

The Entity-Relationship Model

Chapter 2

CENG351

Overview of Database Design

- **Conceptual design:** (*ER Model is used at this stage.*)
 - What are the **entities** and **relationships** in the enterprise?
 - What information about these entities and relationships should we store in the database?
 - What are the *integrity constraints* or *business rules* that hold?
 - A database “schema” in the ER Model can be represented pictorially (*ER diagrams*).
 - We can map an ER diagram into a relational schema.
- **Schema Refinement:** (Normalization)
 - Check relational schema for redundancies and anomalies.
- **Physical Database Design and Tuning:**
 - Consider typical workloads and further refine the db design.

DB Requirements example

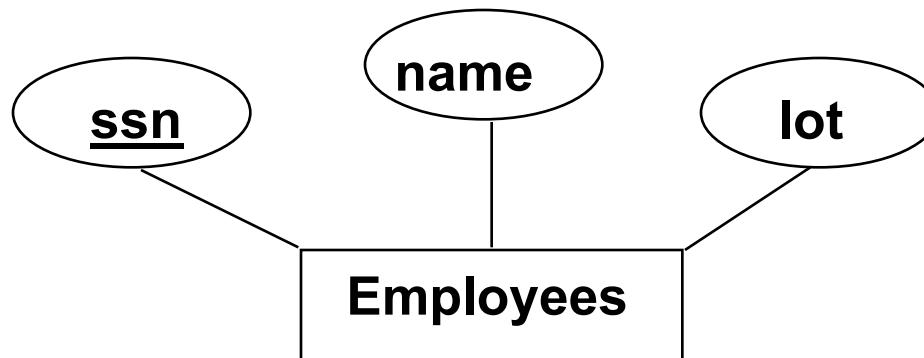
A company database needs to store information about

- employees (identified by ssn, with salary and phone as attributes);
- departments (identified by dno, with dname and budget as attributes); and
- children of employees (with name and age as attributes).
- Employees work in departments;
- each department is managed by an employee;
- a child must be identified uniquely by name when the parent (who is an employee; assume that only one parent works for the company) is known. We are not interested in information about a child once the parent leaves the company.

ER Model Basics – Entity Set

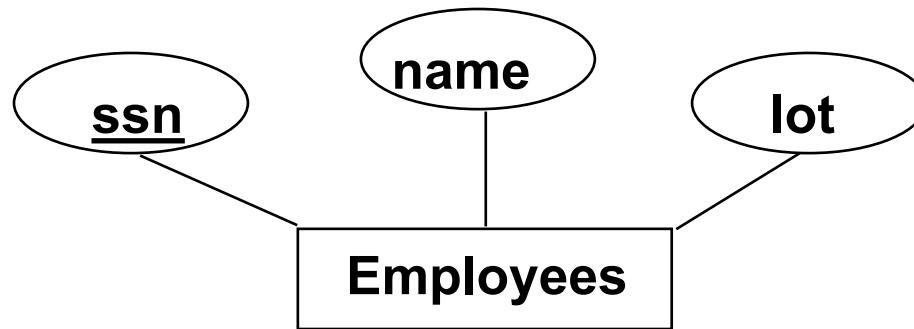
- **Entity:** Real-world object distinguishable from other objects. An entity is described (in DB) using a set of attributes.
- **Entity Set:** A collection of similar entities. E.g., all employees.
 - All entities in an entity set have the same set of attributes.
 - Each entity set has a **key**.
 -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

Employees Entity Set:



Logical DB Design: ER to Relational Model

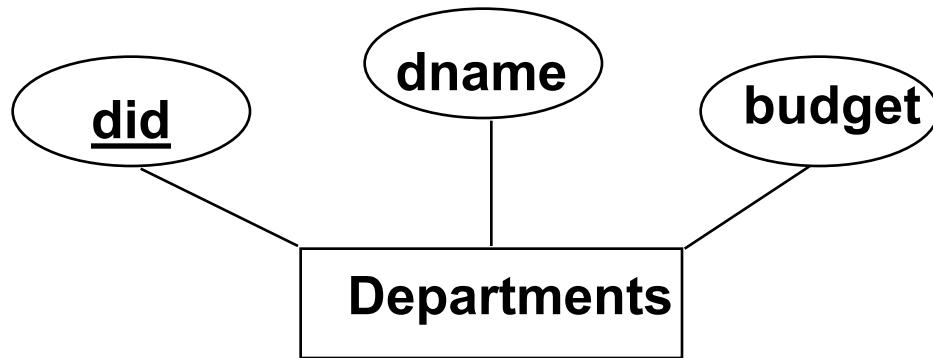
- Entity sets to tables:



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

Logical DB Design: ER to Relational Model

- Entity sets to tables:

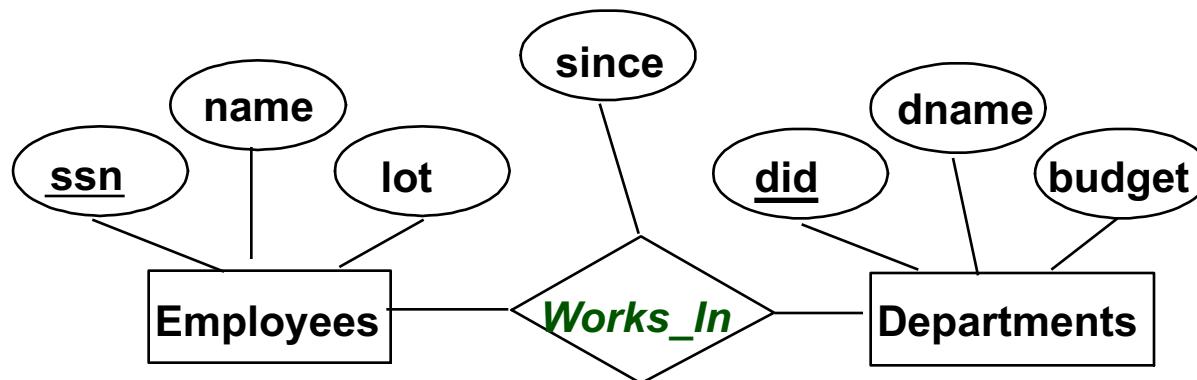


```
CREATE TABLE Departments(  
    did INTEGER,  
    dname CHAR(20),  
    budget REAL,  
    PRIMARY KEY (did))
```

ER Model Basics- Relationship set

- **Relationship:** Association among two or more entities.
 - e.g., Fred works in Pharmacy department.
- **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 $e_1 \in E_1, \dots, e_n \in E_n$

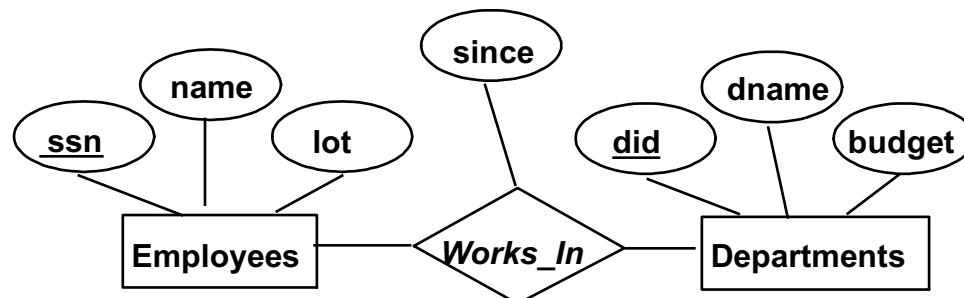
Works_In relationship set:



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.

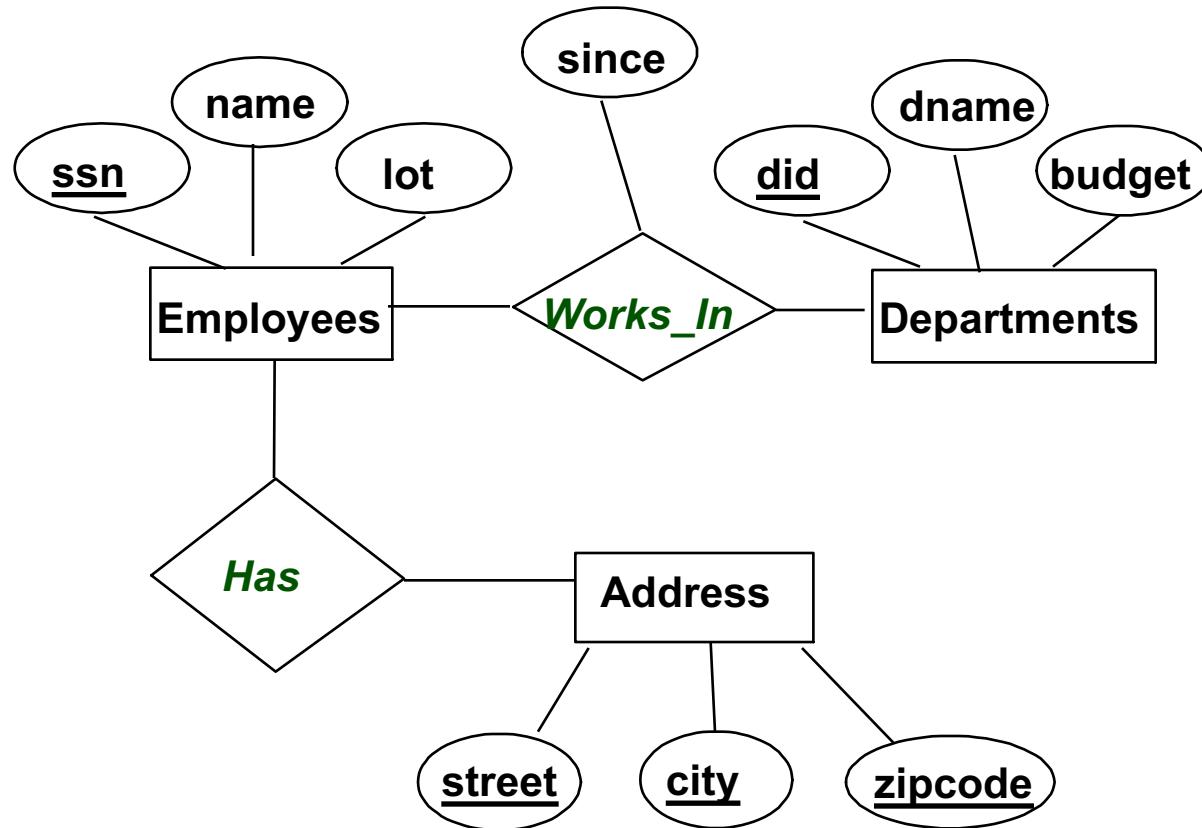
```
CREATE TABLE Works_In(  
    ssn CHAR(11),  
    did INTEGER,  
    since DATE,  
    PRIMARY KEY (ssn, did),  
    FOREIGN KEY (ssn)  
        REFERENCES Employees,  
    FOREIGN KEY (did)  
        REFERENCES Departments)
```



Relationship set (contd.)

- Same entity set could participate in different relationship sets.

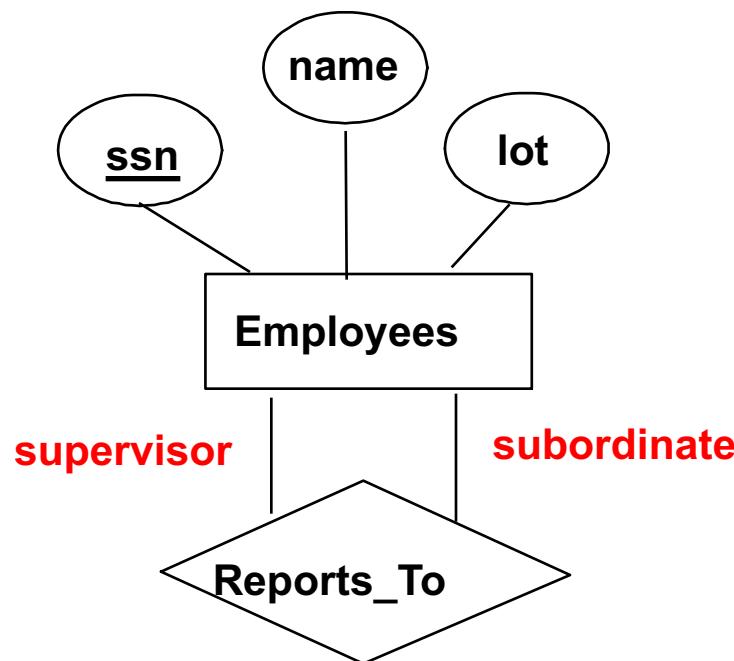
Works_In and *Has* Relationship sets:



Relationship set (contd.)

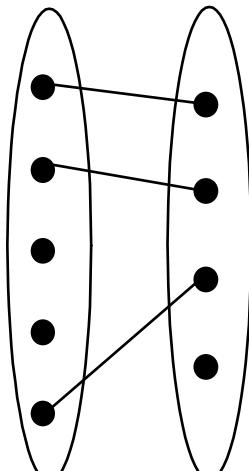
- Same entity set could participate in different “**roles**” in same relationship set.

Reports_To relationship set:

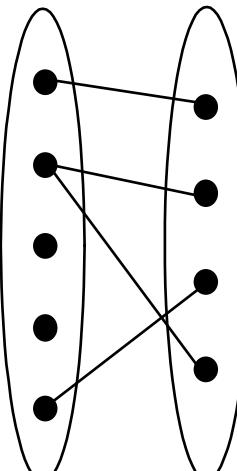


Relationship Types

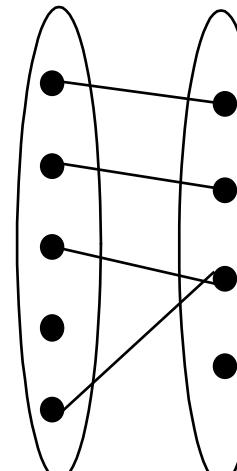
- Consider **Works_In**: An employee can work in **many** departments; a department can have **many** employees:
many-to-many relationship.
- In contrast, each department has **at most one** employee as manager, but a manager can manage **many** departments:
one-to-many relationship.
 - Similarly we can think of examples of **many-to-one** or **one-to-one** relationships.



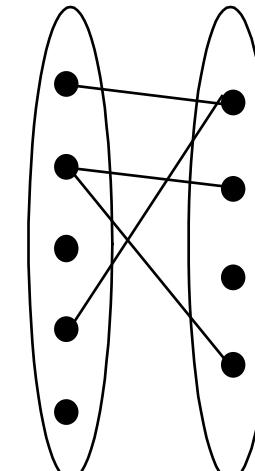
1-to-1



1-to Many



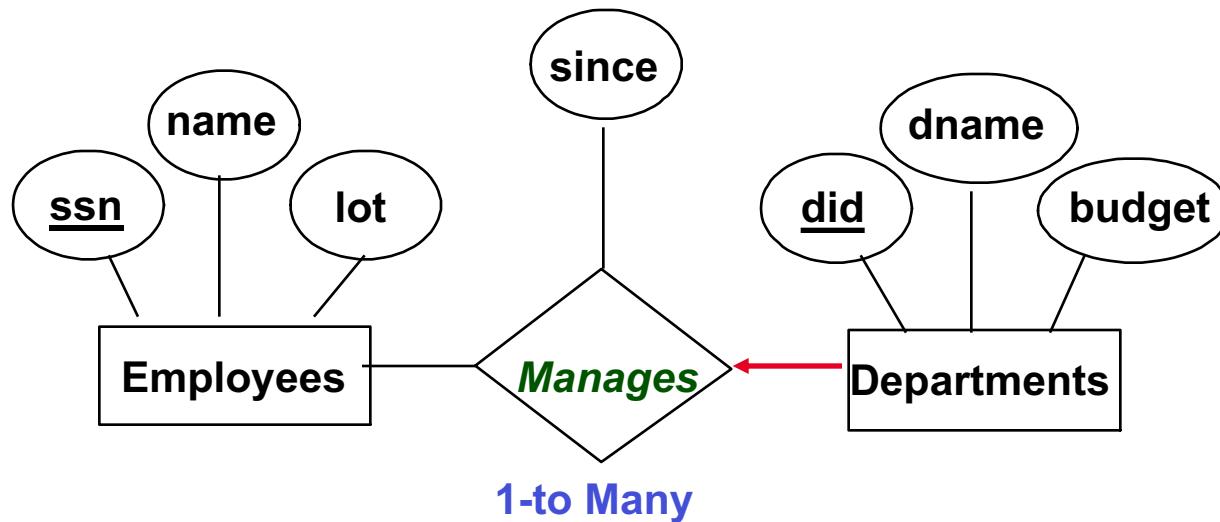
Many-to-1



Many-to-Many

Key Constraints

- The restriction that each department has at most one manager is an example of a key constraint.
- It implies that each department entity appears in *at most one Manages* relationship.



Translating Relationships with Key Constraints

- How do we translate *Manages* into a table?

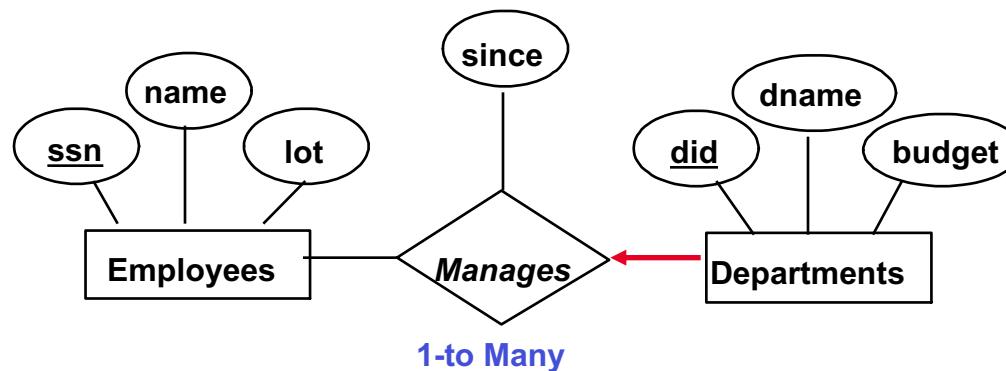
There are two alternative designs:

EITHER:

We could map *Manages* relationship to a separate table (like we did in the previous slide).

OR

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



Alternative 1: Separate Table for *Manages*

- Map *Manages* relationship to a table:
 - Note that **did** is the key now!
 - There are separate tables for *Employees* 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)
```

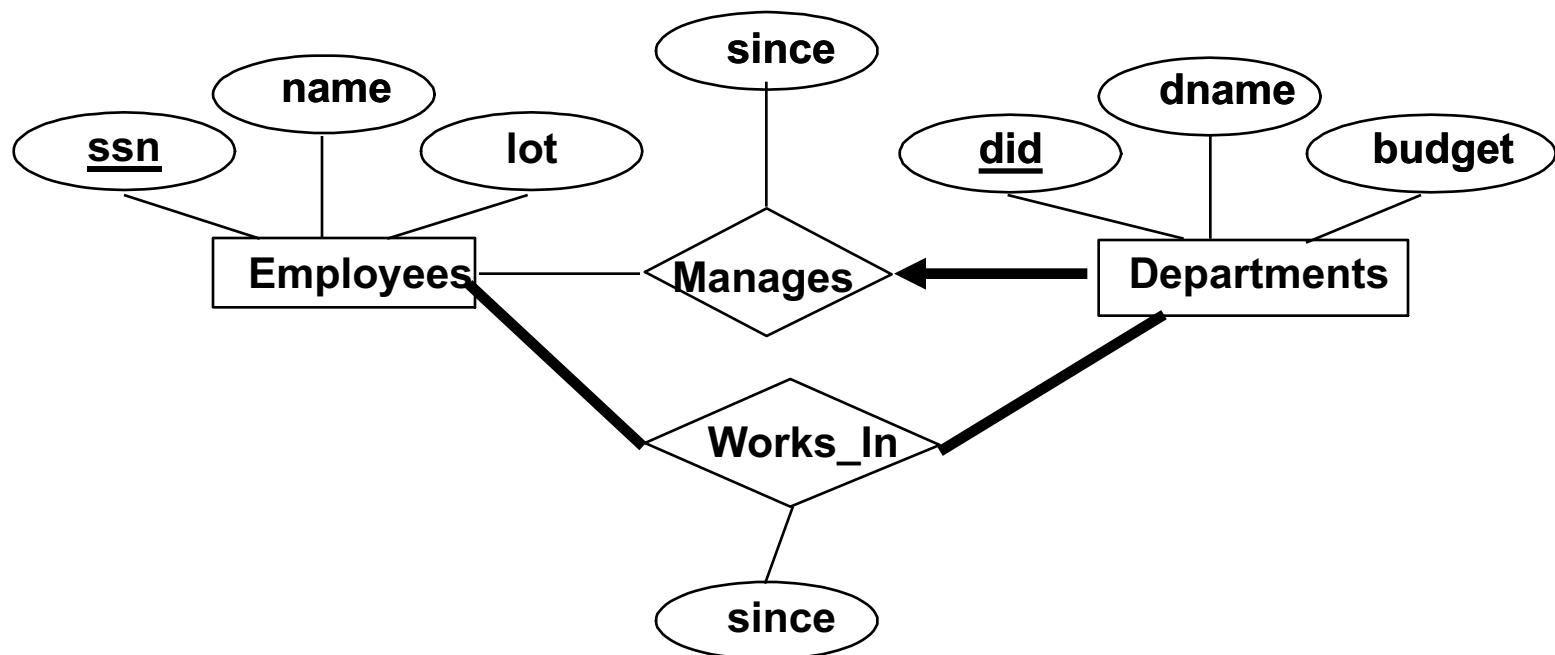
Alternative 2: Embed *Manages* relationship into *Departments* table

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

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

Participation Constraints

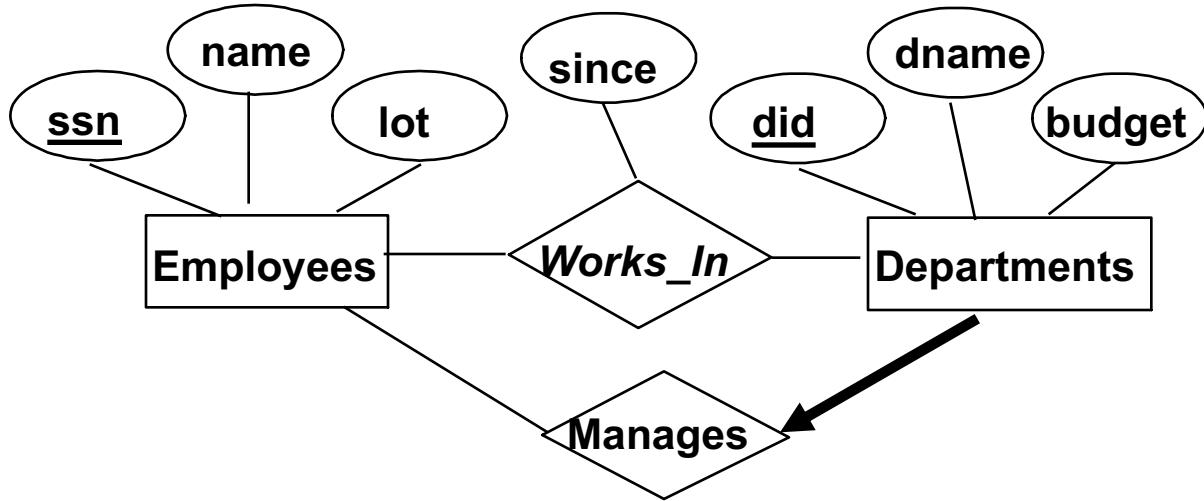
- Does every department have a manager?
 - If so, this is a **participation constraint**: the participation of *Departments* in *Manages* is said to be **total** (vs. **partial**).
 - Every *did* value in *Departments* table must appear in a row of the *Manages* table (with a non-null *ssn* value!)



Participation Constraints in SQL

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

```
CREATE TABLE Department(
    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)
```

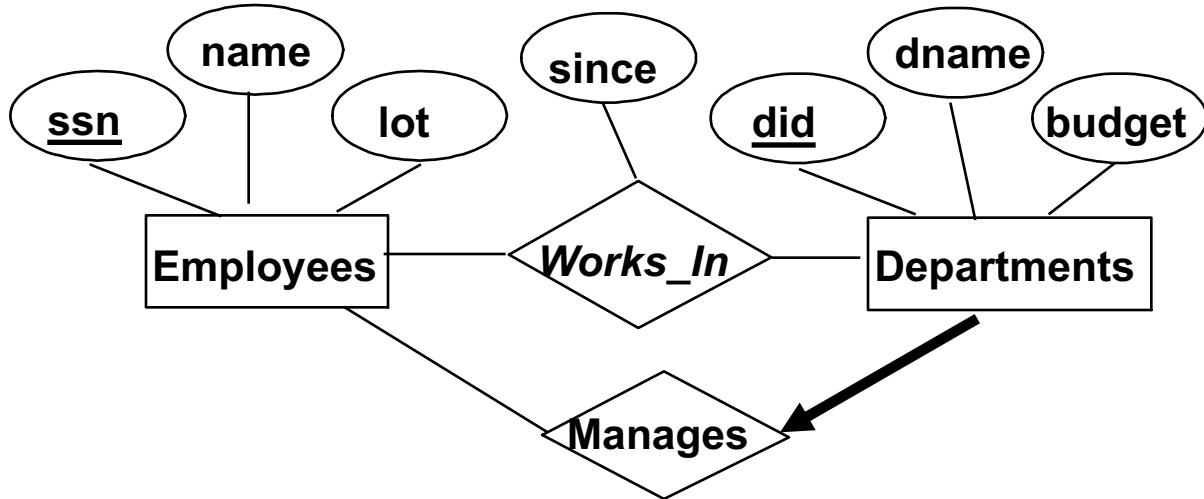


```

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

```

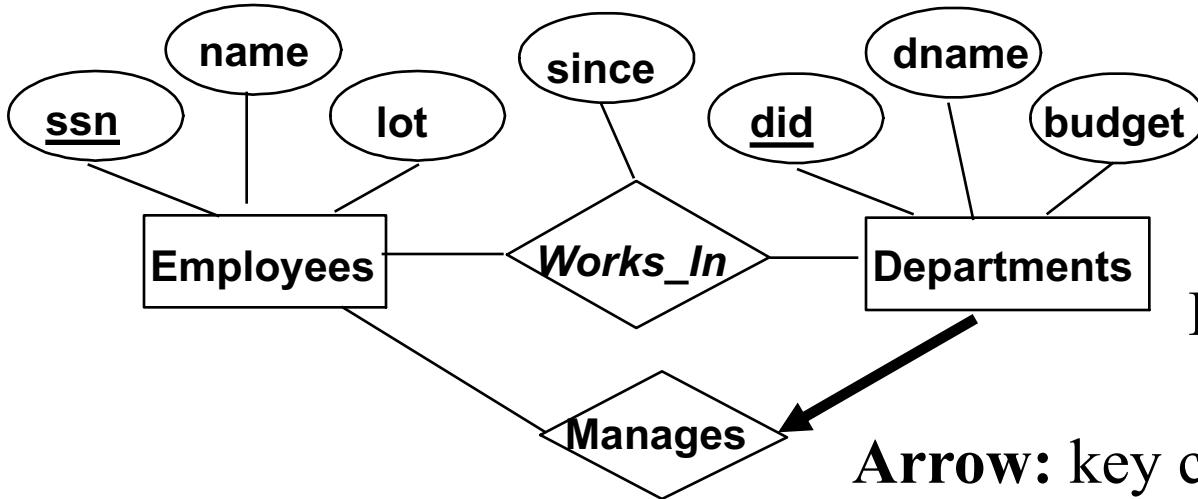
CREATE TABLE Departments(
    did INTEGER,
    dname CHAR(20),
    budget REAL,
    PRIMARY KEY (did)
)
    
```



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

```
CREATE TABLE Departments(
    did INTEGER,
    dname CHAR(20),
    budget REAL,
    PRIMARY KEY (did))
```

```
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)
```



Bold: total participation
at least one manager

Arrow: key constraint

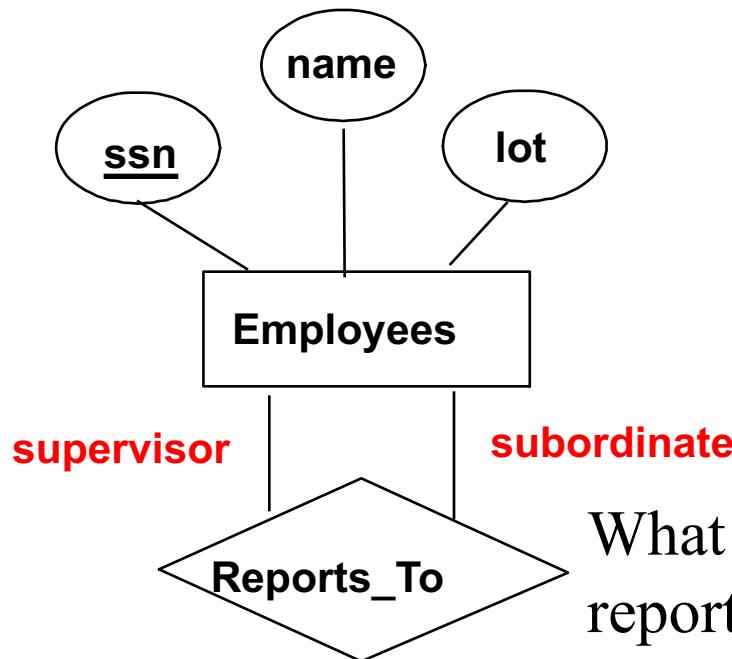
(1-M relationship); at most one manager

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

```
CREATE TABLE Dept_Mans(
    did INTEGER,
    dname CHAR(20),
    budget REAL,
    ssn CHAR(11) NOT NULL,
    since DATE,
    PRIMARY KEY (did),
    FOREIGN KEY (ssn) REFERENCES Employees)
```

```
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)
```

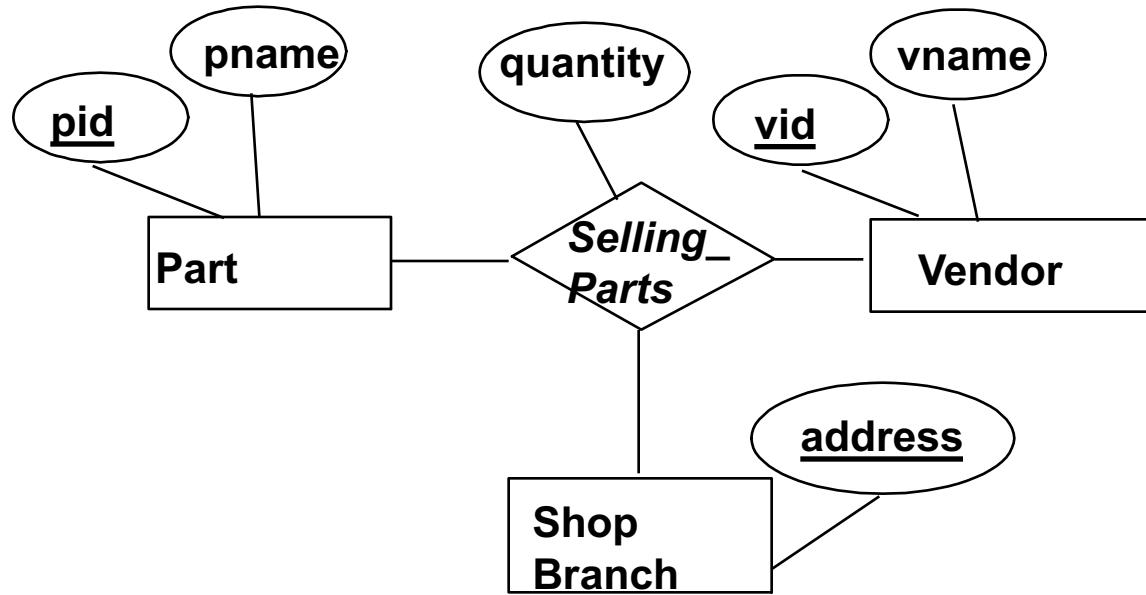
A binary relationship with roles



What if each employee (subordinate) reports to at most one employee (supervisor)?

```
CREATE TABLE Reports_To (
    supervisor_ssn CHAR(11),
    subordinate_ssn CHAR(11),
    PRIMARY KEY (supervisor_ssn, subordinate_ssn ),
    FOREIGN KEY (supervisor_ssn) REFERENCES Employees(ssn),
    FOREIGN KEY (subordinate_ssn) REFERENCES Employees(ssn))
```

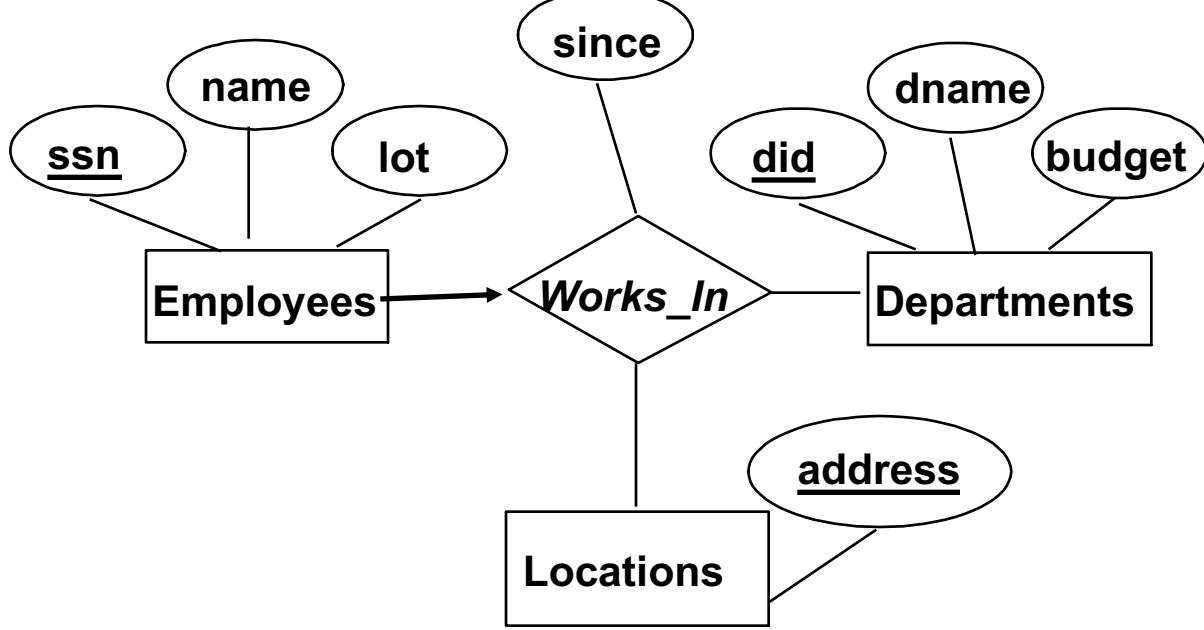
Translating a ternary relationship



Selling_parts(pid, vid, address, quantity)

Translating a ternary relationship

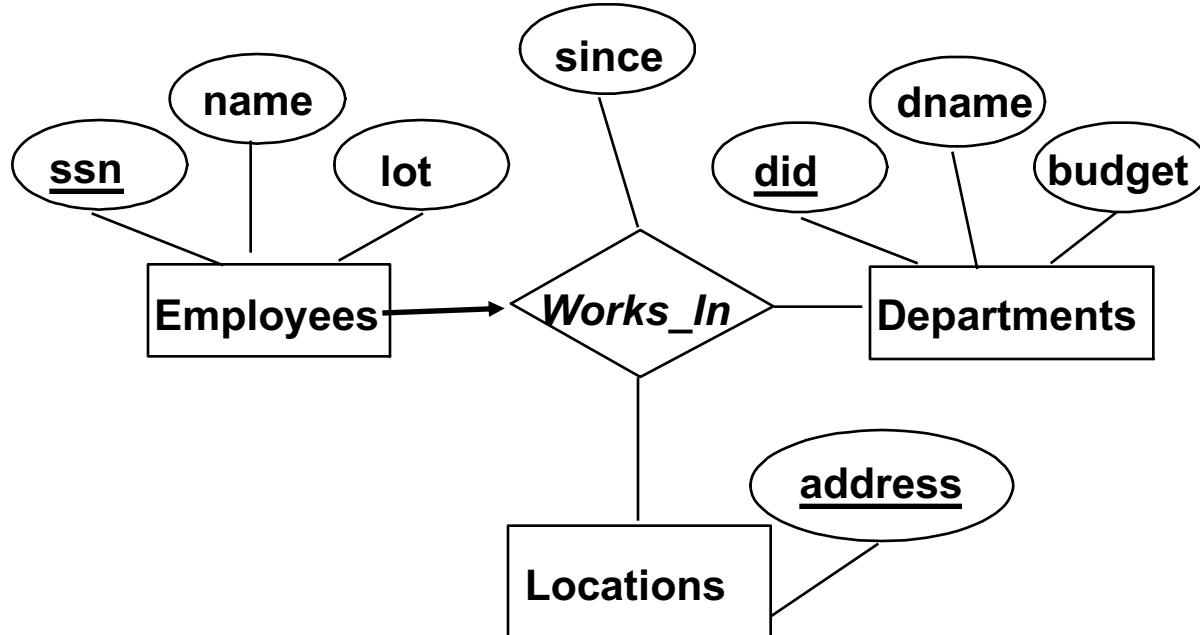
- Assume each employee *works in* at most one department and at a single location (but:
 - each dept can be associated with several locations and employees, and
 - each location can be associated with several departments and employees,



| WorksIn | | |
|------------|------------|----------|
| <u>ssn</u> | <u>did</u> | location |
| 100 | 1 | Ankara |
| 200 | 1 | Ankara |
| 300 | 2 | Ankara |
| 400 | 2 | İstanbul |
| 500 | 3 | İstanbul |

Translating a ternary relationship

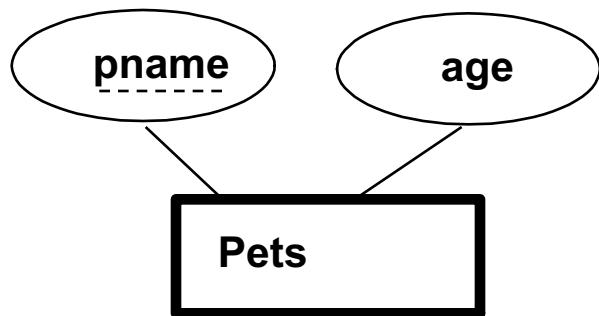
```
CREATE TABLE Works_In(  
    ssn CHAR(11),  
    did INTEGER,  
    address CHAR(20),  
    since DATE,  
    PRIMARY KEY (ssn),  
    FOREIGN KEY (ssn) REFERENCES Employees,  
    FOREIGN KEY (did) REFERENCES Departments  
    FOREIGN KEY (address) REFERENCES Locations )
```



| WorksIn | | | |
|------------|------------|-----------------|--|
| <u>ssn</u> | <u>did</u> | <u>location</u> | |
| 100 | 1 | Ankara | |
| 200 | 1 | Ankara | |
| 300 | 2 | Ankara | |
| 400 | 2 | İstanbul | |
| 500 | 3 | İstanbul | |

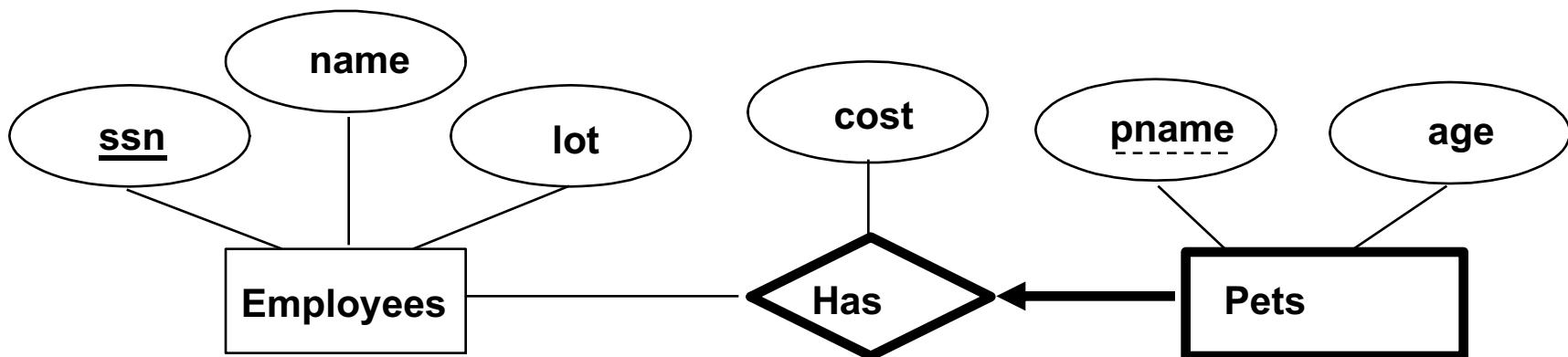
Weak Entities

- A **weak entity** can be identified uniquely only by considering the primary key of another (*owner*) entity.
 - identified by its discriminator attributes (partial key) + PK of another (owner) entity
- Suppose our employees have pets, each pet has a name!



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.



Translating Weak Entity Sets

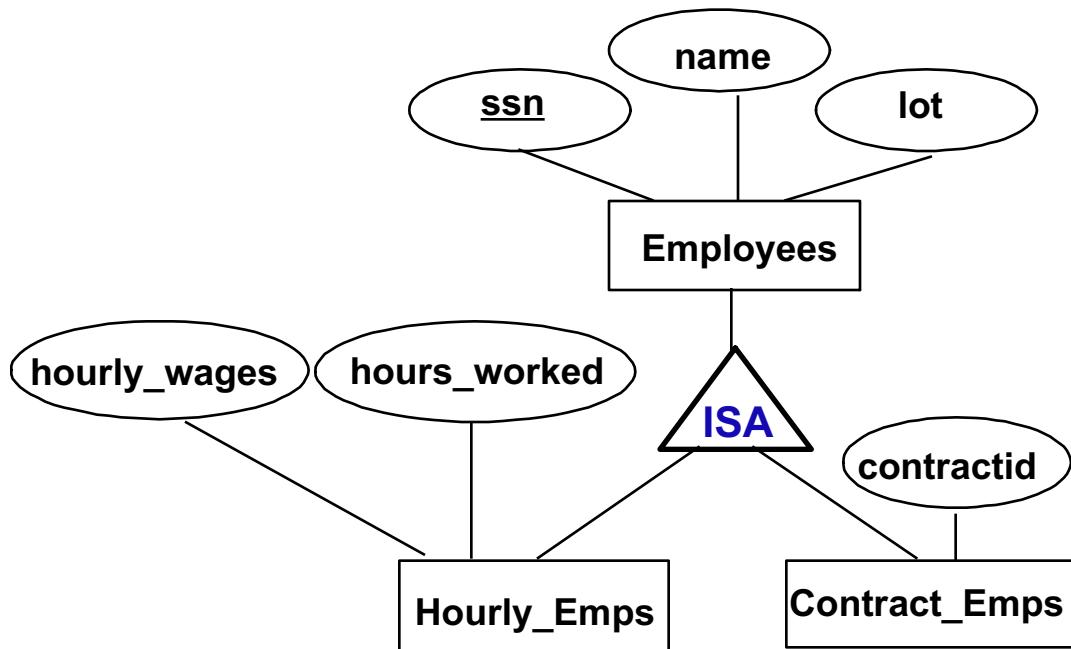
- Weak entity set and identifying relationship set are translated into a **single table**.
 - When the owner entity is deleted, all owned weak entities must also be deleted.

Do I need this?

```
CREATE TABLE Has_Pets (
    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 ('is a') Hierarchies

- As in object-oriented programming languages, attributes are inherited.
 - If we declare **A ISA B**, every A entity is also considered to be a B entity.
- Reasons for using ISA:
 - To add descriptive attributes specific to a subclass.
 - To identify entities that participate in a relationship.



ISA ('is a') Hierarchies

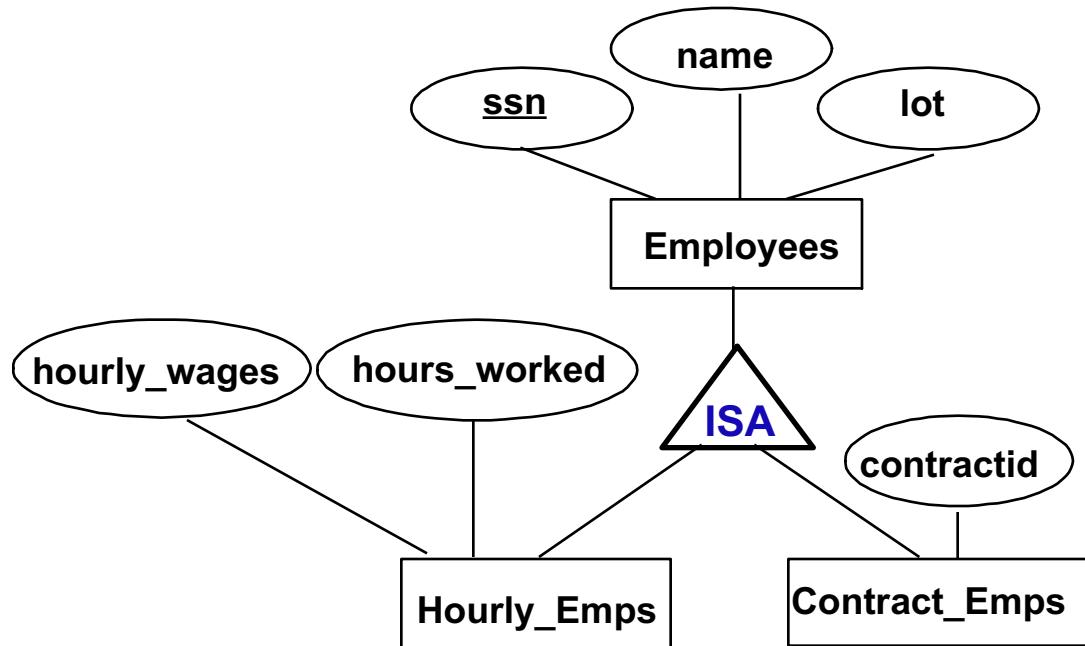
- **Overlap constraints:** Can Joe be an *Hourly_Emps* as well as a *Contract_Emps* entity? (**Allowed/disallowed**)

- **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

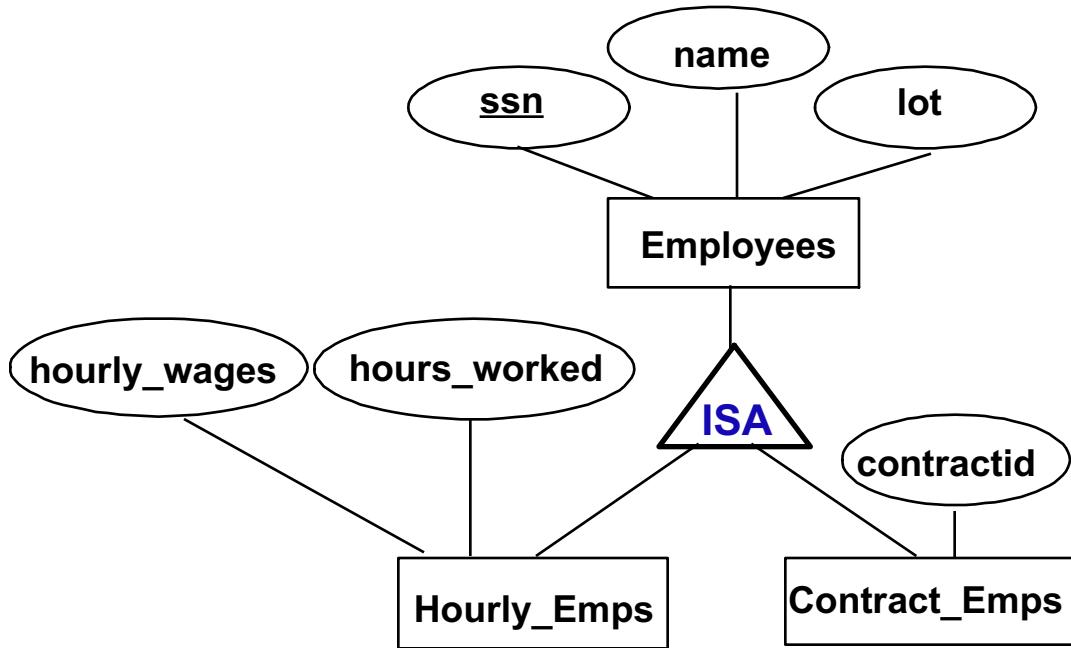
- **General approach:**
 - 3 relations: Employees, Hourly_Emps and Contract_Emps.
 - Hourly_Emps: Every employee is recorded in Employees. For hourly emps, extra info recorded in Hourly_Emps (hourly_wages, hours_worked, ssn); must delete Hourly_Emps tuple if referenced Employees tuple is deleted).
 - 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.

Translating ISA Hierarchies to Relations



- **First approach:**
 - 3 relations.
 - Employees(ssn, name, lot)
 - Hourly_Emps(ssn, **hourly_wages**, **hours_worked**)
 - Contract_Emps(ssn, **contractid**)
 - In Hourly_Emps & Contract_Emps tables:
 - FOREIGN KEY (ssn) REFERENCES Employees ON DELETE CASCADE

Translating ISA Hierarchies to Relations

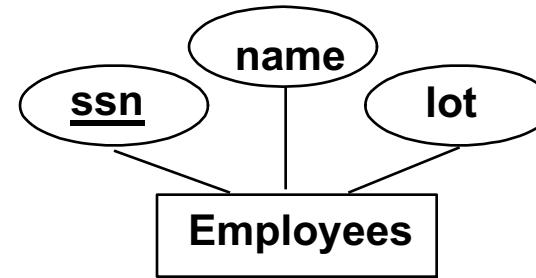


- **Second approach:**
 - 2 relations.
 - Hourly_Emps(ssn, name, lot, **hourly_wages**, **hours_worked**)
 - Contract_Emps(ssn, name, lot, **contractid**)
 - Not applicable if not "covering"

Aggregation

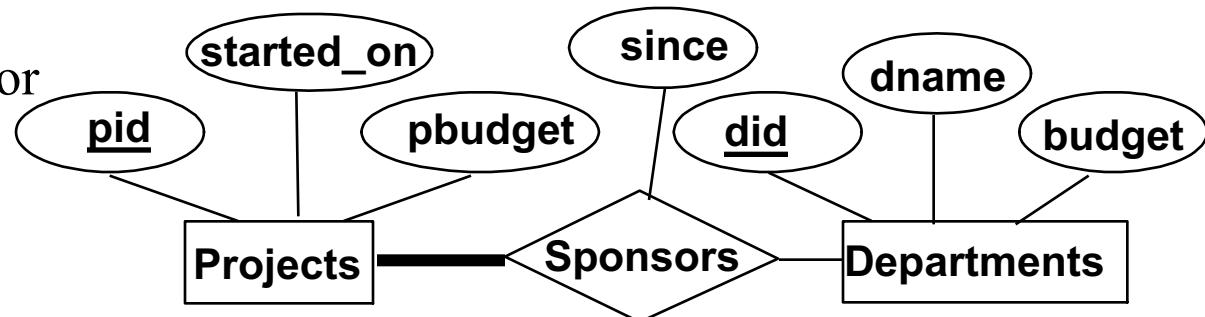
Suppose:

- entity set *Projects*
- each *Projects* is sponsored by at least one *Departments*
- each *Departments* that sponsors a *Projects* might assign employees to monitor sponsorship



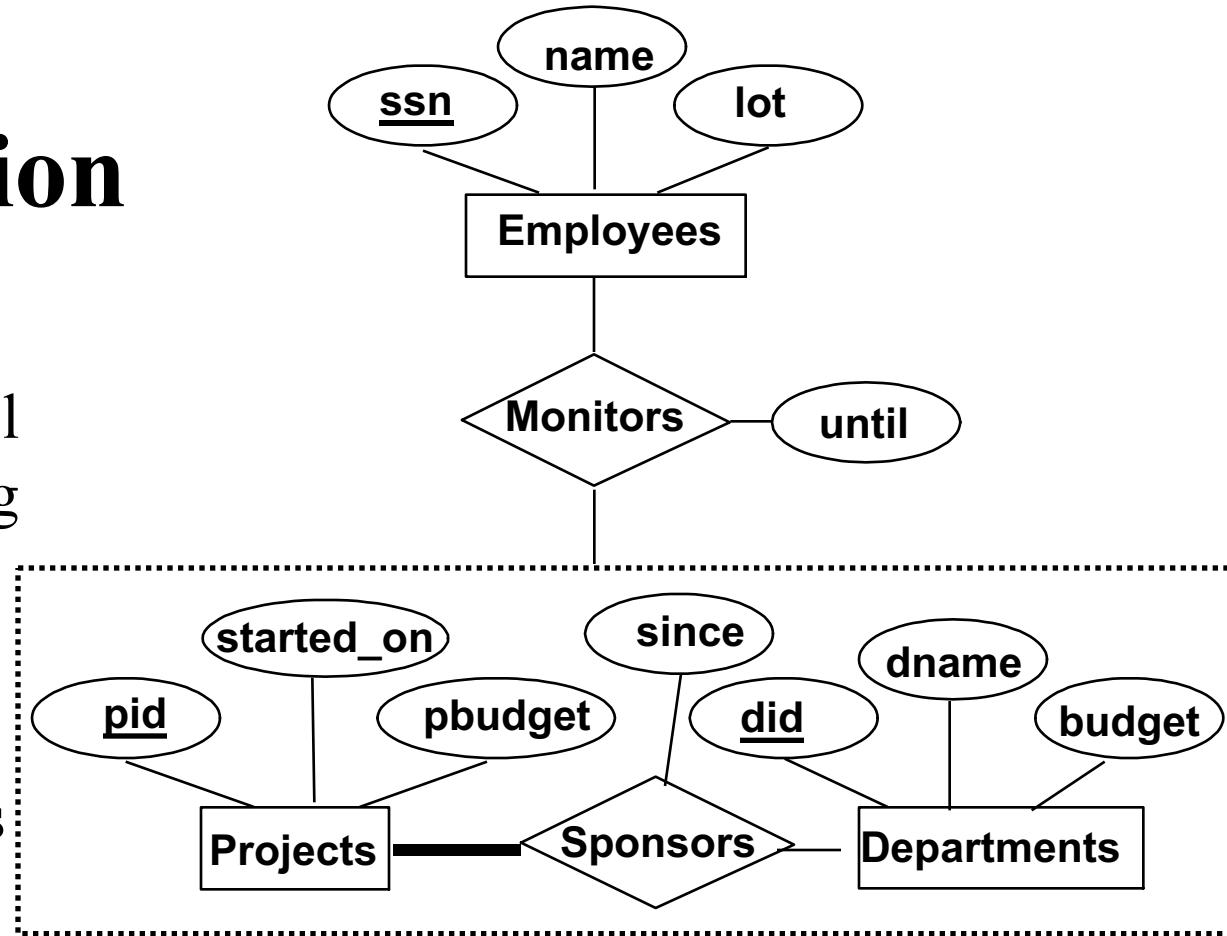
Intuitively...

- *Monitors* should be a relationship set that associates a *Sponsors* relation (versus a *Projects* or *Departments*) with an *Employees* entity.

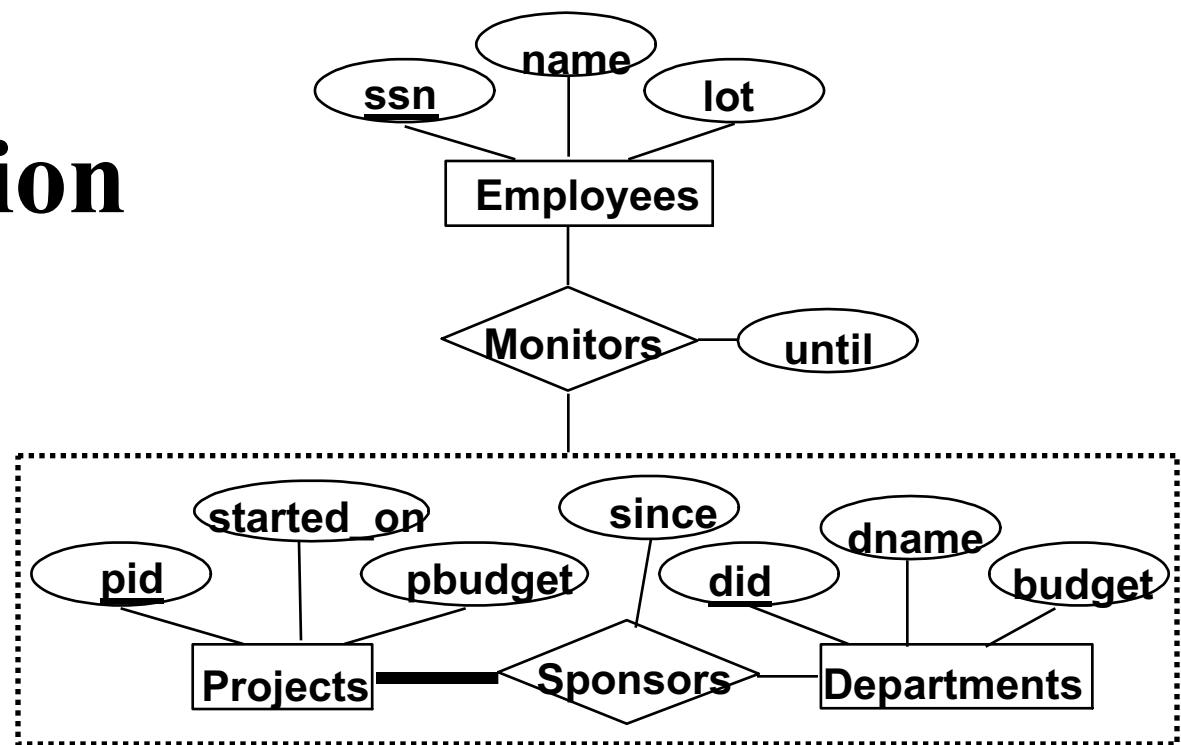


Aggregation

- Aggregation is used when we have to model a relationship involving (entity sets and) a relation: a *relationship set*.
- **Aggregation** allows us to treat a relationship set as an entity set for purposes of participation in (other) relationships.



Aggregation



➤ Mapping to Relational Model:

Employees(ssn, name, lot)

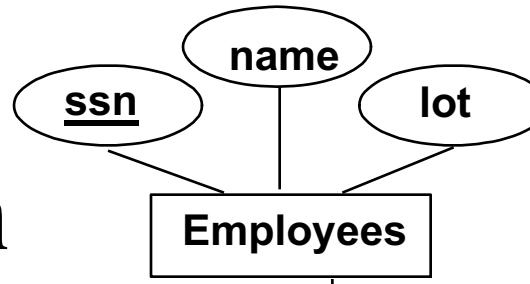
Projects(pid, started_on, pbudget)

Departments(did, dname, budget)

Sponsors(pid, did, since)

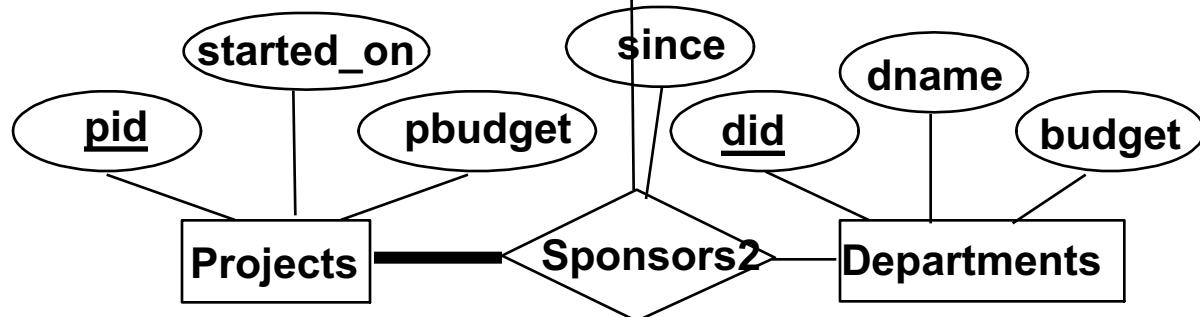
Monitors(ssn, pid, did, until)

Aggregation vs. Ternary relation



Suppose:

- No need to record *until* attribute of *Monitors*
- Then...
 - Can use ternary relationship *Sponsors2*



But suppose we have constraint that:

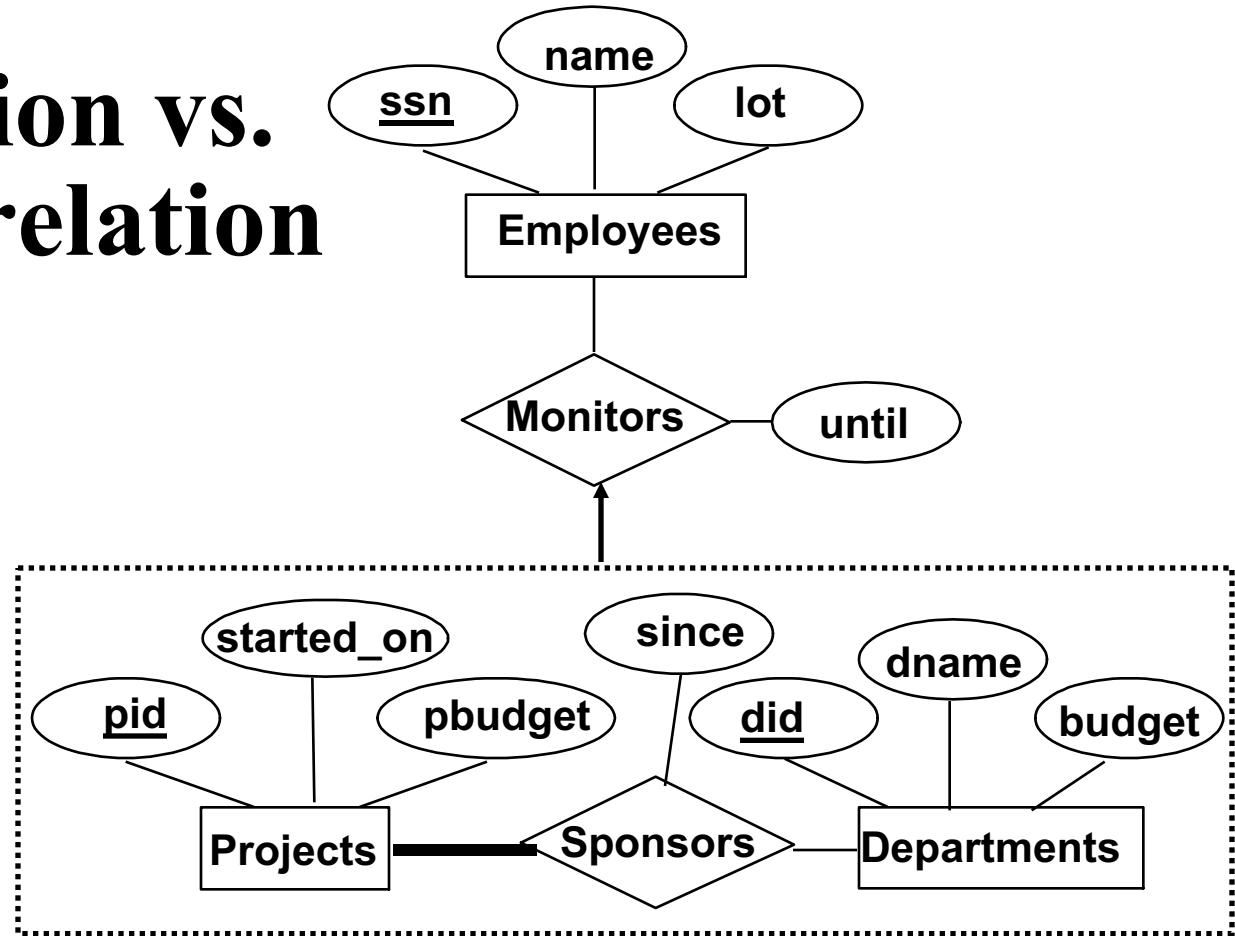
Each sponsorship (of a project by a department) be monitored by at most one employee?

Then...

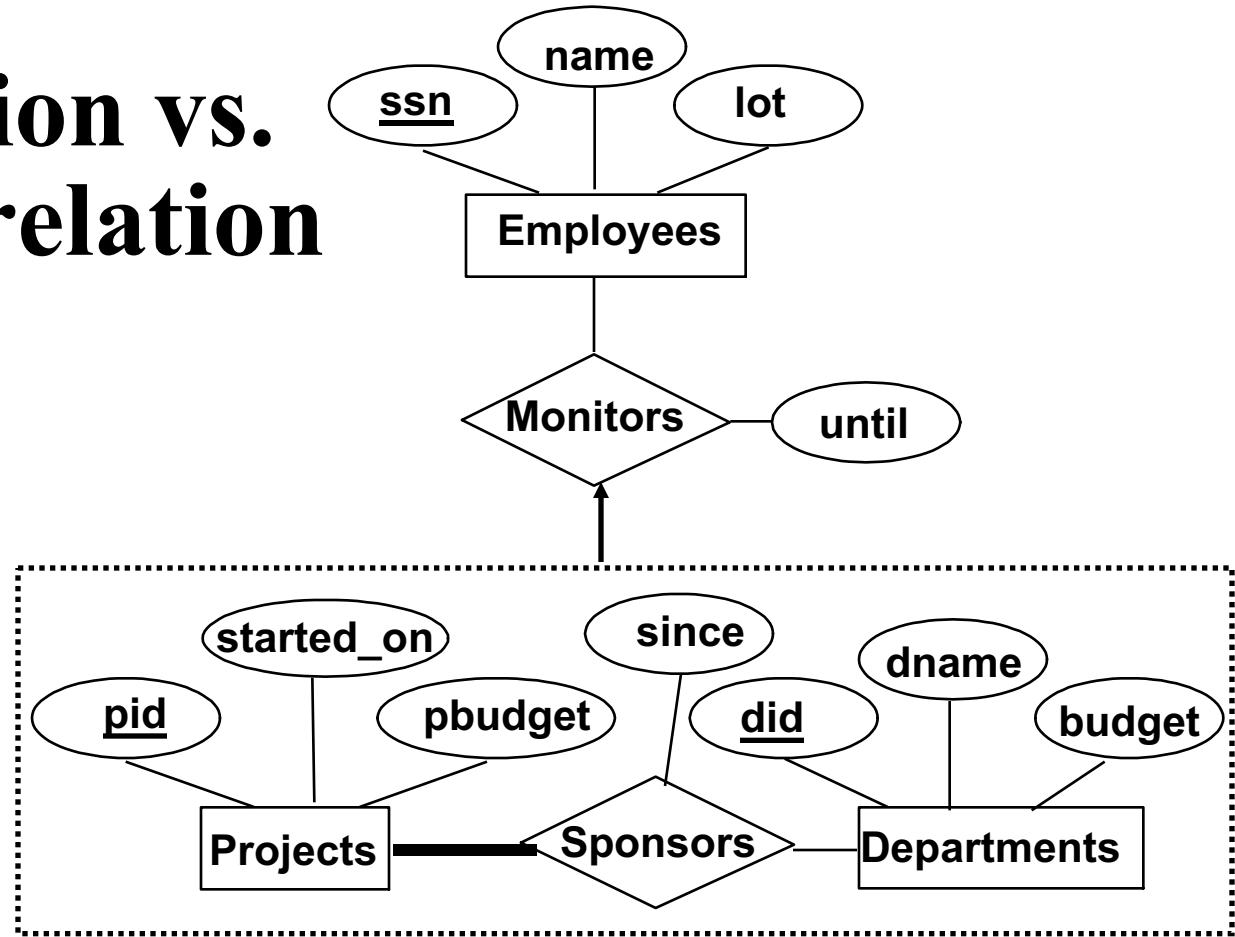
Can't do it with *Sponsors2* → Need aggregated relationship *Sponsors*

Aggregation vs. Ternary relation (Contd.)

- *Monitors* is a distinct relationship, with a descriptive attribute (*until*).
- Also, can say that each sponsorship is monitored by at most one employee.



Aggregation vs. Ternary relation (Contd.)



➤ Mapping:

Projects(pid, started_on, pbudget)

Departments(did, dname, budget)

Sponsors(pid, did, since)

Employees(ssn, name, lot)

Monitors(pid, did, ssn, until)

Where does pid and did refer to?

Conceptual Design Using the ER Model

➤ 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?

➤ Constraints in the ER Model:

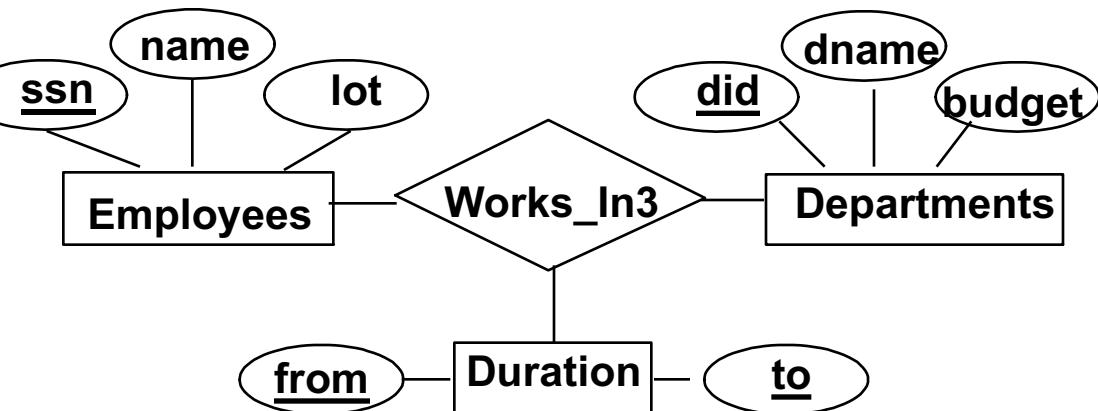
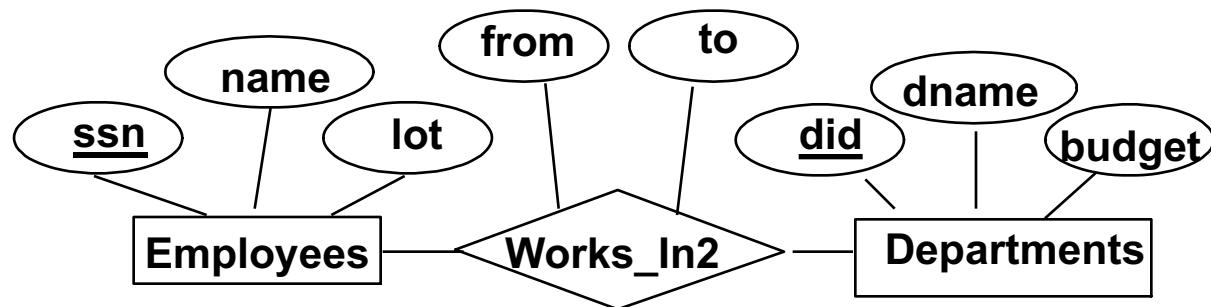
- A lot of data semantics can (and should) be captured.
- But some constraints cannot be captured in ER diagrams.

Entity vs. Attribute

- 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).

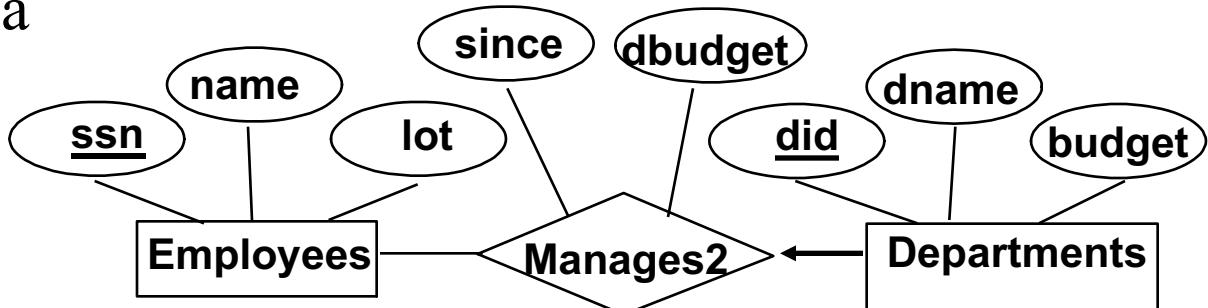
Entity vs. Attribute (Cont.)

- Works_In2 does not allow an employee to work in a department for two or more periods.
- Similar to the problem of wanting to record several addresses for an employee: we want to record several values of the descriptive attributes for each instance of this relationship.



Entity vs. Relationship

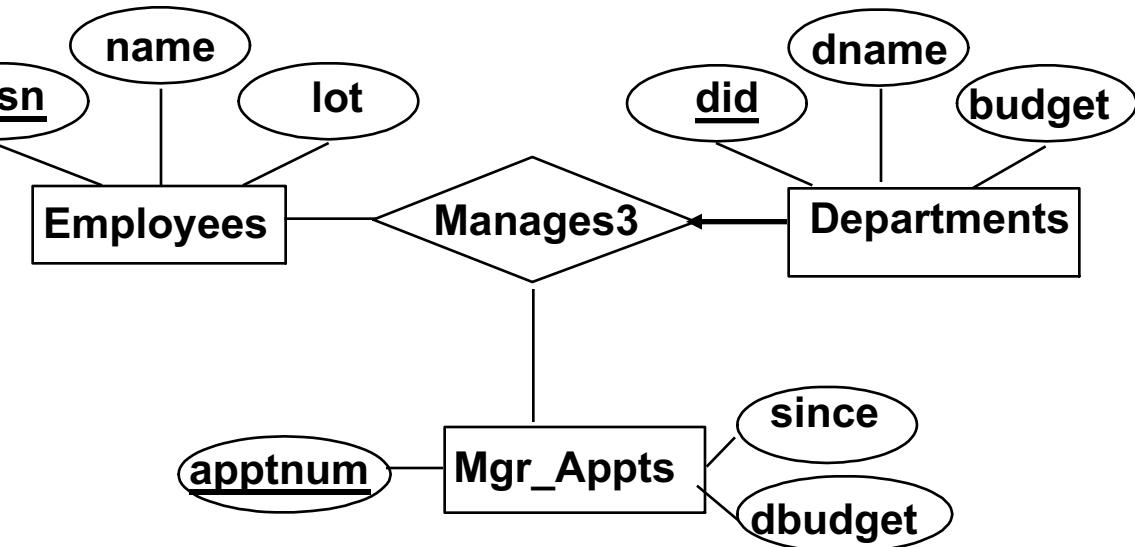
- First ER diagram OK if a manager gets a separate discretionary budget for each dept.



- What if a manager gets a discretionary budget that covers all managed depts?

- Redundancy of budget, which is stored for each dept managed by the manager.

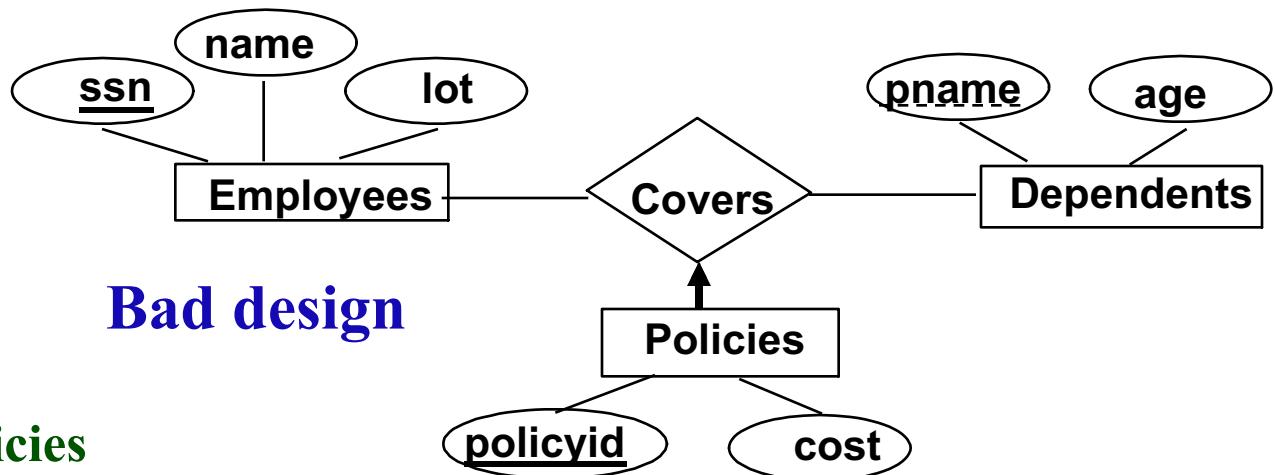
Misleading: suggests dbudget tied to managed dept.



Binary vs. Ternary Relationships

Suppose that:

- a policy cannot be owned jointly by two or more Employee:
 - Key constraint on Policies would mean policy can only cover 1 dependent!

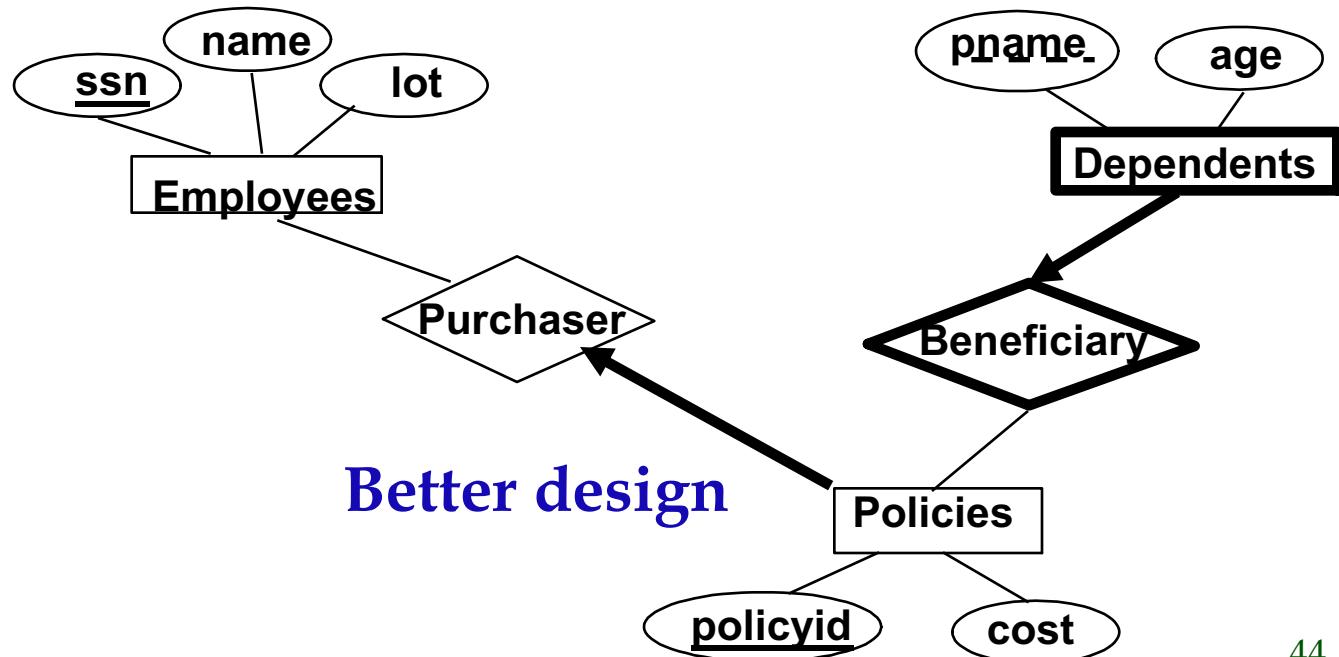


- Suppose that:
 - Every policy must be owned by some employee
 - Acceptable if each policy covers at least one dependent

Binary vs. Ternary Relationships (Cont.)

➤ What are the additional constraints in this 2nd diagram?

- Policy cannot be owned jointly by two or more employees
- Every policy must be owned by some employee
- Dependent is a weak entity set, and each dependent entity is uniquely identified by taking pname in conjunction with the policyid of a policy.
- Dependents cannot be beneficiary of more than one policy.



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.
- What if Policies is a weak entity set?

```
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)
```

Binary vs. Ternary Relationships (Cont.)

- Previous example illustrated a case when two binary relationships were better than one ternary relationship.
- An example in the other direction: a ternary relation **Contracts** relates entity sets **Parts**, **Departments** and **Suppliers**, and has descriptive attribute *qty*.
- No combination of binary relationships is an adequate substitute for at least two reasons:
 - S “can-supply” P, D “needs” P, and D “deals-with” S does not imply that D has agreed to buy P from S.
 - How do we record *qty*?

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.

Example

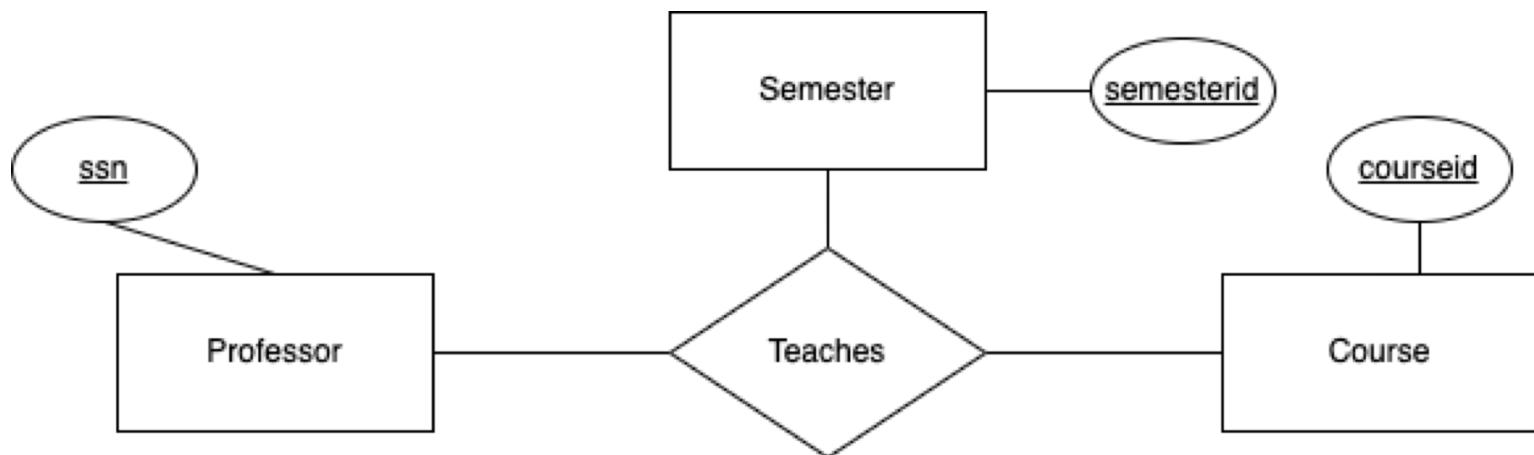
- A University database contains information about **professors** (identified by social security number, or *SSN*) and **courses** (identified by *courseid*).
Professors teach courses.
- Each of the following situations concerns the **Teaches** relationship set. For each situation, draw an ER diagram that describes it (assuming that no further constraints hold).

Example (cont')

- 1) Professors can teach the same course in several semesters, and each offering must be recorded.
- 2) Professors can teach the same course in several semesters, and only the most recent such offering needs to be recorded. (Assume this condition applies to all subsequent questions.)
- 3) Every professor must teach some course
- 4) Every professor teaches exactly one course (no more, no less).
- 5) Every professor teaches exactly one course (no more no less), and every course must be taught by some professor.
- 6) Now assume that certain courses can be taught by a team of professors jointly, but it is possible that no one professor in a team can teach the course. Model this situation introducing additional entity sets and relationship sets if necessary.

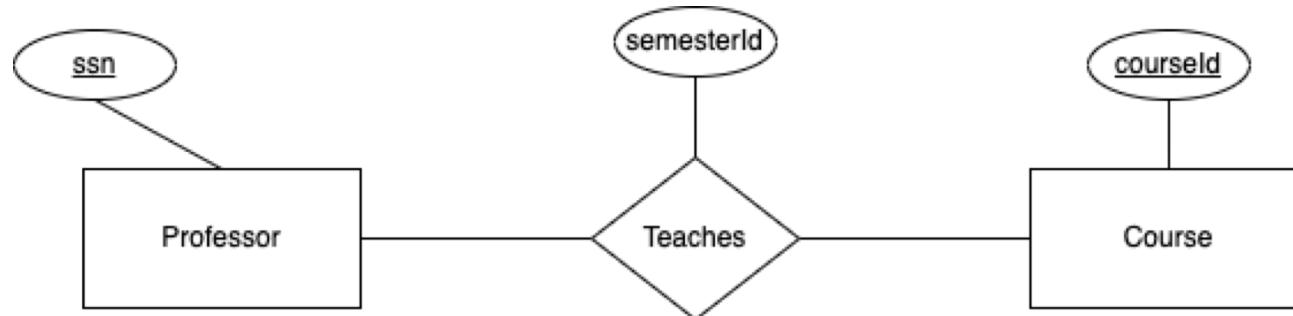
Example (cont')

- 1) Professors can teach the same course in several semesters, and each offering must be recorded.



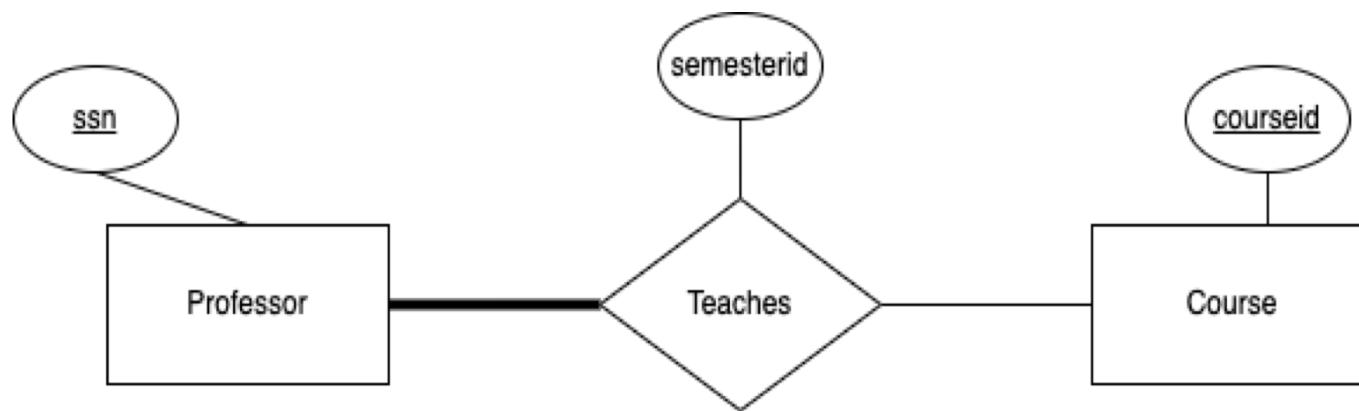
Example (cont')

- 2) Professors can teach the same course in several semesters, and only the most recent such offering needs to be recorded. (Assume this condition applies to all subsequent questions.)



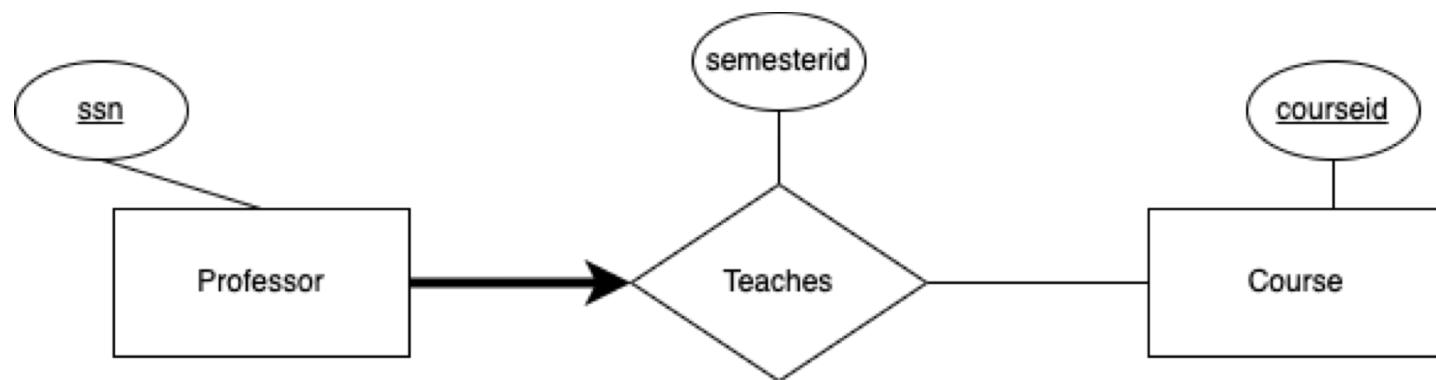
Example (cont')

- 3) Every professor must teach some course



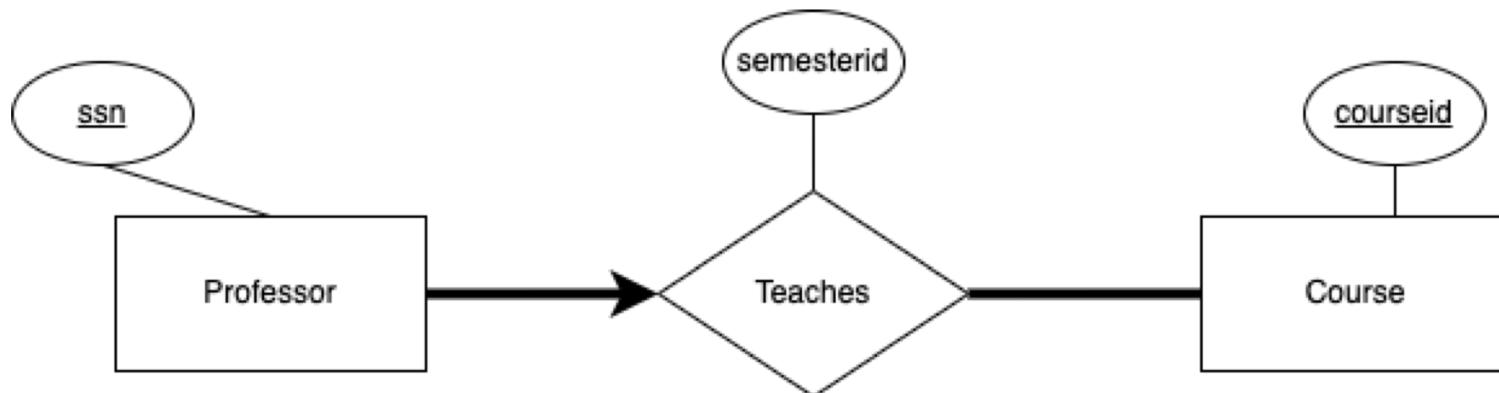
Example (cont')

- 4) Every professor teaches exactly one course (no more, no less).



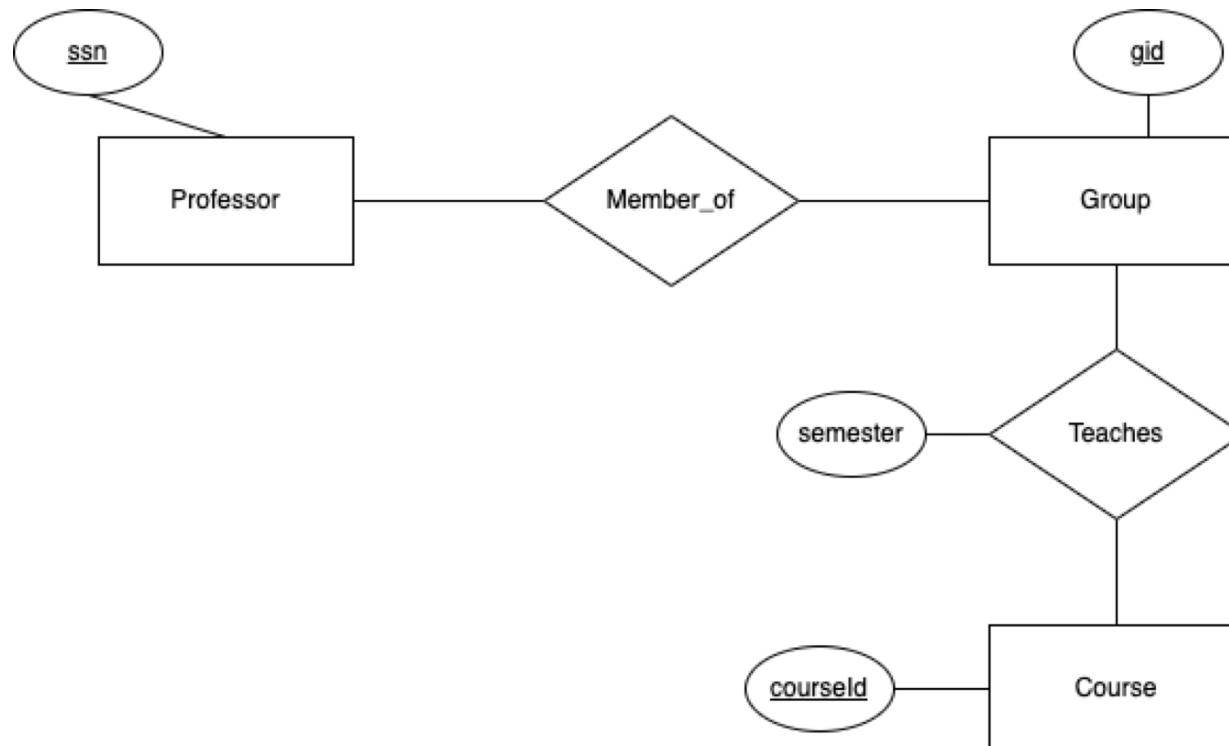
Example (cont')

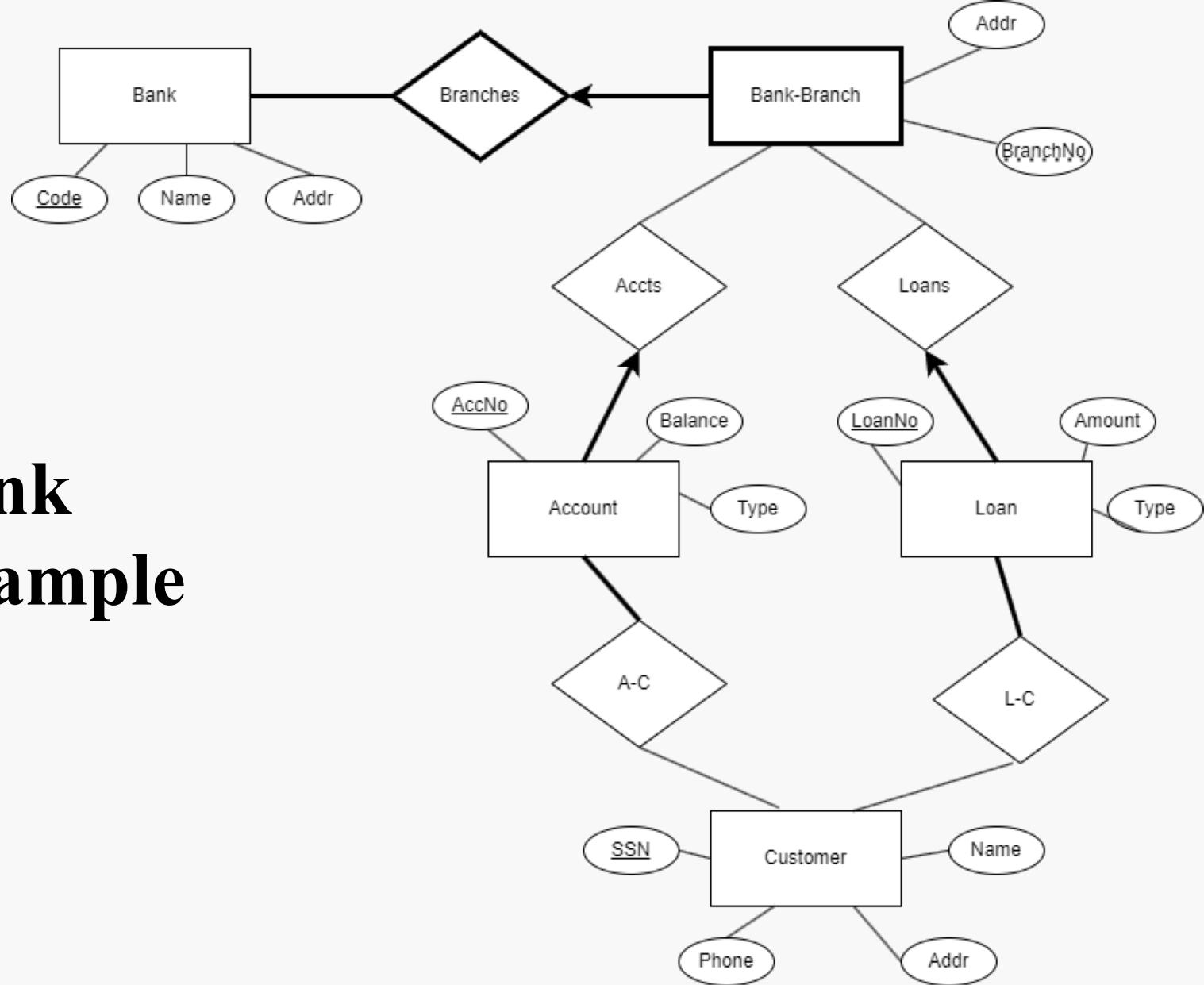
- 5) Every professor teaches exactly one course (no more no less), and every course must be taught by some professor.



Example (cont')

- 6) Now assume that certain courses can be taught by a team of professors jointly, but it is possible that no one professor in a team can teach the course. Model this situation introducing additional entity sets and relationship sets if necessary.





Bank Example