# The Entity-Relationship Model

Chapter 2

# Databases Model the Real World

- **Data Model:**

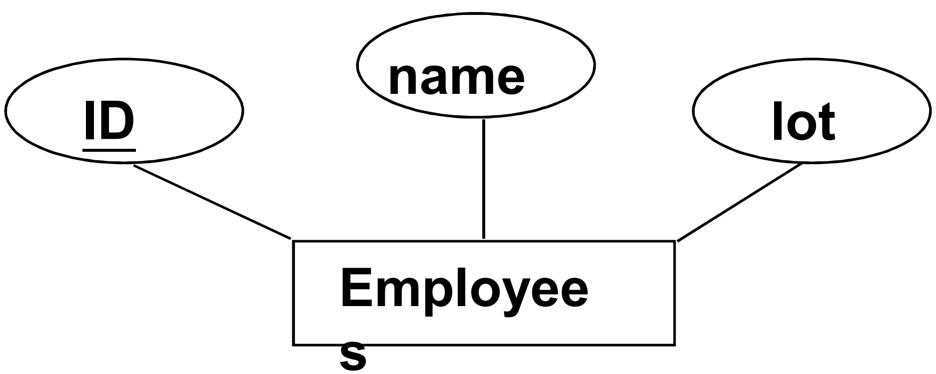
  A *data Model* is a collection of conceptual tools for describing data, data relationships, data semantics and consistency constraints.

- **E-R Model:**

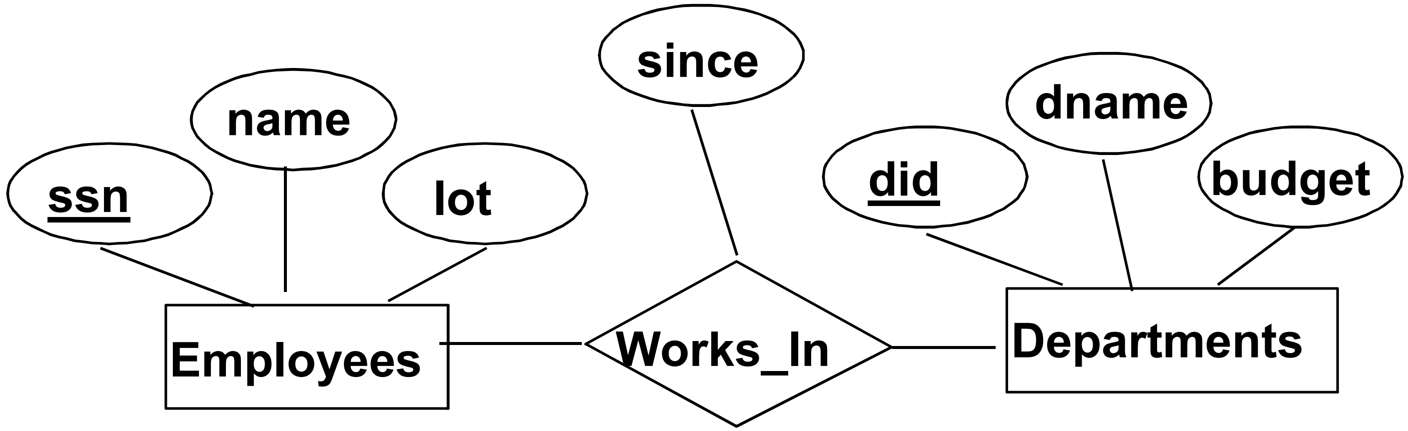
  The Entity-Relationship data model is based on a perception of a real world that consists of a set of basic objects called entities and of relationships among these objects.

# E-R Model

- ***Entity:*** **An entity is an object that exists in the real world and is distinguishable from other objects. An entity is described using a set of *__attributes__*.**
- ***Attribute:*** **Attributes are descriptive properties possessed by each member of an entity set.**
- ***Entity Set*****: A collection of all entities of the same type is an entity set. E.g. all employees.**
  - All entities in an entity set have the same set of attributes.



# ER Model Basics




- ***Relationship***: A relationship is an association among two or more entities. relationships can have their own attributes.

  Relationships indicate a meaningful connection between two entity types.

- ***Relationship Set***: The Collection of all similar relationships of the same type is called relationship set.

# Attribute & Constraint

- Attributes are facts, aspects, properties, or details about an entity
  - Students have IDs, names, courses, addresses, …
  - Modules have codes, titles, credit weights, levels, …

**Constraint:**

Constraints are restrictions on legal relation states

# Types of Attribute

- **Simple-value attribute:**

  These attributes are not divided into subparts.

- **Composite attributes:**

  They can be divided into subparts.

  Example: name which contain first-name, middle-name and last-name.

- **Single-valued attribute:**

  The attribute which specify only one number is called single-valued attribute.

  Exam: loan-number which indicate only one loan number.

- **Multi-valued attribute:**

  When an attribute has a set of values for specific entity is called multi-valued attribute.

  Exam: Phone-number- An employee may have one or more phone numbers.

# Attribute

- **Derived Attribute:** The value for this type of attribute can be derived from the values of other related attributes or entities.

**Example:** ***Date-of birth*** --- we can calculate age from ***date-of-birth*** and the current date.Thus age is a derived attribute.

***Loans_held,***which represents how many loans a customer has from the bank.

We can derive the value for this attribute by counting the number of loan entities associated with that customer.

- **Null value:** An attribute takes a null value when an entity does not have a value for it,that means the value does not exist for the entity.

**Example: middle_name –** One may have no middle name.

# Mapping Cardinalities

- **Mapping cardinalities, or cardinality ratios, express the number of entities to which another entity can be associated via a relationship set.**

  **---**Mapping cardinalities are most useful in describing binary relationship sets.

- **There are four types of binary relationship in E-R model:**

➢ ***One to many* :**

An entity in A is associated with at most one entity in B and an entity in B is associated with at most one entity in A.

➢ ***One to Many*:**

An entity in A is associated with any number of(zero or more) entities in B.An entity in B can be associated with at most one entity in A.

# Types of E-R model

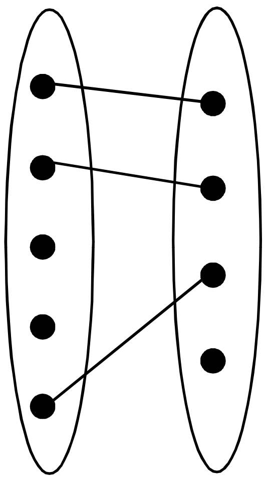
- **Many to One :**

  An entity in A is associated with at most one entity in B. An entity in B, however, can be associated with any number (zero or more) of entities in A.

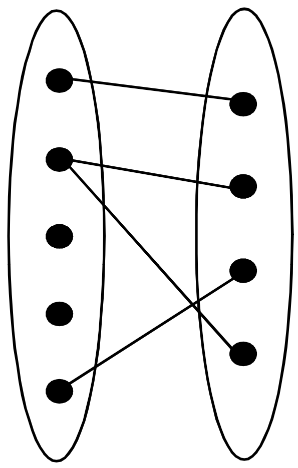
- **Many to Many :**

  An entity in A is associated with any number (zero or more) of entities in B and an entity in B is associated with any number (zero or more) of entities in A.

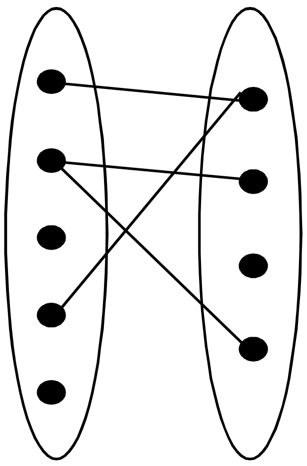
# Types of E-R model



One to one



one-to Many



Many-to-Many

# Keys

- **The key allow us to identify a set of attributes that suffice to distinguish entities from each other.**
- Keys are used to identify particular entities.
- Keys also help uniquely identify relationships and this distinguish relationships from each other.
- **There are many types of keys:**

- **Superkey**
- **Candidate key**
- **Primary key**
- **Foreign key**

# Superkey

- **A superkey is a set of one or more attributes that,taken collectively allow us to identify uniquely an entity in the entire set.**

  **--**Example: customer_id attribute of the entity set customer is sufficient to distinguish one customer entity to another.

  --The combination of customer_name and customer_id is a superkey for the entity set customer.

  --The customer_name attribute of customer is not a superkey,because several people might have the same name.

# Keys

- **Candidate Key:**

✓ A candidate is a subset of a super key. A candidate key is a single field or the least combination of fields that uniquely identifies each record in the table.

✓ Every table must have at least one candidate key but at the same time can have several.

Candidate Keys

| StudentId | firstName | lastName | courseId |
|---|---|---|---|
| L0002345 | Jim | Black | C002 |
| L0001254 | James | Harradine | A004 |
| L0002349 | Amanda | Holland | C002 |
| L0001198 | Simon | McCloud | S042 |
| L0023487 | Peter | Murray | P301 |
| L0018453 | Anne | Norris | S042 |

## Primary key

- ❖ It is a candidate key that is chosen by the database designer to identify entities with in an entity set. Primary key is a single attribute or combination of attributes that uniquely defines a database record

-- The primary key should be chosen such that its attributes are never,or very rarely,changed.

Primary Keys

| StudentId | firstName | lastName | courseId |
|---|---|---|---|
| L0002345 | Jim | Black | C002 |
| L0001254 | James | Harradine | A004 |
| L0002349 | Amanda | Holland | C002 |
| L0001198 | Simon | McCloud | S042 |
| L0023487 | Peter | Murray | P301 |
| L0018453 | Anne | Norris | S042 |

# Foreign Key

- **A** foreign key **is a field (or collection of fields) in one table that uniquely identifies a row of another table.**

- A foreign key is generally a primary key from one table that appears as a field in another where the first table has a relationship to the second.

  Example: if we had a table A with a primary key X that linked to a table B where X was a field in B, then X would be a foreign key in B.

- **The table containing the foreign key is called the** referencing **or** child table**.**
- **The table containing the candidate key is called the** referenced **or** parent table**.**

# Foreign key

| studentId | firstName | lastName | courseId |
|---|---|---|---|
| L0002345 | Jim | Black | C002 |
| L0001254 | James | Harradine | A004 |
| L0002349 | Amanda | Holland | C002 |
| L0001198 | Simon | McCloud | S042 |

Foreign Keys

Relationship

Primary Keys

| courseId | courseName |
|---|---|
| A004 | Accounts |
| C002 | Computing |
| P301 | History |
| S042 | Short Course |

- **Secondary key:** All the rest of candidate keys .
- **Composite Key:**

  Composite key consists of more than one attributes.
- **Tuple:**

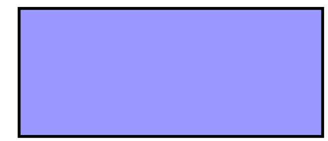
  In the context of databases, a tuple is one record (one row)

  In other word,Tuples are *unordered* sets of known values with names.

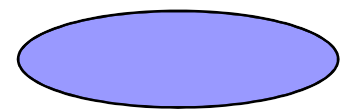
# E-R diagram

- E-R diagram can express the overall logical structure of a database graphically.E-R diagrams are simple and clear.

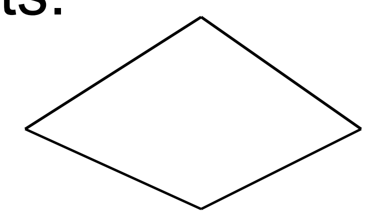
✓ Rectangles – which represent entity sets.



✓ Ellipses – which represent attributes.

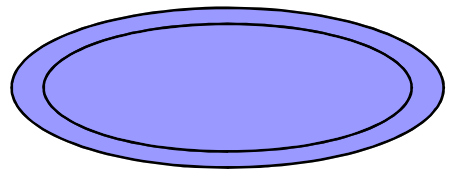


✓ Diamond – which represent relationship sets.



- ✓ Lines-which link attributes to entity sets and entity sets to relationship sets.

- ✓ Double ellipses- which represent maltivalued attributes.

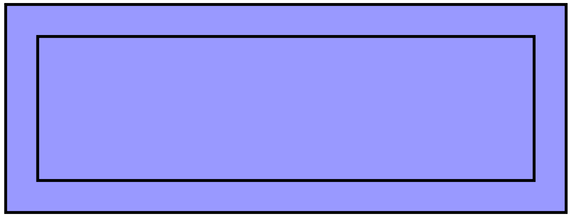


- ✓ Dashed ellipses-which denote derived attributes.

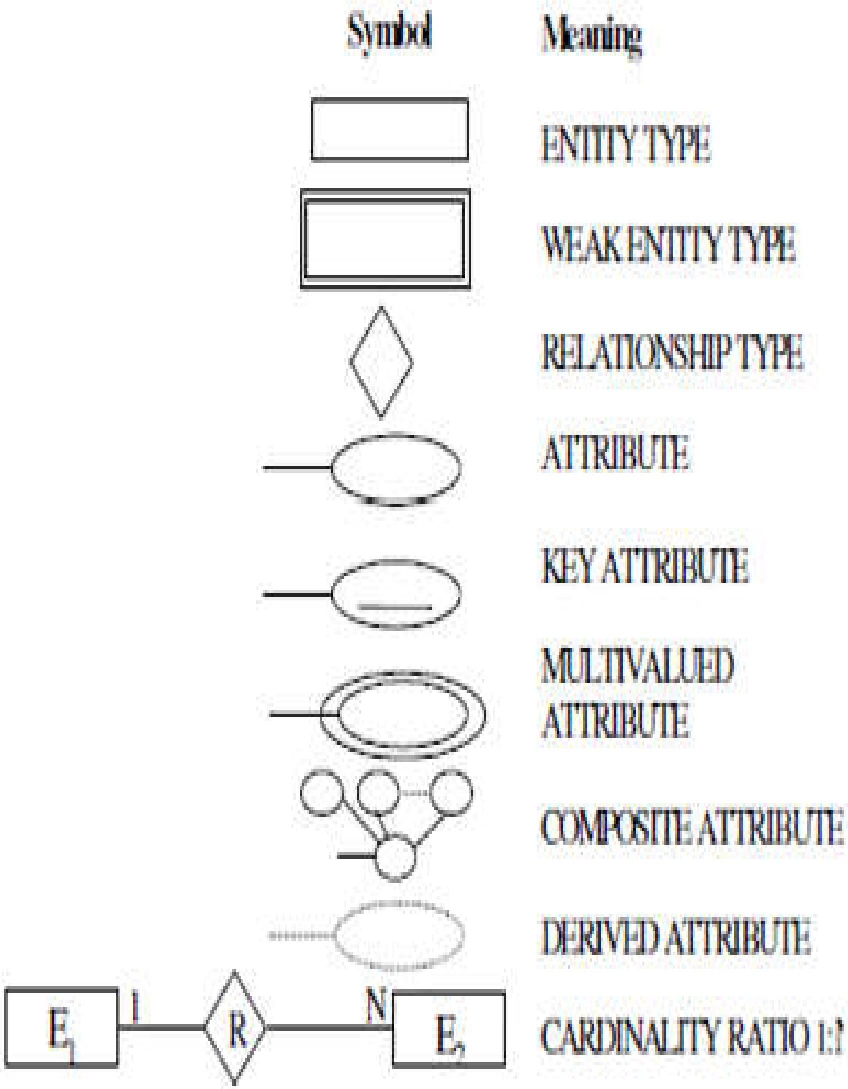
## E-R Diagram

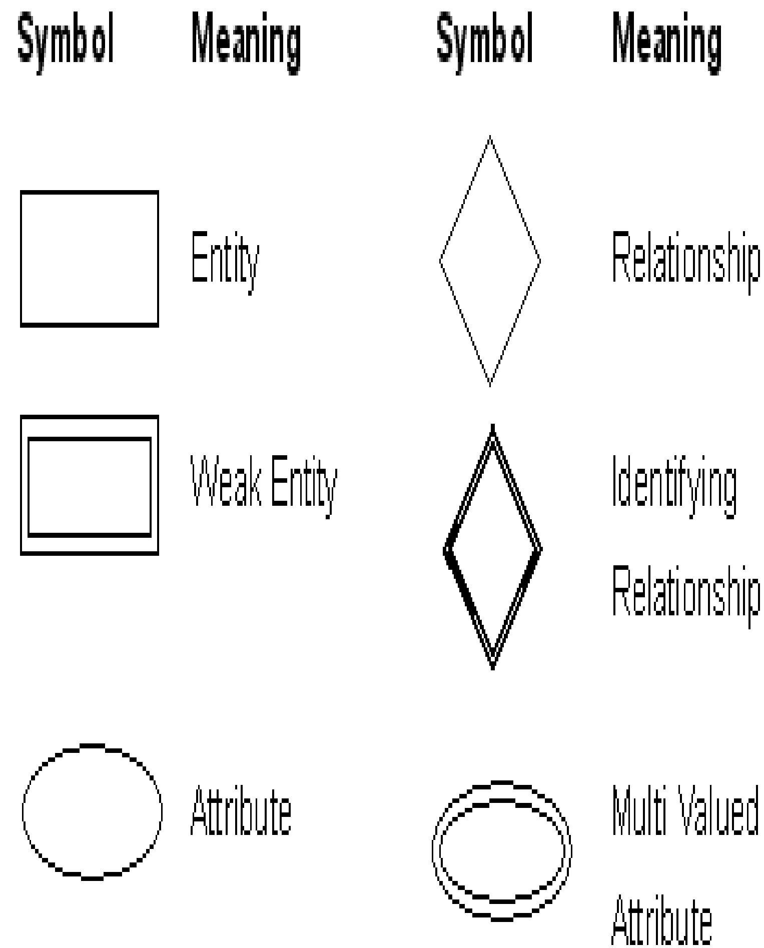
- ✓ Double lines-which indicate total participation of an entity in a relationship set.

- ✓ Double rectangle:-which represent weak entity sets .



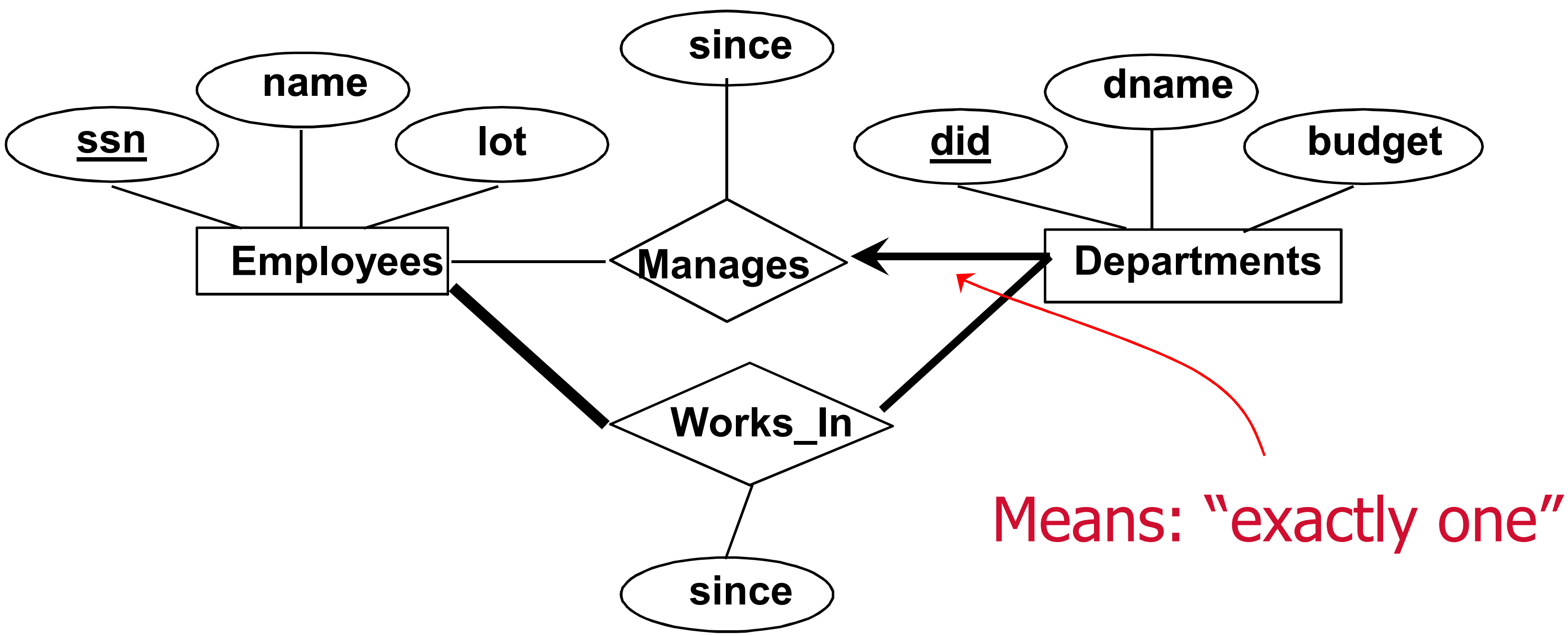
# E-R Diagram


Symbol
Meaning
ENTITY TYPE
WEAK ENTITY TYPE
RELATIONSHIP TYPE
ATTRIBUTE
KEY ATTRIBUTE
MULTIVALUED ATTRIBUTE
COMPOSITE ATTRIBUTE
DERIVED ATTRIBUTE
E1
1
R
N
E2
CARDINALITY RATIO 1:


Symbol
Meaning
Symbol
Meaning
Entity
Relationship
Weak Entity
Identifying Relationship
Attribute
Multi Valued Attribute

# Participation Constraints

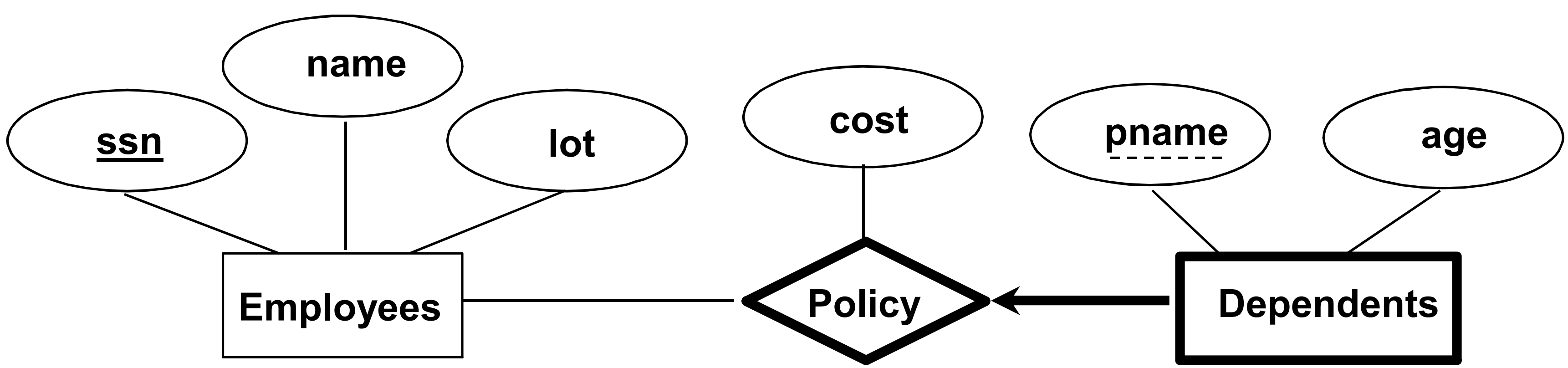
- Does every employee work in a department?
- If so, this is a *participation constraint*
  - the participation of Employees in Works_In is said to be *total* (vs. *partial*)
  - What if every department has an employee working in it?
- Basically means "at least one"



# Weak Entities

A *weak entity* can be identified uniquely only by considering the primary key of another (*owner*) entity.

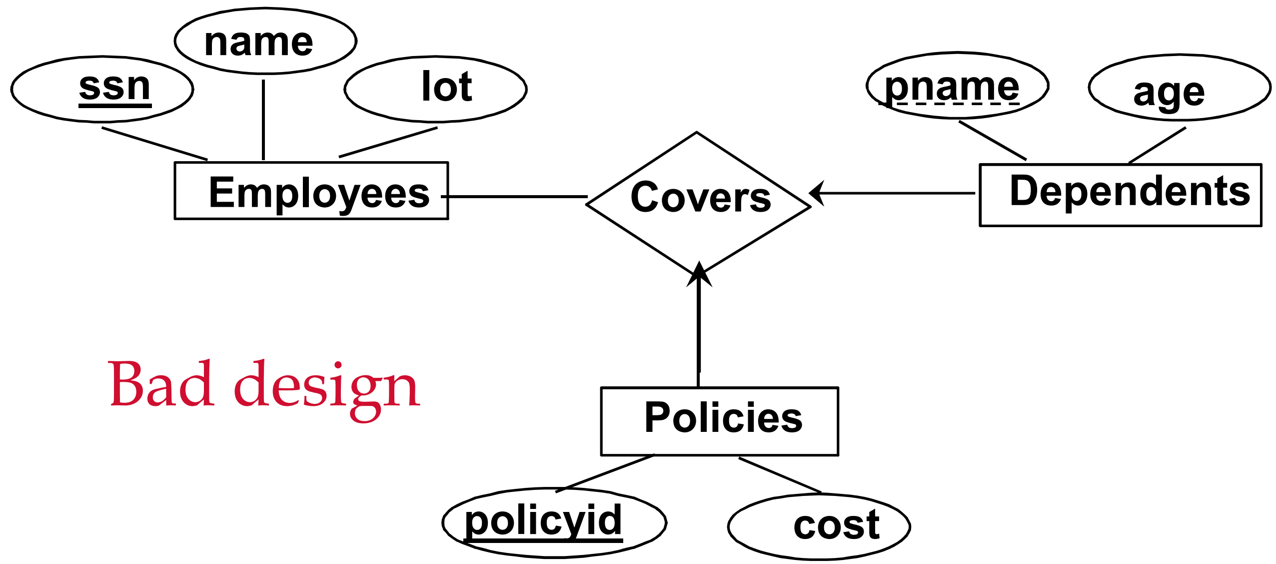
- Owner entity set and weak entity set must participate in a one-to-many relationship set (one owner, many weak entities).
- Weak entity set must have total participation in this *identifying* relationship set.




Weak entities have only a "partial key" (dashed underline)

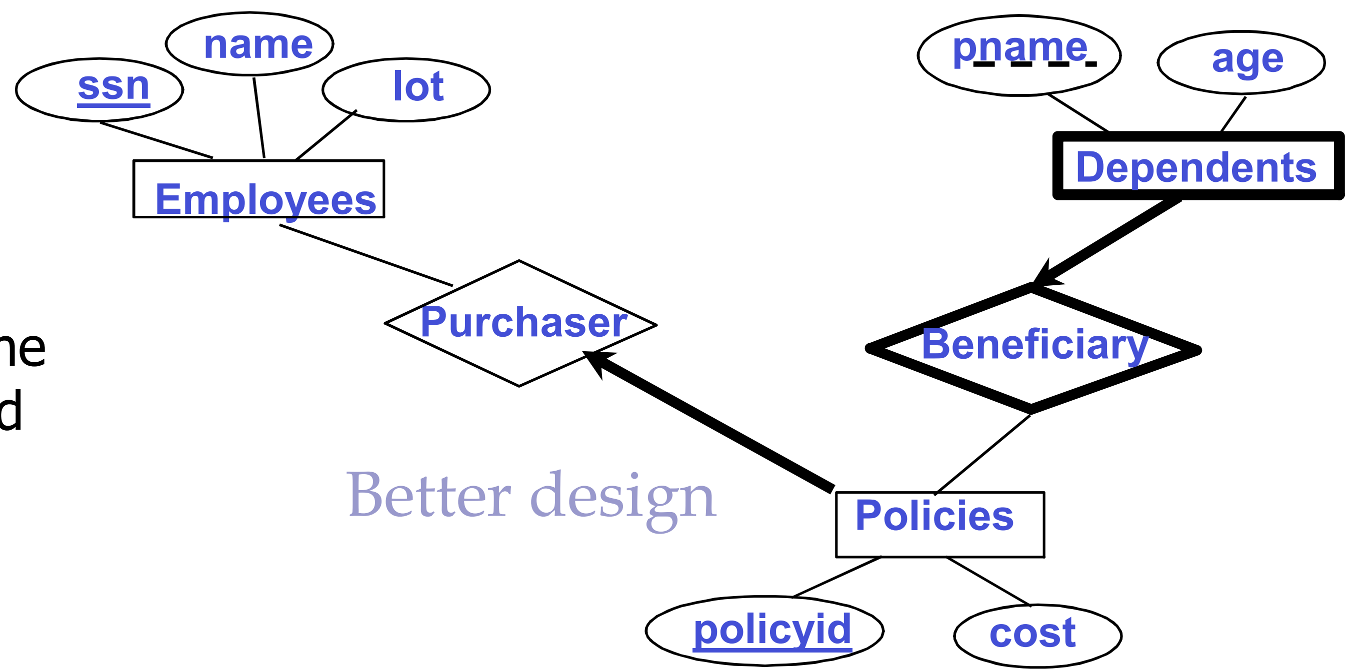
# Binary vs. Ternary Relationships

**If each policy is owned by just 1 employee:**




Key constraint on Policies would mean policy can only cover 1 dependent!

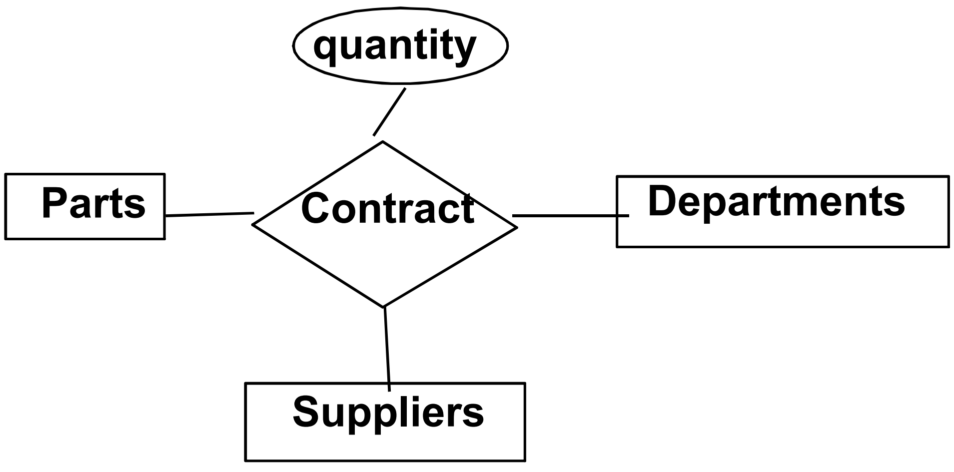
- Think through *all* the constraints in the 2nd diagram!



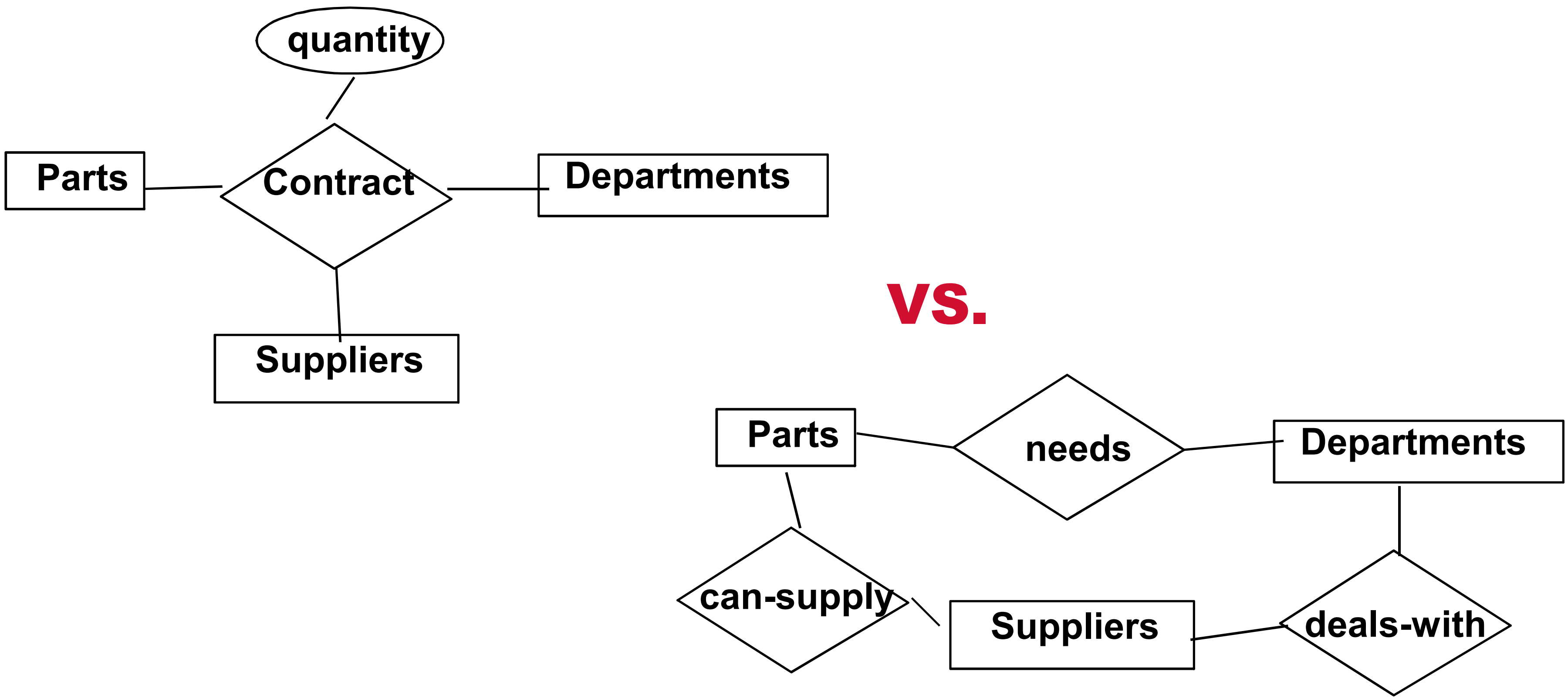
# Binary vs. Ternary Relationships

- Previous example illustrated a case when two binary relationships were better than one ternary.

- An example in the other direction: a ternary relation Contracts relates entity sets Parts, Departments and Suppliers, and has descriptive attribute *quantity*.
  - No combination of binary relationships is an adequate substitute.



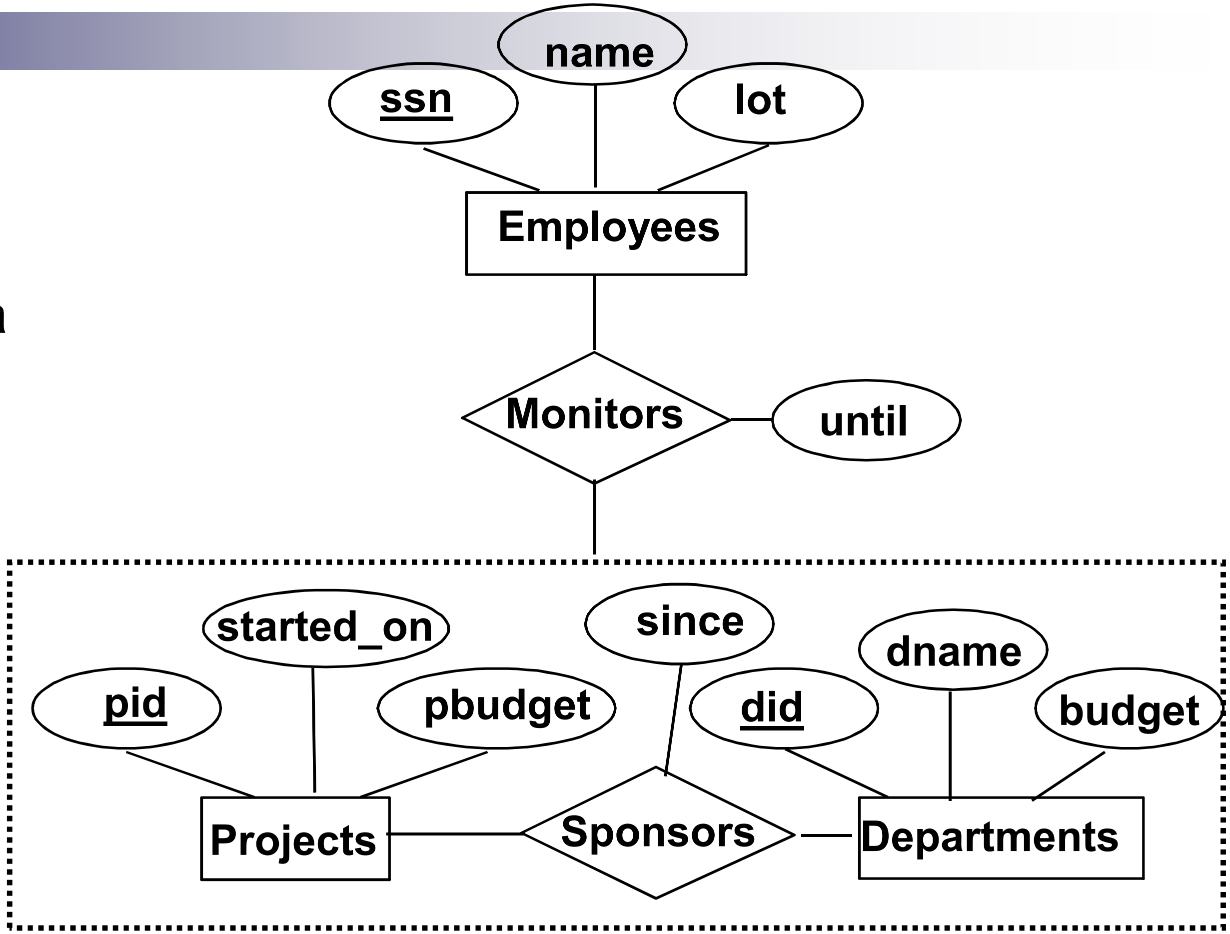
# Binary vs. Ternary Relationships




- 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*?

# Aggregation

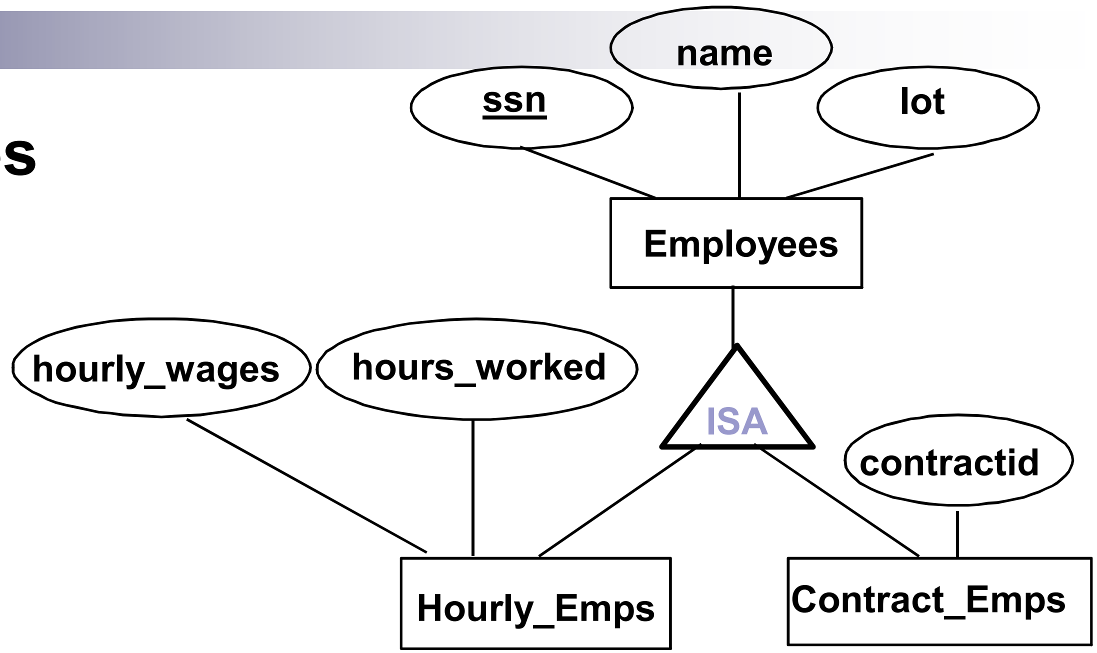
- Used to model a relationship involving a *relationship set*.
- Allows us to treat a relationship set as an entity set for purposes of participation in (other) relationships.




*Aggregation vs. ternary relationship*?

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

# ISA (`is a') Hierarchies




- *Overlap constraints*: Can Simon 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.
    - i.e. not appropriate for all entities in the superclass
  - To identify entities that participate in a particular relationship
    - i.e., not all superclass entities participate

# Review - Our Basic ER Model

- Entities and Entity Set (boxes)
- Relationships and Relationship sets (diamonds)
  - binary
  - n-ary
- Key constraints (1-1,1-M, M-M, arrows on 1 side)
- Participation constraints (bold for Total)
- Weak entities - require strong entity for key
- Aggregation - an alternative to n-ary relationships
- Isa hierarchies - abstraction and inheritance

# Conceptual Design Using the ER Model

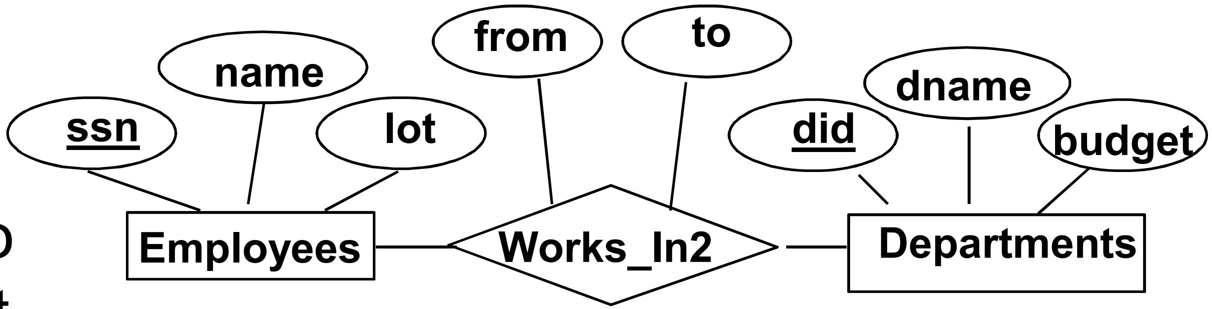
- ER modeling *can* get tricky!
- Design choices:
  - Should a concept be modeled as an entity or an attribute?
  - Should a concept be modeled as an entity or a relationship?
  - Identifying relationships: Binary or ternary? Aggregation?
- Note constraints of the ER Model:
  - A lot of data semantics can (and should) be captured.
  - But some constraints cannot be captured in ER diagrams.
    - We'll refine things in our logical (relational) design

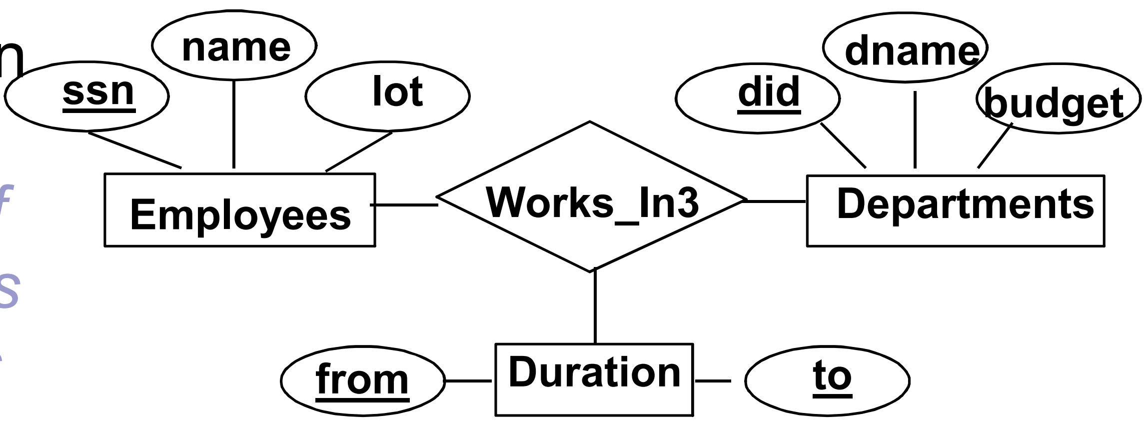
# Entity vs. Attribute

- Should *address* be an attribute of Employees or an entity (related to Employees)?
- Depends upon how we want to use 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, *address* must be modeled as an entity (since attribute values are atomic).

# Entity vs. Attribute (Cont.)

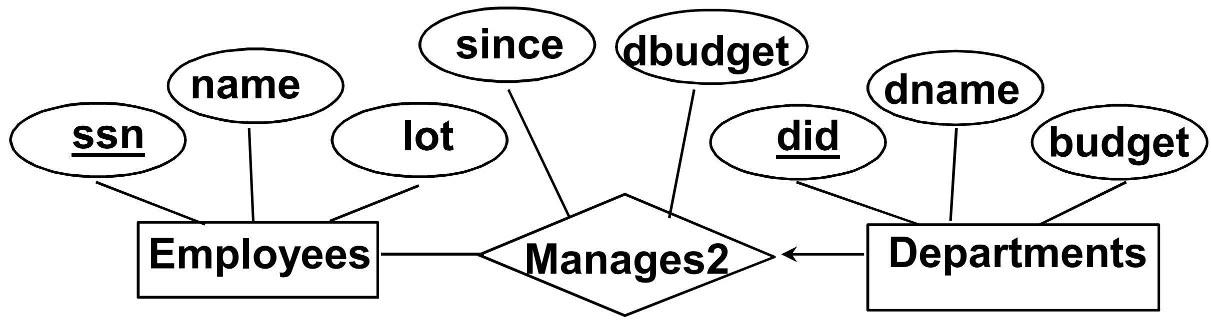
- Works_In2 does not allow an employee to work in a department for two or more periods.
- Similar to the problem of wanting to record several addresses for an employee: we want to record *several values of the descriptive attributes for each instance of this relationship.*

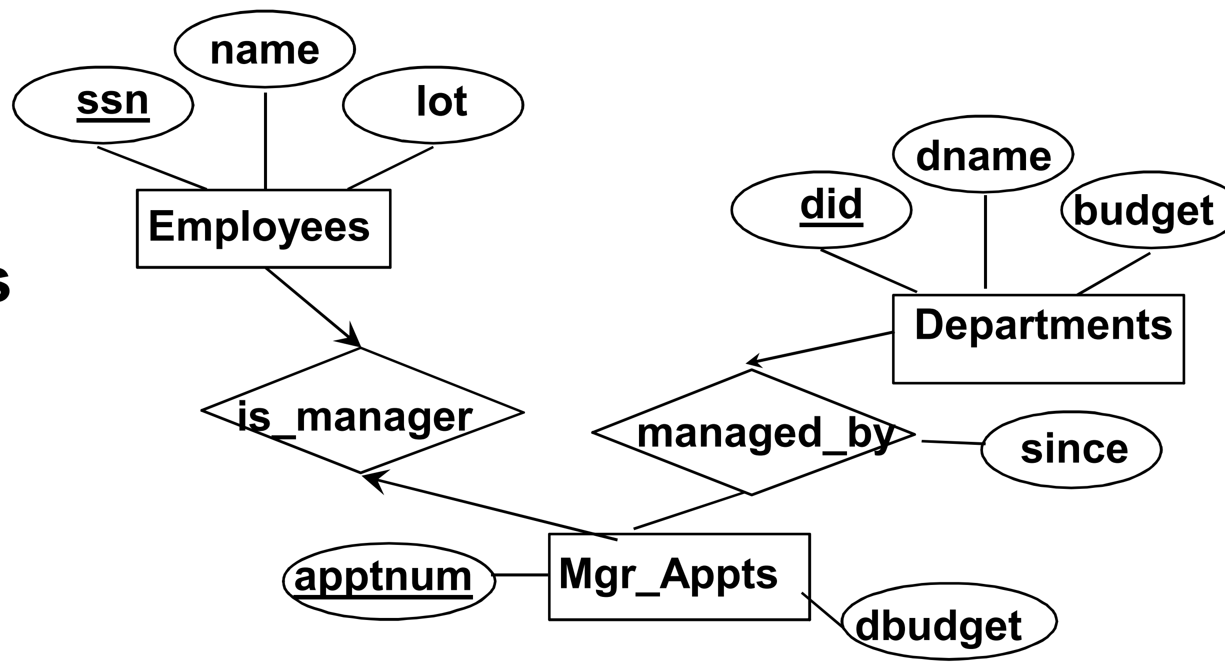


# Entity vs. Relationship

- OK as long as a manager gets a separate discretionary budget (*dbudget*) for each dept.
- What if manager's *dbudget* covers *all* managed depts?
- (can repeat value, but such redundancy is problematic)

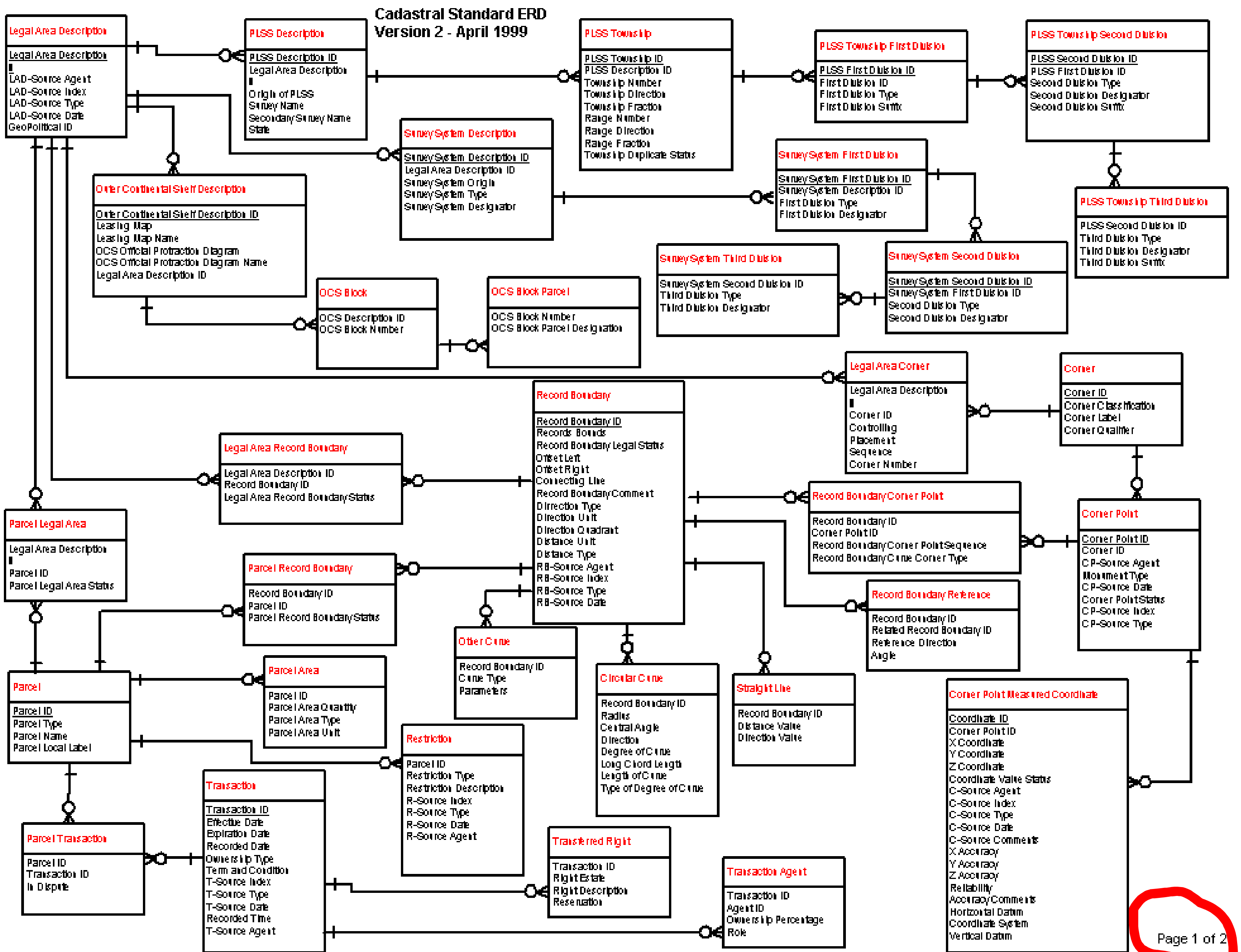


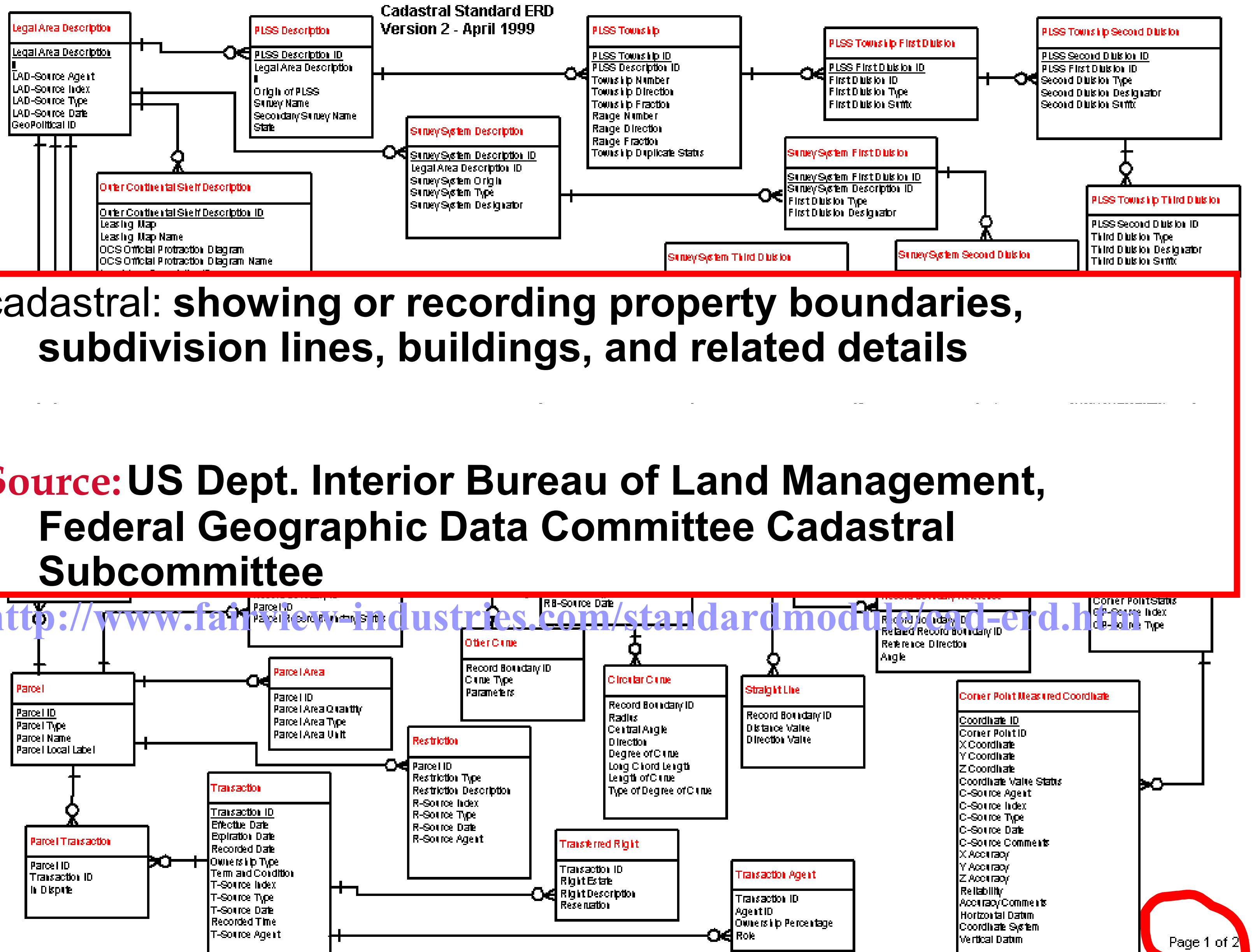
# These things get pretty hairy!

- Many E-R diagrams cover entire walls!
- A modest example:

# A Cadastral E-R Diagram



# A Cadastral E-R Diagram



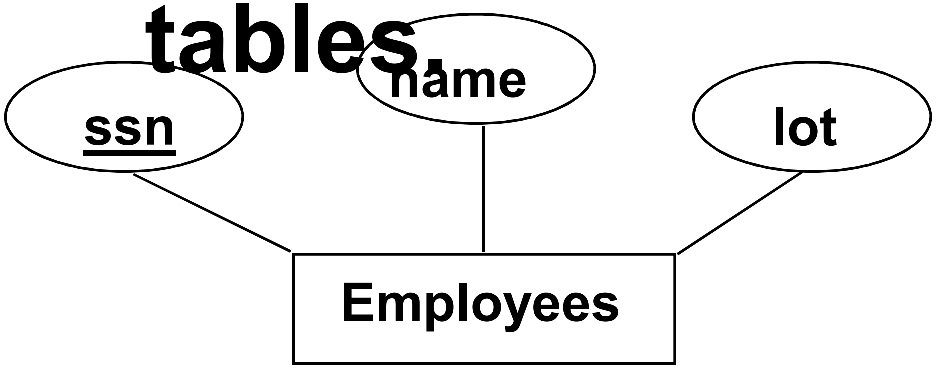

cadastral: **showing or recording property boundaries, subdivision lines, buildings, and related details**

Source: **US Dept. Interior Bureau of Land Management, Federal Geographic Data Committee Cadastral Subcommittee**

http://www.fairview-industries.com/standardmodule/cad-erd.htm

# Logical DB Design: ER to Relational

- **Entity sets to tables.**




| 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

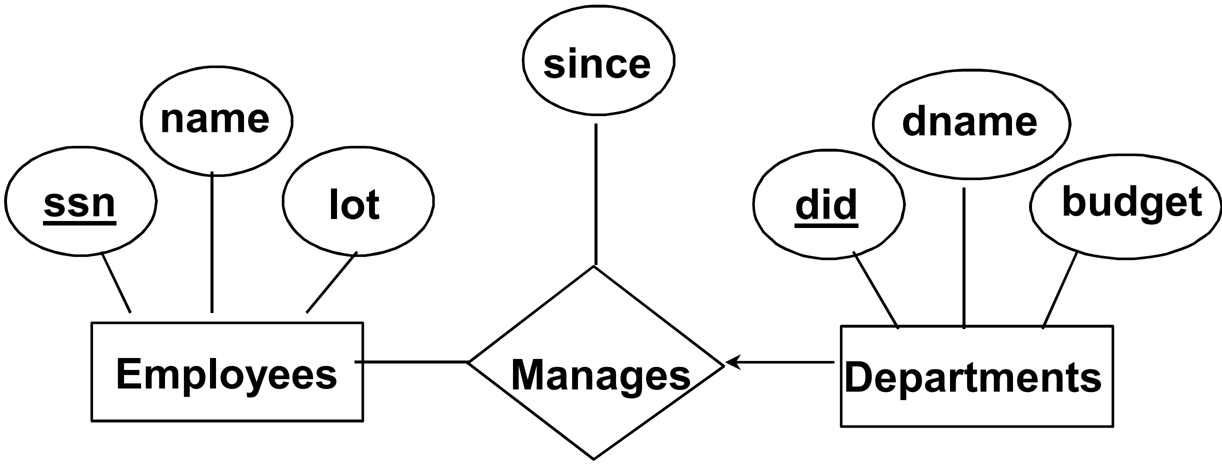
- **In translating a many-to-many relationship set to a relation, attributes of the relation must include:**
  1) Keys for each participating entity set (as foreign keys). This set of attributes forms a *superkey* for the relation.
  2) All descriptive attributes.

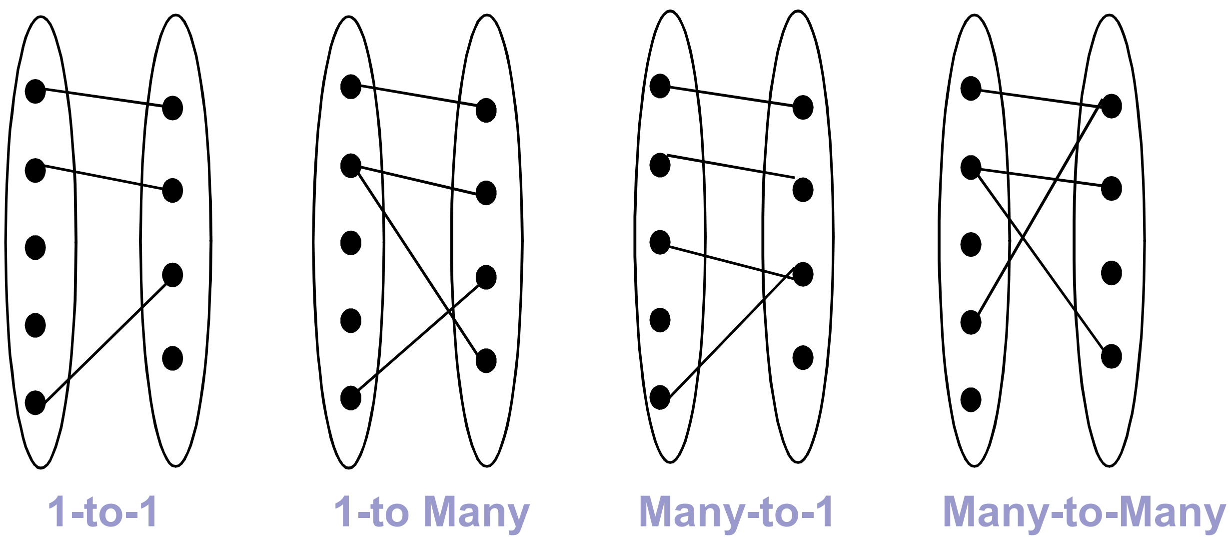
```
CREATE TABLE Works_In(
  ssn   CHAR(1),
  did   INTEGER,
  since   DATE,
 PRIMARY KEY (ssn, did),
 FOREIGN KEY (ssn)
    REFERENCES Employees,
 FOREIGN KEY (did)
  REFERENCES Departments)
```

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

# Review: Key Constraints

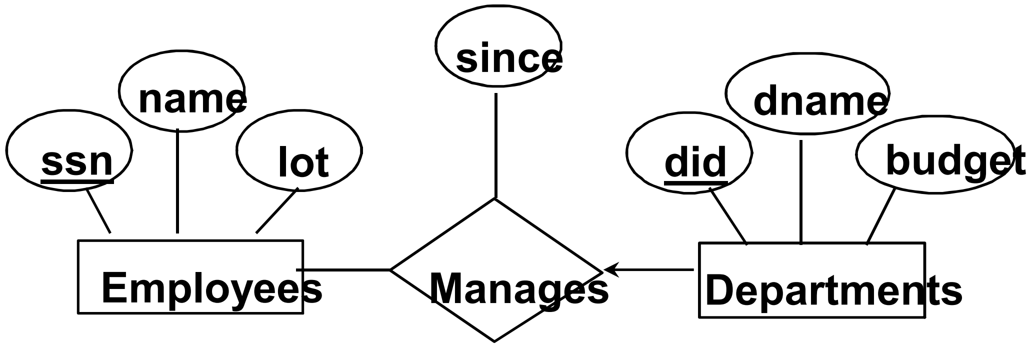
- Each dept has at most one manager, according to the *key constraint* on Manages.




*Translation to relational model?*

# Translating ER with Key Constraints




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

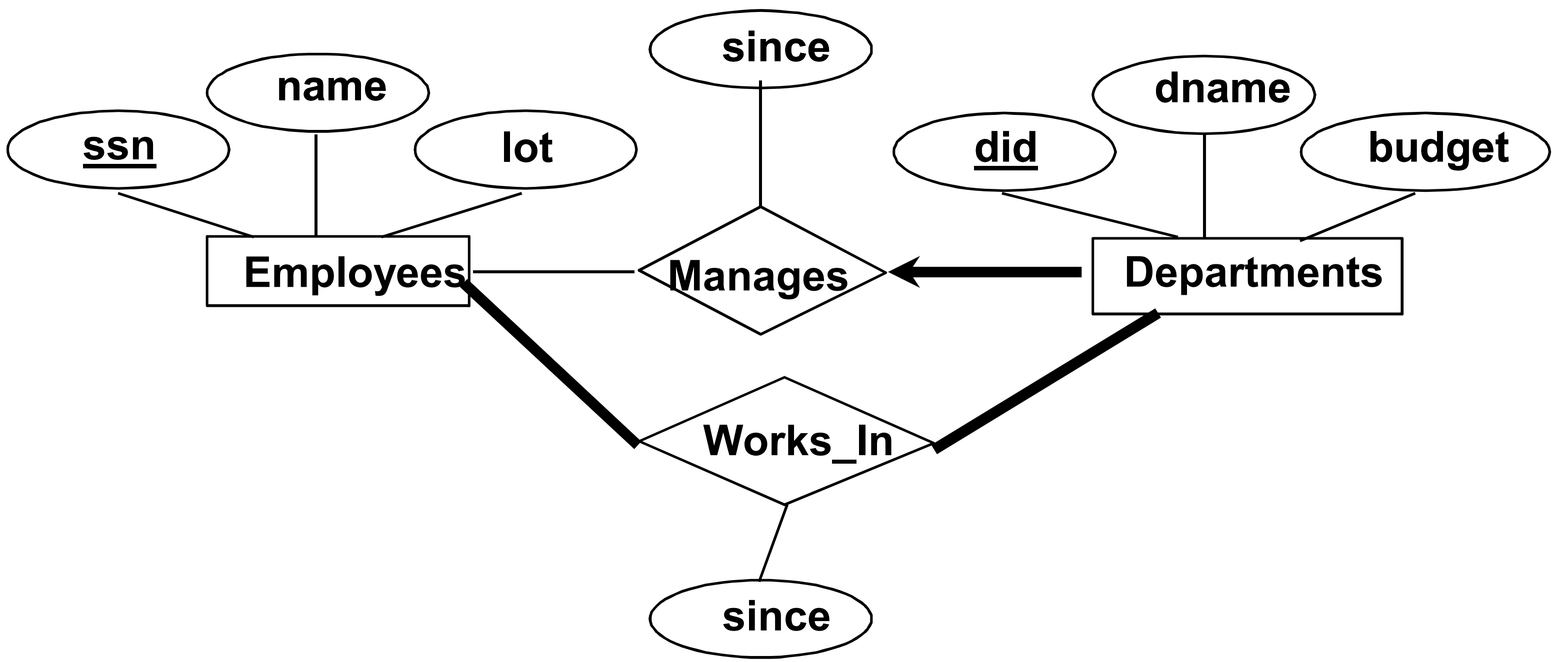
```
CREATE TABLE  Manages(
 ssn  CHAR(11),
 did  INTEGER,
 since  DATE,
 PRIMARY KEY  (did),
 FOREIGN KEY (ssn)
REFERENCES Employees,
   FOREIGN KEY (did)
REFERENCES Departments)
```

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)
```

# 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!)



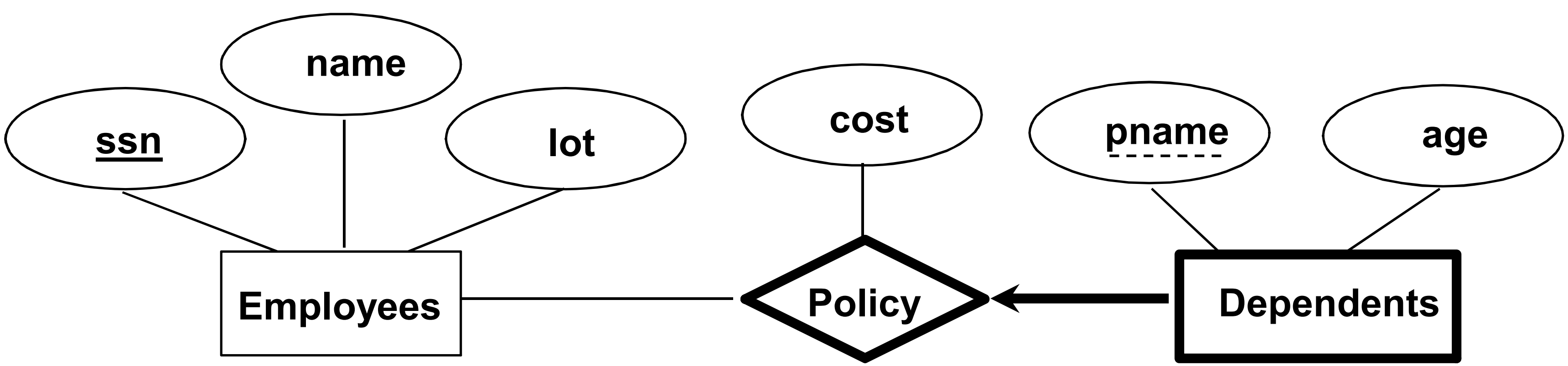
# Participation Constraints in SQL

- 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)
```

# 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.

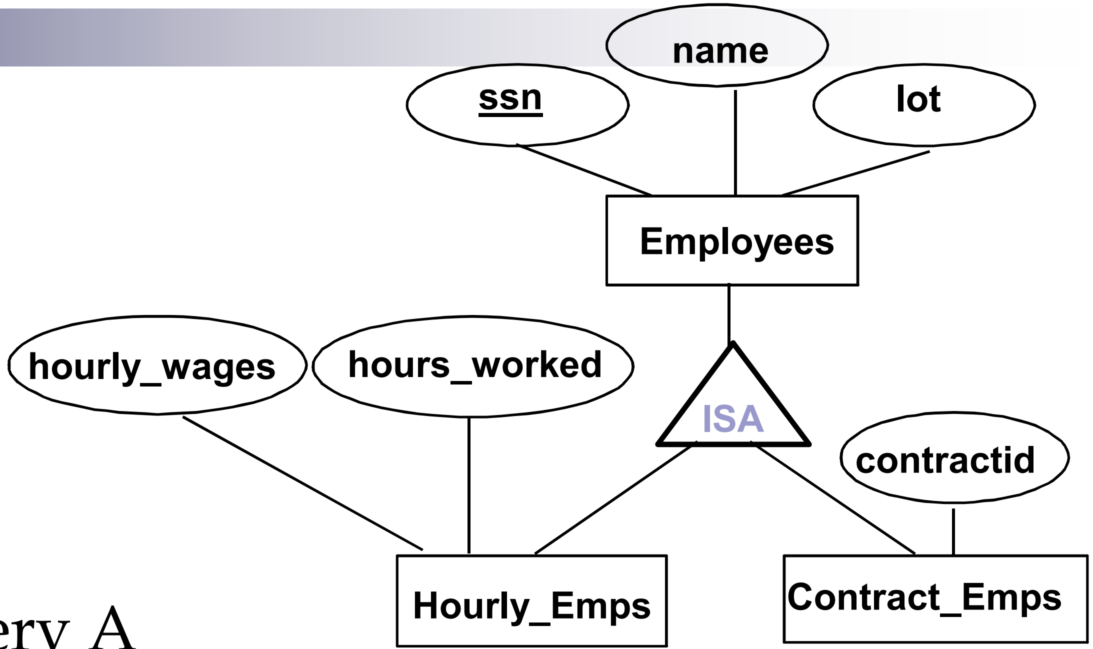


## Translating Weak Entity Sets

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

## Review: 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*, <u>*ssn*</u>); 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*: <u>*ssn*</u>, *name, lot, hourly_wages, hours_worked.*
  - Each employee must be in one of these two subclasses.

# Now you try it

University database:

- Courses, Students, Teachers
- Courses have ids, titles, credits, …
- Courses have multiple sections that have time/rm and exactly one teacher
- Must track students’ course schedules and transcripts including grades, semester taken, etc.
- Must track which classes a professor has taught
- Database should work over multiple semesters

# Other SQL DDL Facilities

- Integrity Constraints (ICs) - Review
- An IC describes conditions that every *legal instance* of a relation must satisfy.
  - Inserts/deletes/updates that violate IC's are disallowed.
  - Can be used to ensure application semantics (e.g., *sid* is a key), or prevent inconsistencies (e.g., *sname* has to be a string, *age* must be < 200)
- *Types of IC's*: Domain constraints, primary key constraints, foreign key constraints, general constraints.
  - *Domain constraints*: Field values must be of right type. Always enforced.
  - *Primary key and foreign key constraints*: you know them.

## General Constraints

- Useful when more general ICs than keys are involved.
- Can use queries to express constraint.
- Checked on insert or update.
- Constraints can be named.

```
CREATE TABLE Sailors
    ( sid INTEGER,
    sname CHAR(10),
    rating INTEGER,
    age REAL,
    PRIMARY KEY (sid),
    CHECK ( rating >= 1
        AND rating <= 10 ))
```

```
CREATE TABLE Reserves
    ( sname CHAR(10),
    bid INTEGER,
    day DATE,
    PRIMARY KEY (bid,day),
    CONSTRAINT noInterlakeRes
    CHECK (`Interlake' <>
        ( SELECT B.bname
        FROM Boats B
        WHERE B.bid=bid)))
```

# Constraints Over Multiple Relations

- Awkward and wrong!
- Only checks sailors!
- Only required to hold if the associated table is non-empty.
- ASSERTION is the right solution; not associated with either table.
- Unfortunately, not supported in many DBMS.
- *Triggers* are another solution.

```
CREATE TABLE Sailors
( sid INTEGER,
sname CHAR(10),
rating INTEGER,
age REAL,
PRIMARY KEY (sid),
CHECK
( (SELECT COUNT (S.sid) FROM Sailors S)
+ (SELECT COUNT (B.bid) FROM
        Boats B) < 100 )
```

*Number of boats plus number of sailors is < 100*

```
CREATE ASSERTION smallClub
CHECK
( (SELECT COUNT (S.sid) FROM Sailors S)
+ (SELECT COUNT (B.bid)
 FROM Boats B) < 100 )
```

# Or, Use a Trigger

- Trigger: procedure that starts automatically if specified changes occur to the DBMS
- Three parts:
  - Event (activates the trigger)
  - Condition (tests whether the triggers should run)
  - Action (what happens if the trigger runs)
- Triggers (in some form) are supported by most DBMSs; Assertions are not.
- Support for triggers is defined in the SQL:1999 standard.

# Triggers

```
CREATE TRIGGER trigger_name
ON TABLE
{FOR {[INSERT][,][UPDATE][,][DELETE]}
[WITH APPEND]
AS
sql-statements
```

- Cannot be called directly – initiated by events on the database.
- Can be *synchronous* or *asynchronous* with respect to the transaction that causes it to be fired.

## Triggers: Example

```
CREATE TRIGGER member_delete
ON member FOR DELETE
AS
IF (Select COUNT (*) FROM loan INNER JOIN deleted
        ON loan.member_no = deleted.member_no) > 0
  BEGIN
     PRINT 'ERROR - member has books on loan.'
     ROLLBACK TRANSACTION
  END
ELSE
DELETE reservation WHERE  reservation.member_no
    = deleted.member_no
```

# Summary: Triggers, Assertions, Constraints

- Very vendor-specific (although standard has been developed).
- Triggers vs. Contraints and Assertions:
  - Triggers are “operational”, others are declarative.
- Triggers can make the system hard to understand if not used with caution.
  - ordering of multiple triggers
  - recursive/chain triggers
- Triggers can be hard to optimize.
- But, triggers are also very powerful.
- Use to create high-performance, “*active”* databases.

# 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.
  - 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*, and *aggregation*.**

# Summary of ER (Cont.)

- **Several kinds of integrity constraints:**
  - *key constraints*
  - *participation constraints*
  - *overlap/covering* for ISA hierarchies.
- **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*. 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, aggregation.
- **Ensuring good database design: resulting relational schema should be analyzed and refined further.**
  - Functional Dependency information and normalization techniques are especially useful.