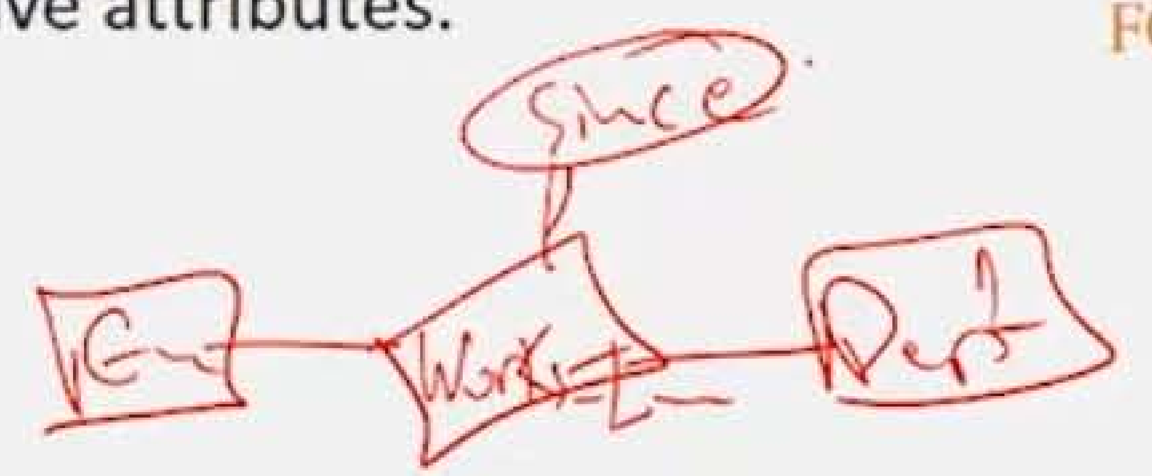
```
CREATE TABLE Employees
  (ssn CHAR(11),
  name CHAR(20),
  lot  INTEGER,
  PRIMARY KEY  (ssn))
```

Relational (SQL)

underlined here, that means that ssn is the key.

Relationship Sets to Tables

- In translating a relationship set to a relation, 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.

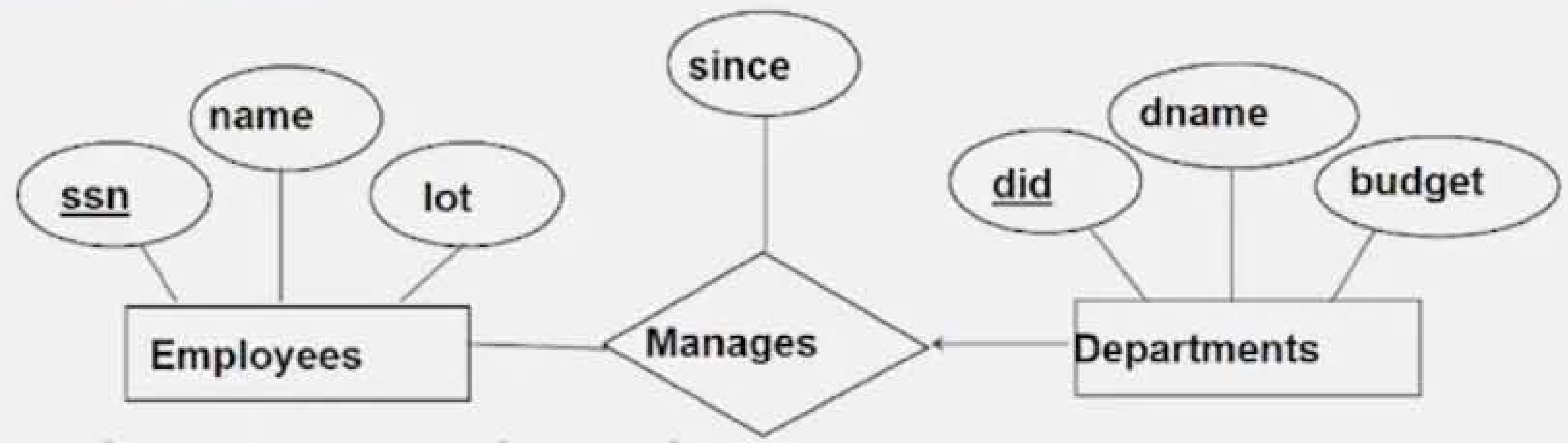


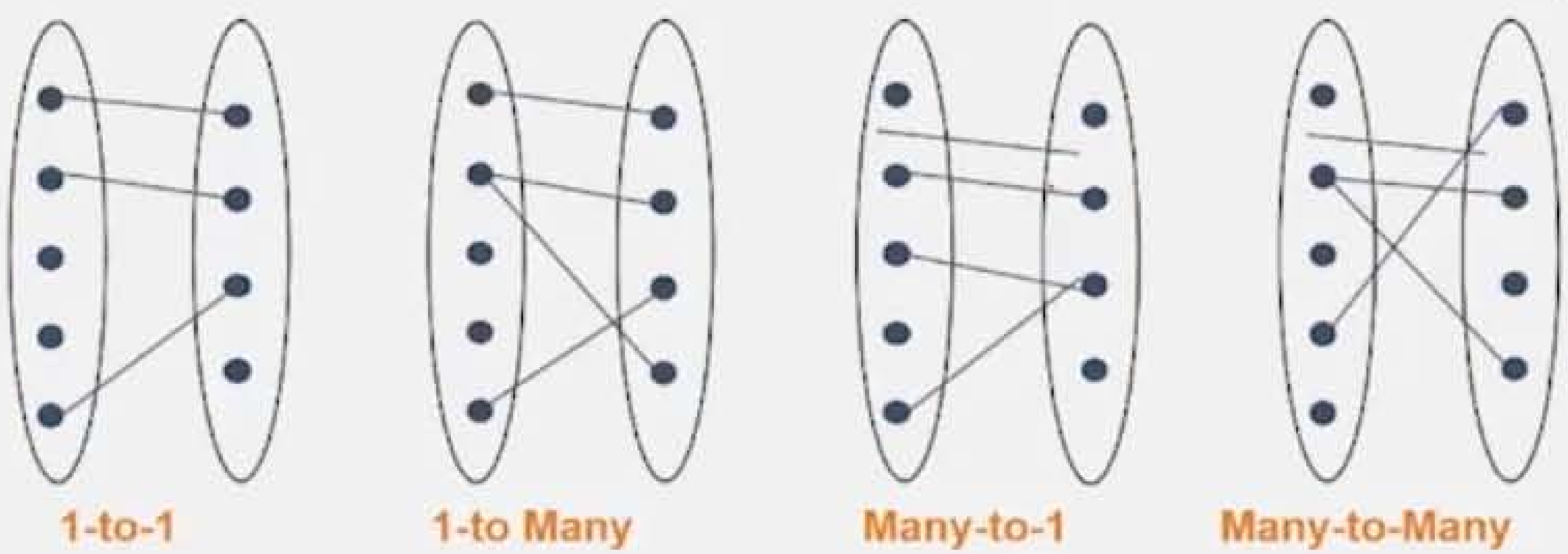

```
CREATE TABLE Works_In(
 ssn CHAR(11),
 did INTEGER,
 since DATE,
 PRIMARY KEY (ssn, did),
 FOREIGN KEY (ssn)
     REFERENCES Employees,
 FOREIGN KEY (did)
     REFERENCES Departments)
```

So, ssn reference employees and did references the departments.

Review: Key Constraints

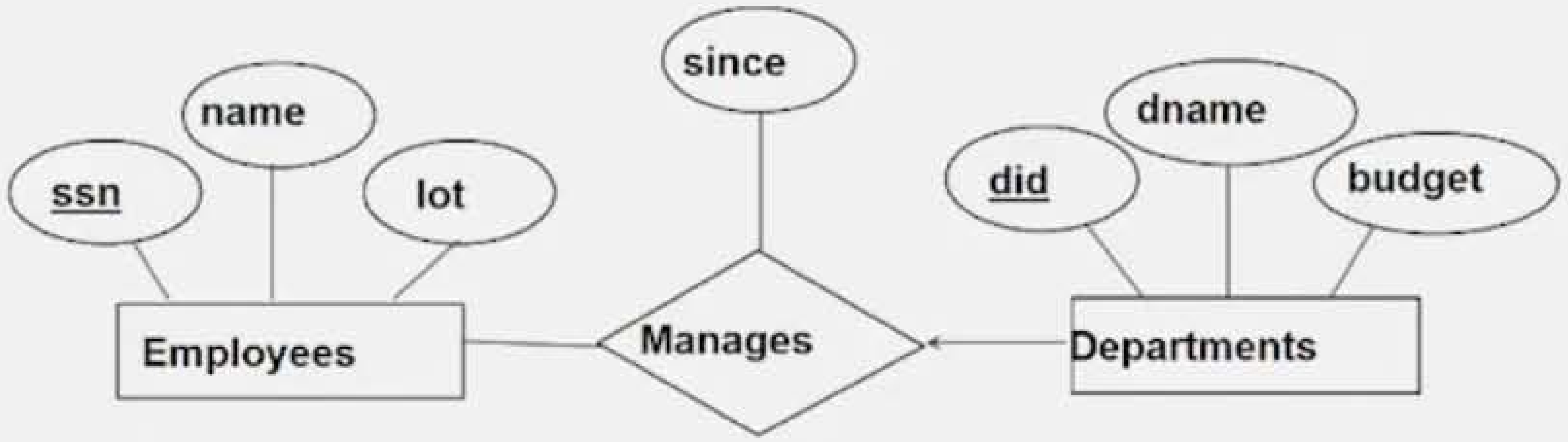
- Each dept has at most one manager, according to the *key constraint* on Manages.





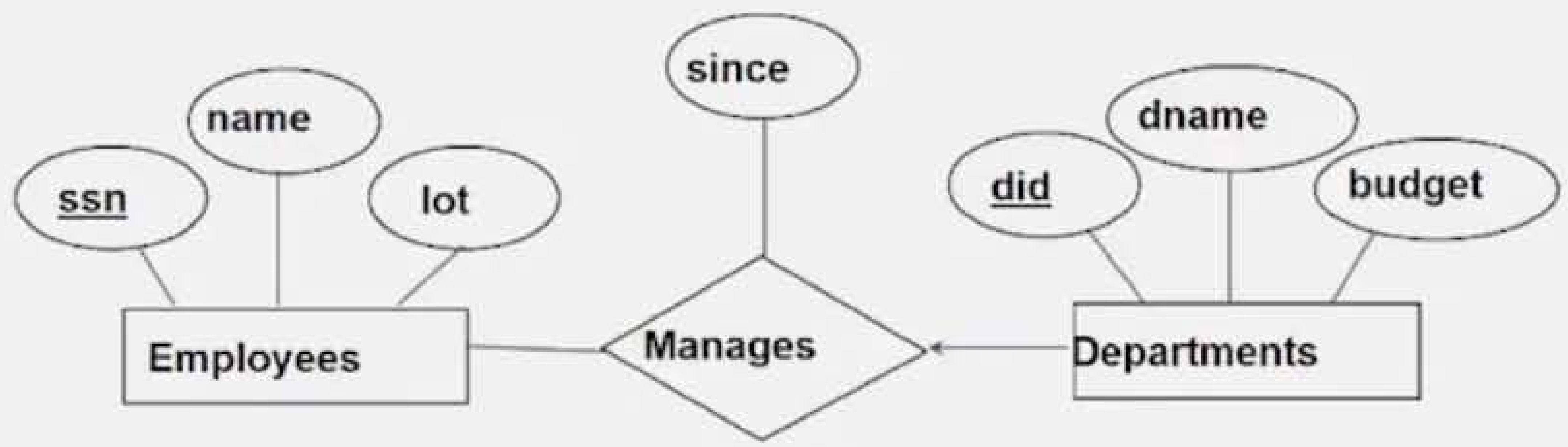
Translation to relational model?

like when you converted it to SQL or to relational.



a one-to-many relationship between the departments and employees,

Share



Translation to relational model (SQL)?

So, one way to do that is to create a table for managers,

3:28 / 8:48

Translating ER Diagrams with Key Constraints

- Map relationship to a table:
  - Note that did is the key now!
  - Separate tables for Employees and Departments.
- 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),      <-- Manager
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn) REFERENCES Employees)
```

So, one-to-many relationship is satisfied very well here.

In the following SQL, what is/are the key constraints?

```sql
CREATE TABLE Employees (
    employee_id INTEGER,
    badge_id INTEGER,
        PRIMARY KEY (employee_id),
        FOREIGN KEY (badge_id) REFERENCES Badge
    );
```

- ○ FOREIGN KEY
- ◉ PRIMARY KEY and FOREIGN KEY

Correct

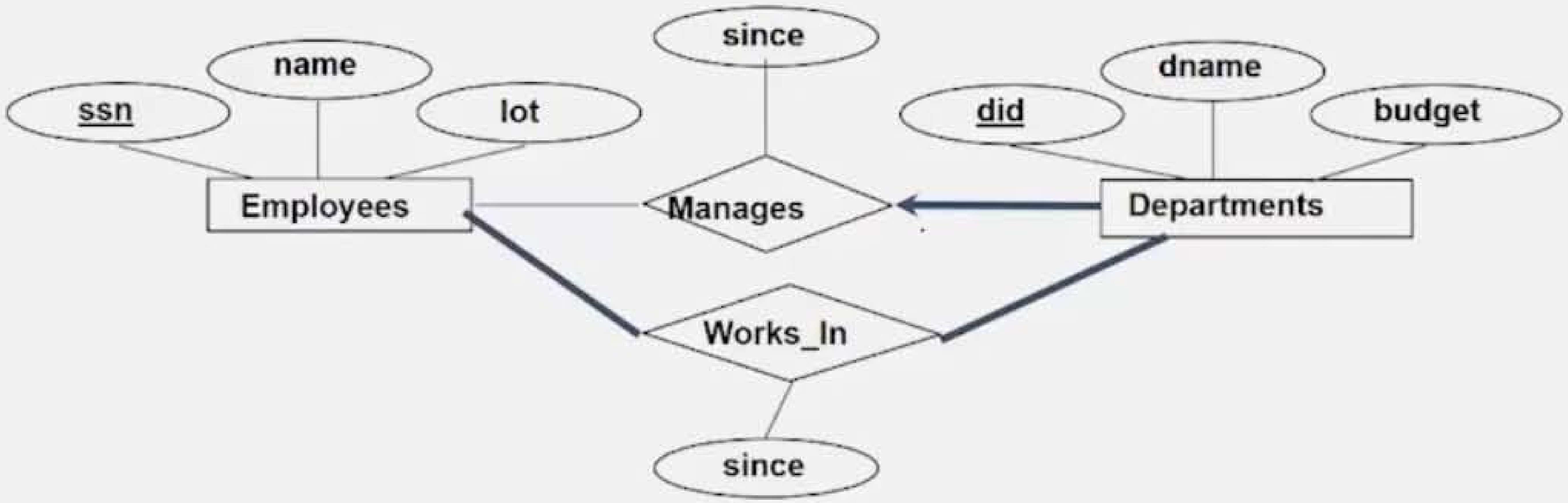
Both the primary key and foreign key are key constraint values.

- ○ PRIMARY KEY

Continue

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!)



the department has a total participation if it must appear in the manager's relationship.

Participation Constraints in SQL

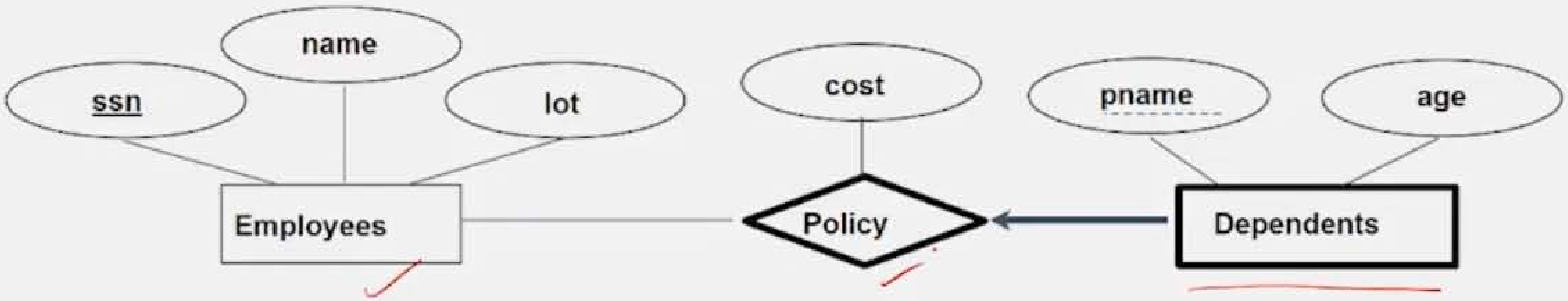
- We can capture participation constraints involving one entity set in a binary relationship, but little else (without resorting to CHECK constraints).

```
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)
```

Not null here means that the department must have a manager.

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.



has a total participation in the identifying relationship.

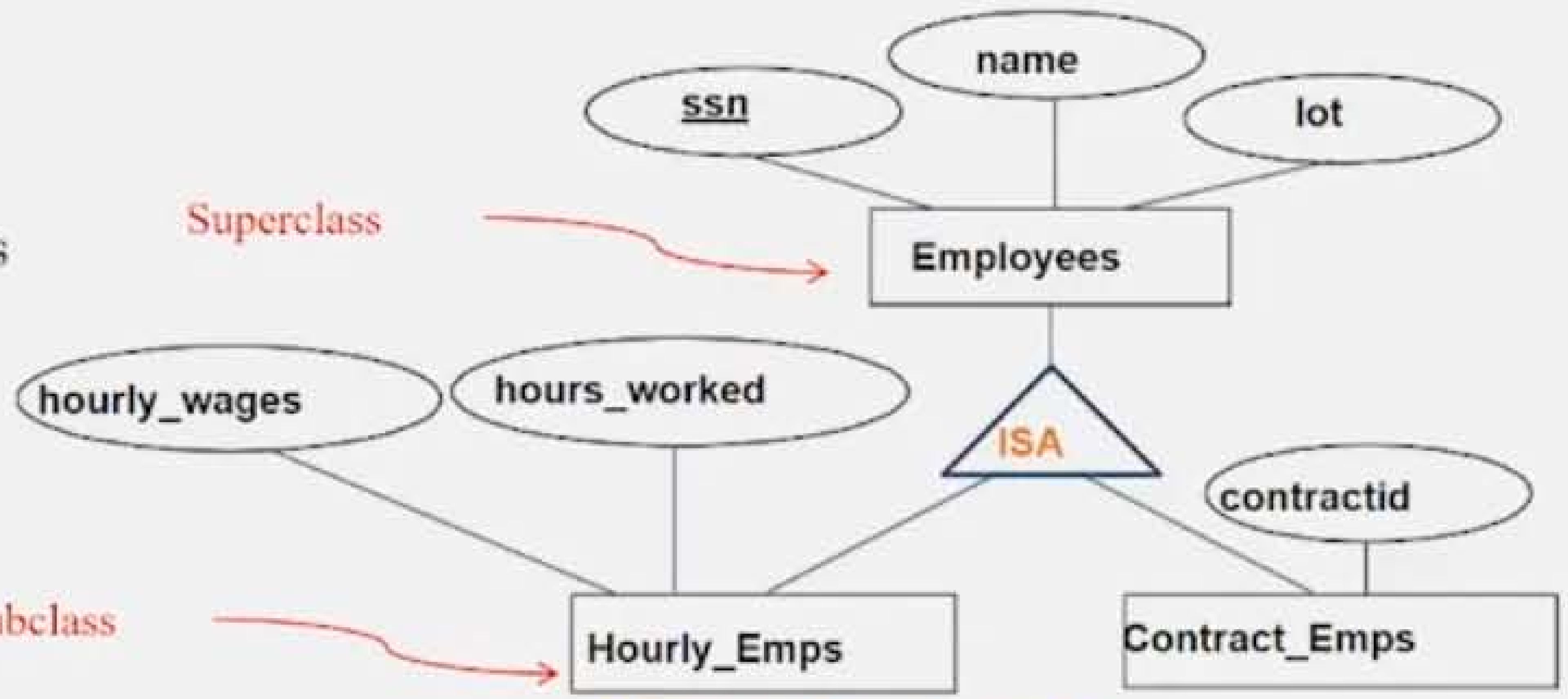
Translating Weak Entity Sets

- 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)
```

the age and the cost.

Class 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)*
- Reasons for using ISA:
  - To add descriptive attributes specific to a subclass.
  - To identify entities that participate in a relationship.

Again, this is just a reminder why we need class hierarchies as we mentioned before.

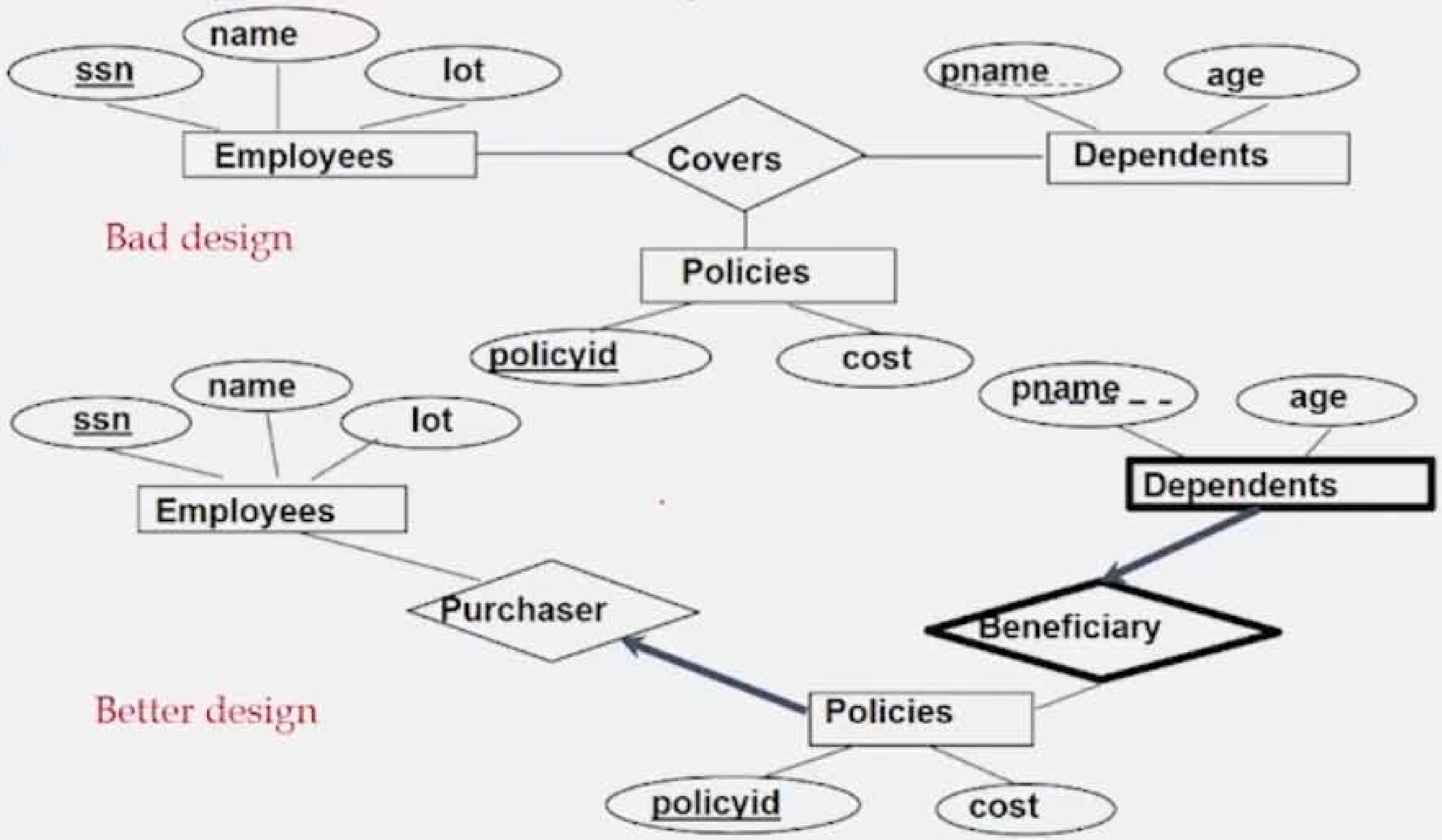
Translating ISA Hierarchies to Relations

- ***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.

you can just create two tables for these instead.

Binary vs. Ternary Relationships

- If each policy is owned by just 1 employee, and each dependent is tied to the covering policy, first diagram is inaccurate.
- What are the additional constraints in the 2nd diagram?



So, to define that,

Binary vs. Ternary Relationships (Contd.)

- The key constraints allow us to combine Purchaser with Policies and Beneficiary with Dependents.
- Participation constraints lead to NOT NULL constraints.

```
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)
```

So, you have this deleted cascade,