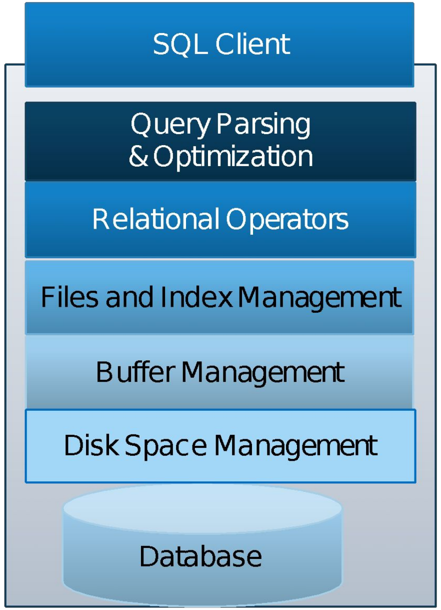
Logical Database Design: Entity-Relation Models

R&G 2

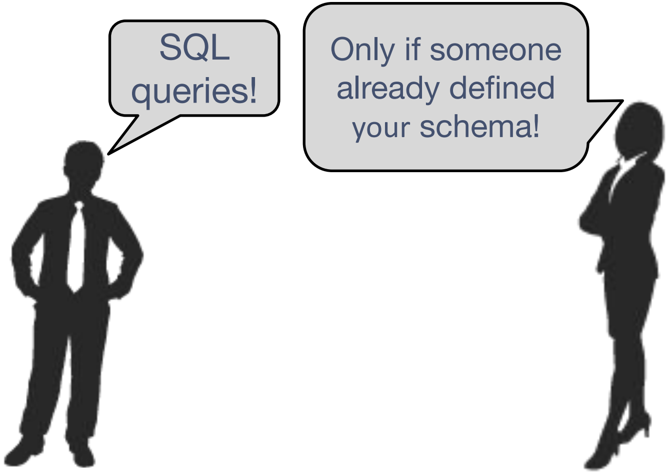
Architecture of a DBMS

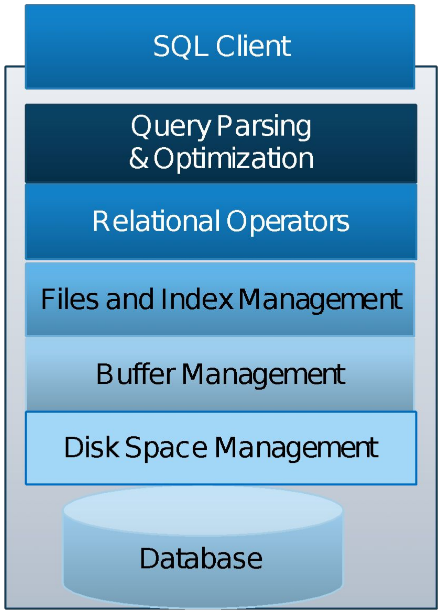
- Gives us a good sense of how to build a DBMS
- How about using one?



Architecture of a DBMS, Pt 2

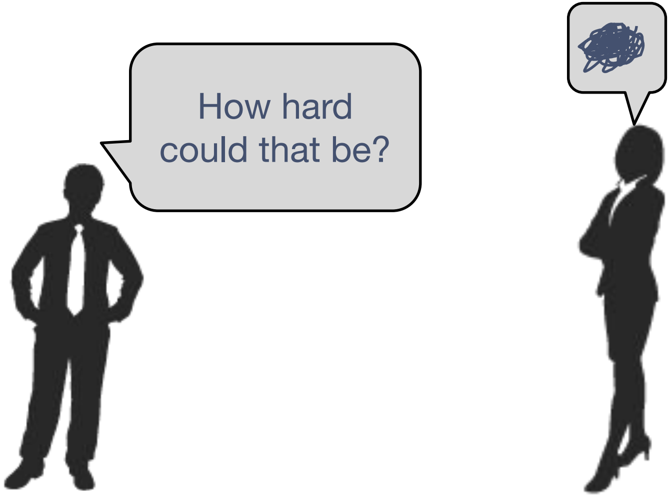
- Gives us a good sense of how to build a DBMS
- How about using one?

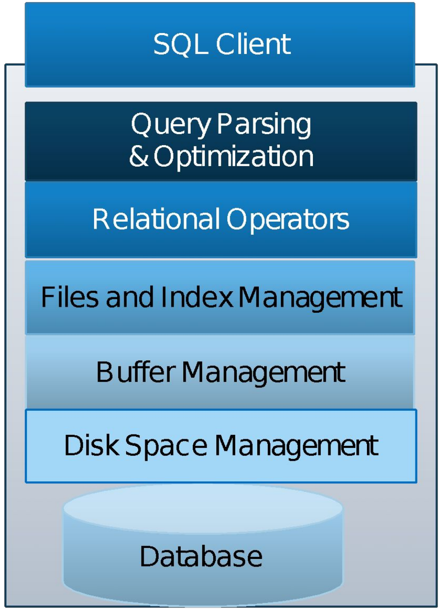




Architecture of a DBMS, Pt 3

- Gives us a good sense of how to build a DBMS
- How about using one?

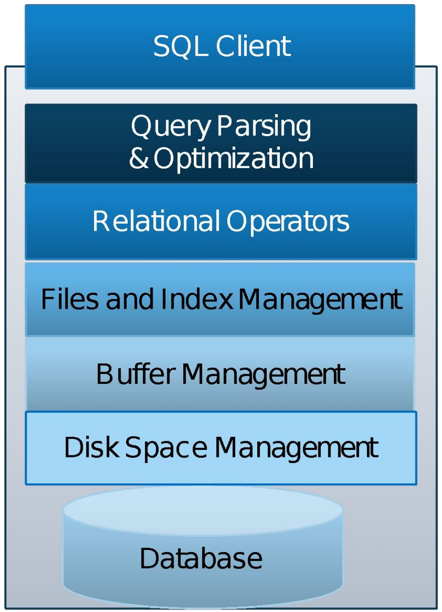




Design of a Database

- Gives us a good sense of how to build a DBMS
- How about using one?

- Today let's talk about how to design a database
  - Not a database system



Steps in Database Design

- **Requirements Analysis**
  - user needs; what must database do?
- **Conceptual Design**
  - *high level description (often done w/ER model)*
  - Object-Relational Mappings (ORMs: Hibernate, Rails, Django, etc) encourage you to program here
- **Logical Design**
  - translate ER into DBMS data model
  - ORMs often require you to help here too
- **Schema Refinement**
  - consistency, normalization
- **Physical Design** - indexes, disk layout
- **Security Design** - who accesses what, and how

You are here

Describing Data: Data Models

- **Data model :** collection of concepts for describing data.
- **Schema:** description of a particular collection of data, using a given data model.
- **Relational model of data**
  - Main concept: relation (table), rows and columns
  - Every relation has a schema
    - describes the columns
    - column names and domains

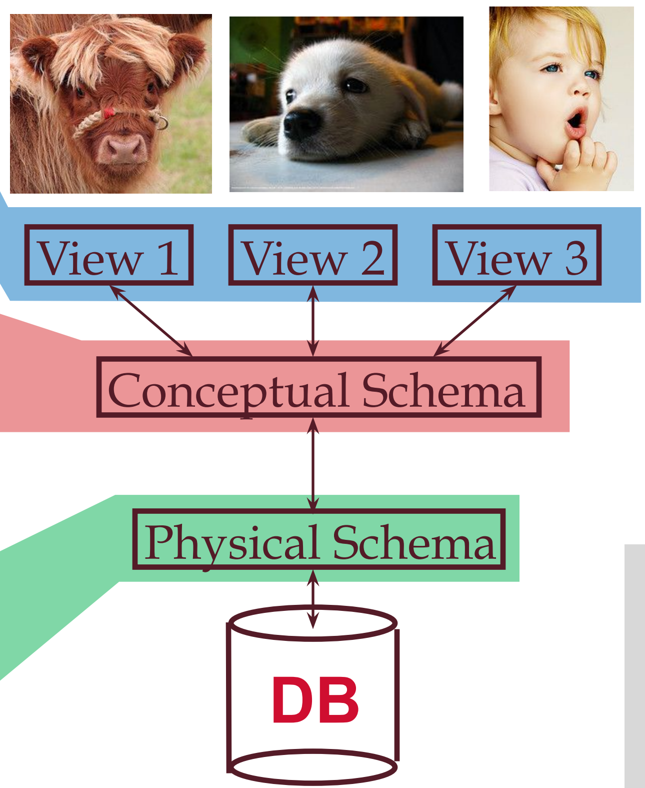
Levels of Abstraction

Users

Views describe how users see the data.

Conceptual schema defines logical structure

Physical schema describes the files and indexes used.



Example: University Database

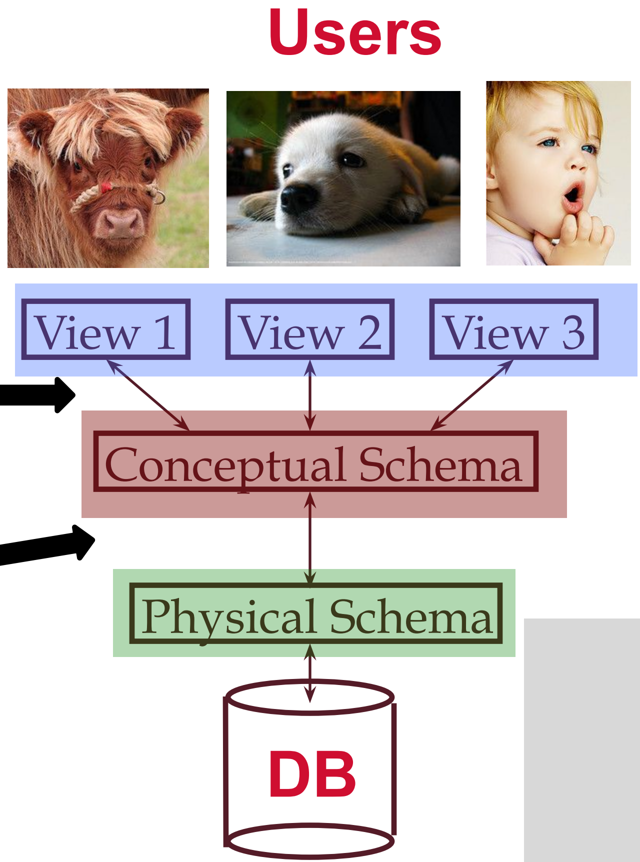
- **Conceptual schema:**
  - `Students(sid text, name text, login text, age integer, gpa float)`
  - `Courses(cid text, cname text, credits integer)`
  - `Enrolled(sid text, cid text, grade text)`
- **Physical schema:**
  - Relations stored as unordered files.
  - Index on first column of Students.
- **External Schema** (View):
  - Course_info(cid text, enrollment integer)

Data Independence

- Insulate apps from structure of data

- **Logical data independence:**
  - Maintain views when logical structure changes
- **Physical data independence:**
  - Maintain logical structure when physical structure changes

Levels of Abstraction, cont



Logical data independence

Physical data independence

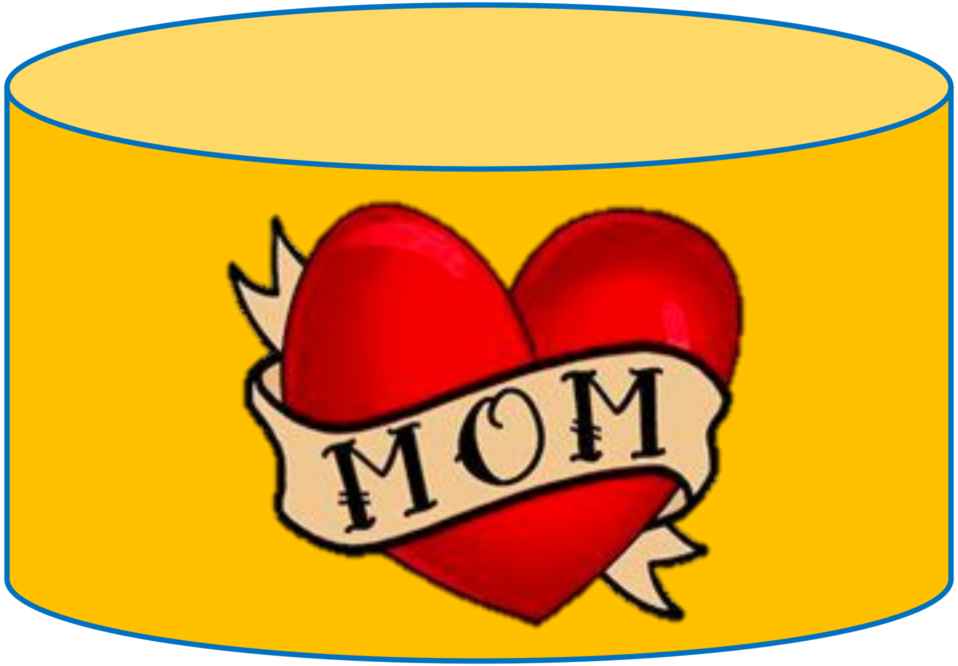
Data Independence, cont

- Insulate apps from structure of data

- **Logical data independence:**
    - Maintain views when logical structure changes
- **Physical data independence:**
    - Maintain logical structure when physical structure changes

- Q: Why particularly important for DBMS?
    - Because databases and their associated applications persist

An Anecdote



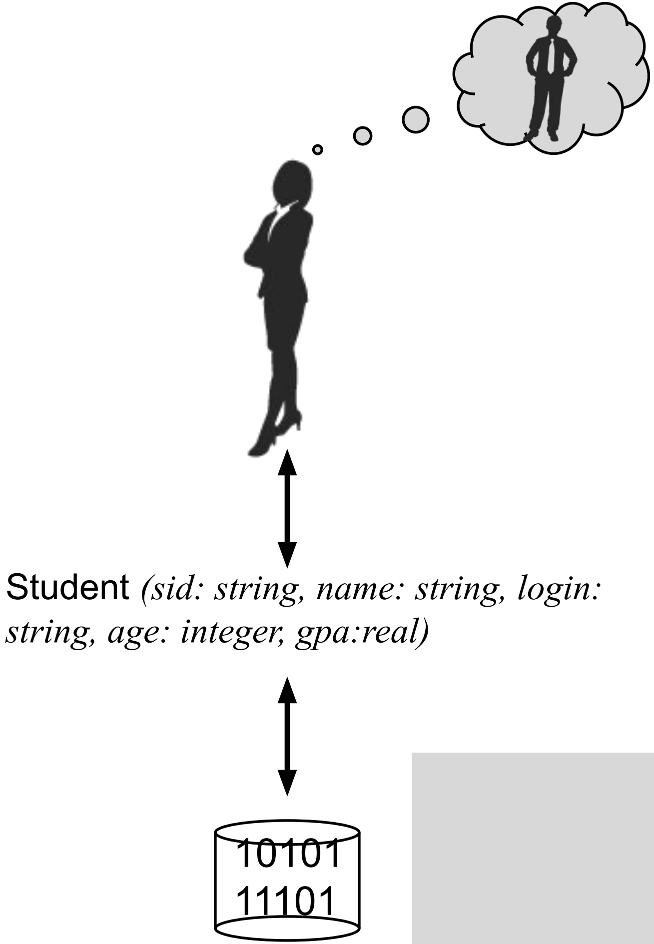
Hellerstein’s Inequality

$$\frac{dapp}{dt} << \frac{denv}{dt}$$

Data independence is most important when
the rate of change of your environment
exceeds the rate of change of your applications.

Data Models

- Connect concepts to bits!
- Many models exist
- We will ground ourselves in the Relational model
  - clean and common
  - generalization of key/value
- Entity-Relationship model also handy for design
  - Translates down to Relational



Entity-Relationship Model

- Relational model is a great formalism
  - But a bit detailed for design time
  - Too fussy for brainstorming
  - Hard to communicate to “customers”

- Entity-Relationship model: a graph-based model
  - can be viewed as a graph, or a veneer over relations
    - “feels” more flexible, less structured
  - corresponds well to “Object-Relational Mapping”
    - (ORM) SW packages
    - Ruby-on-Rails, Django, Hibernate, Sequelize, etc.

Steps in Database Design, again

- Requirements Analysis
  - user needs; what must database do?
- **Conceptual Design**
  - ***high level description (often done w/ER model)***
  - **ORM encourages you to program here**
- Logical Design
  - translate ER into DBMS data model
  - ORMs often require you to help here too
- Schema Refinement
  - consistency, normalization
- Physical Design - indexes, disk layout
- Security Design - who accesses what, and how

You are here

Conceptual Design

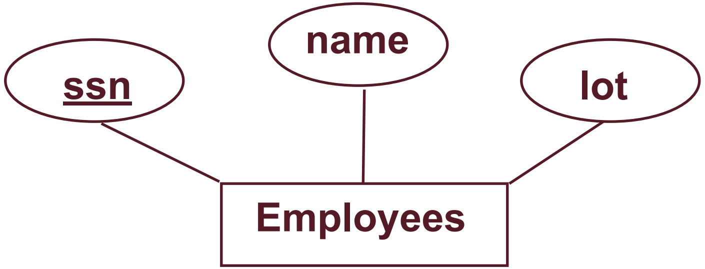
- What are the entities and relationships?
  - And what info about E’s & R’s should be in DB?
- What integrity constraints (“business rules”) hold?

- ER diagram is the “schema”
- Can map an ER diagram into a relational schema.

- Conceptual design is where the data engineering begins
  - If you’re familiar with the jargon, these are the “models” of the MVC pattern in ORMs

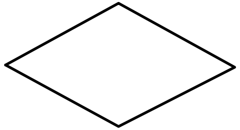
ER Model Basics: Entities

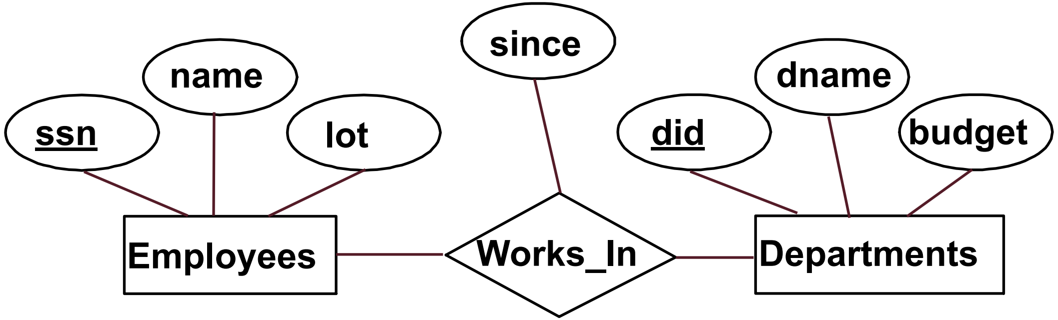
- **<u>Entity</u>**:
  - A real-world object described by a set of attribute values.



- **<u>Entity Set:</u>** A collection of similar entities.
  - E.g., all employees.
  - All entities in an entity set have the same attributes.
  - Each entity set has a key (underlined)
  - Each attribute has a domain

ER Model Basics: Relationships





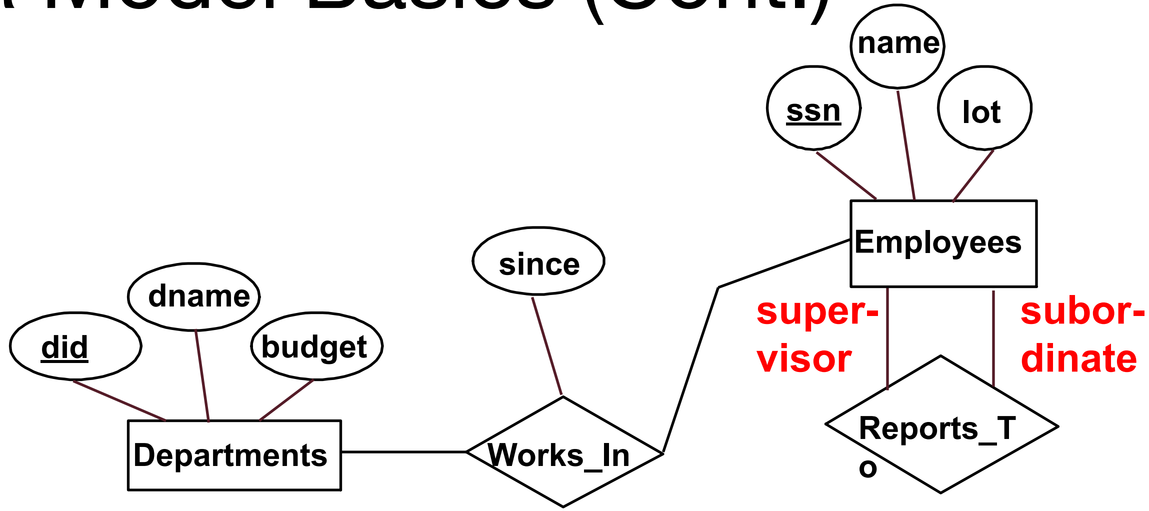
<u>Relationship:</u> Association among two or more entities.

- E.g., Attishoo works in Pharmacy department.
- Relationships can have their own attributes.

<u>Relationship Set:</u> Collection of similar relationships.

- An n-ary relationship set R relates n entity sets E1 ... En ; each relationship in R involves entities e1 $\in$ E1, ..., en $\in$ En

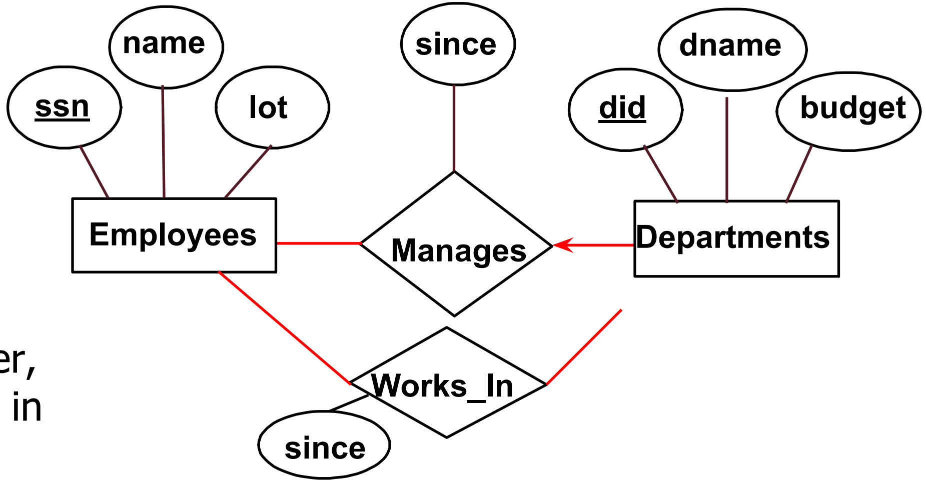
ER Model Basics (Cont.)

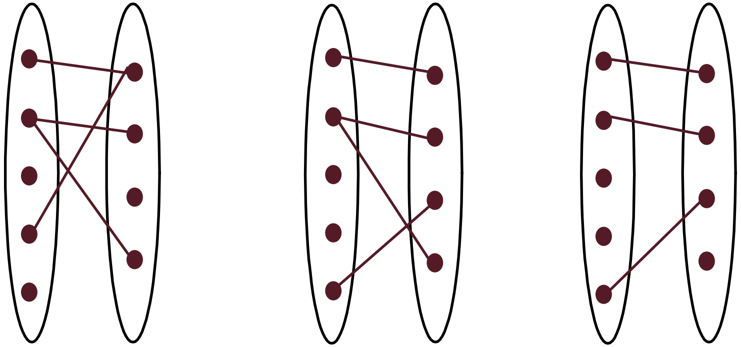


Same entity set can participate in different relationship sets, or in different “roles” in the same relationship set.

Key Constraints

- An employee can work in **many** departments; a dept can have **many** employees.
- In contrast, each dept has **at most one** manager, according to the *key constraint* on **Department** in the **Manages** relationship set.
- A **key constraint** gives a 1-to-many relationship.





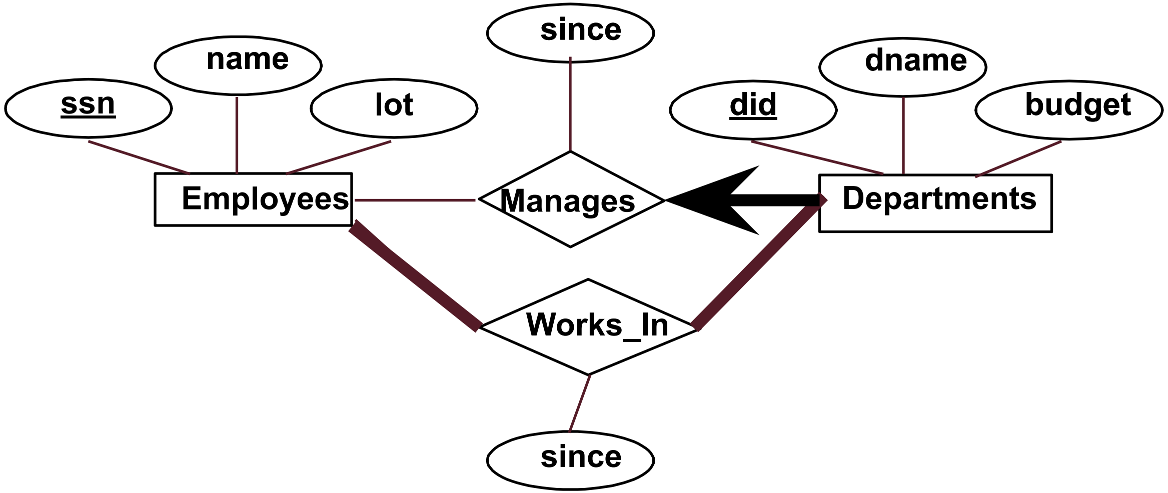
Many-to-Many

1-to-Many | Many-to-1

1-to-1

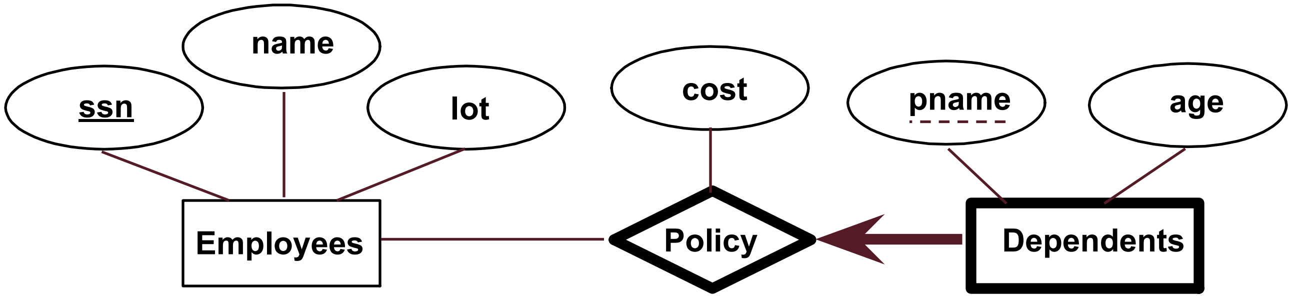
Participation Constraints

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



Weak Entities

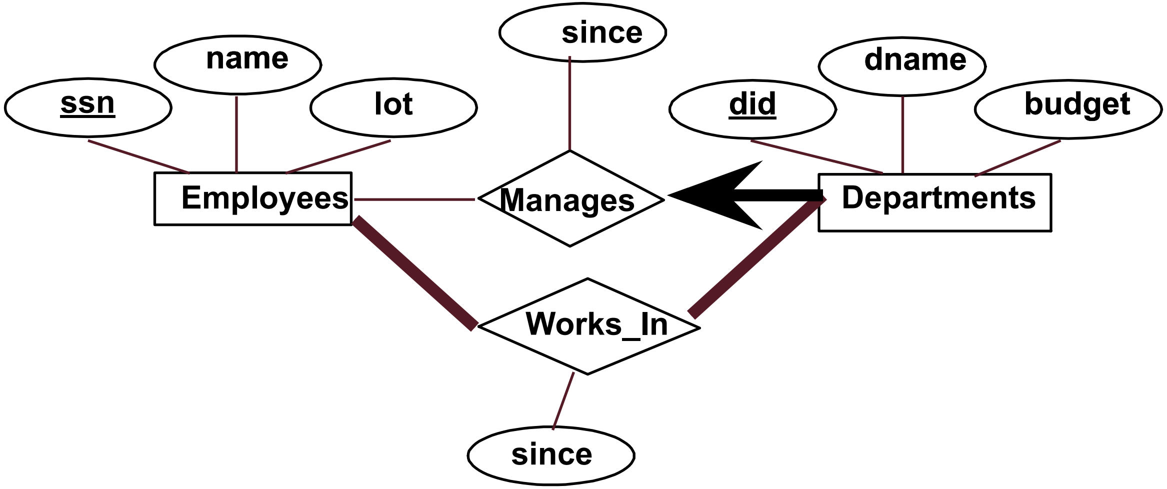
- A **weak entity** can be identified uniquely only by considering the primary key of another (owner) entity.
  - Owner entity set and weak entity set must participate in a one-to-many relationship set (one owner, many weak entities).
  - Weak entity set must have total participation in this identifying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

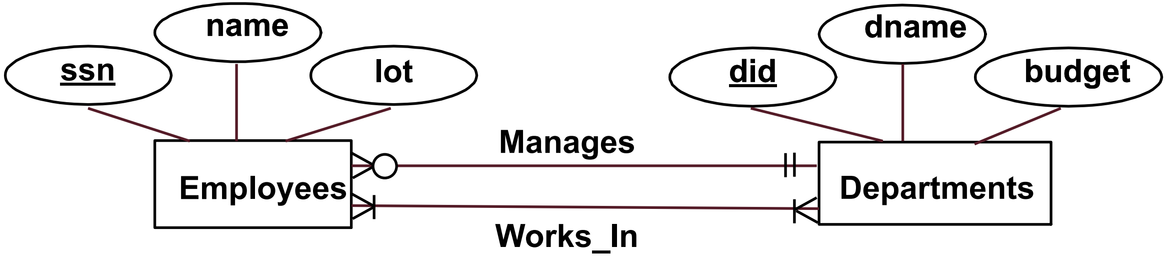


- Weak entities have only a “partial key” (dashed underline)

FYI: Crow's Foot Notation

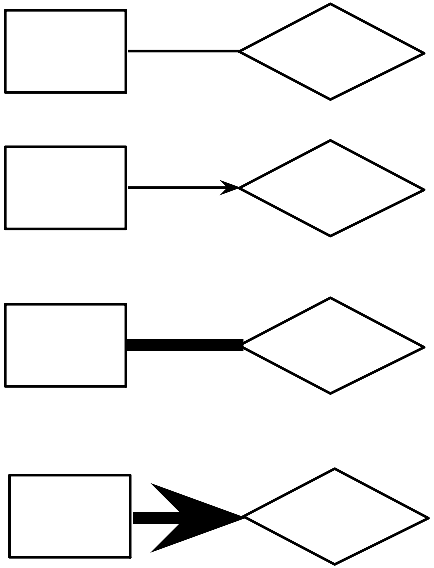
- ○ : 0 or more
- | : 1 or more
- |○ : 1 or 0
- ‖ : exactly one
- < : many

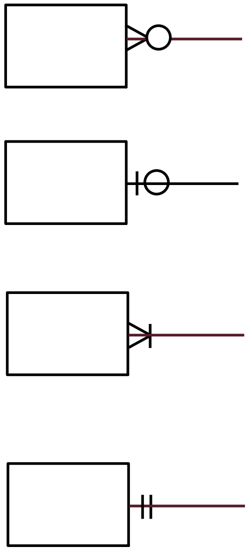




No relationship attributes

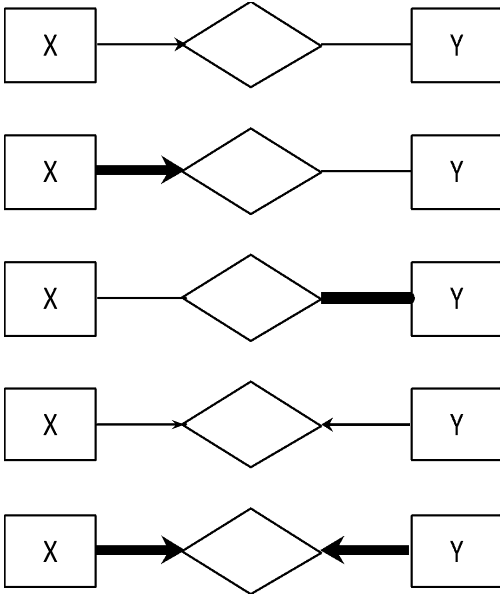
Translating constraints across notations





Translation to Math Terminology on Relations

- Relation R(X, Y) is a (*partial) function*
- Relation R(X, Y) is a *total function*
- Relation R(X, Y) is *surjective (onto)*
- Relation R(X, Y) is *injective (1-1)*
- Relation R(X, Y) is *a bijection*

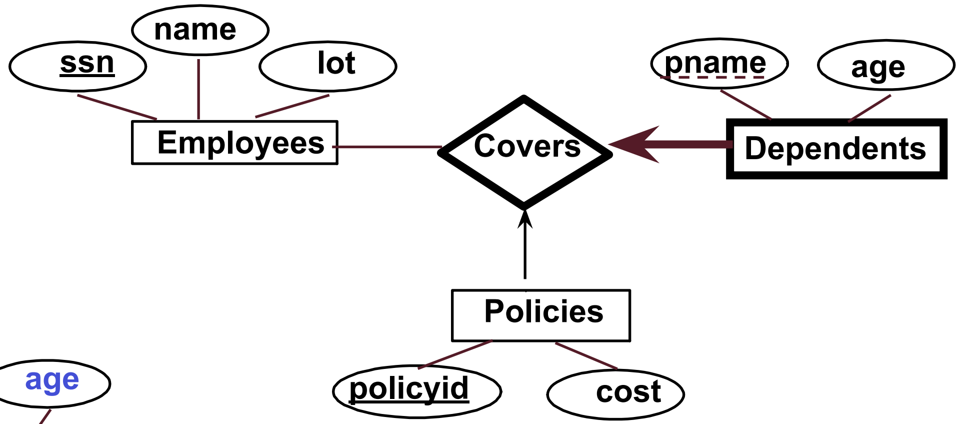


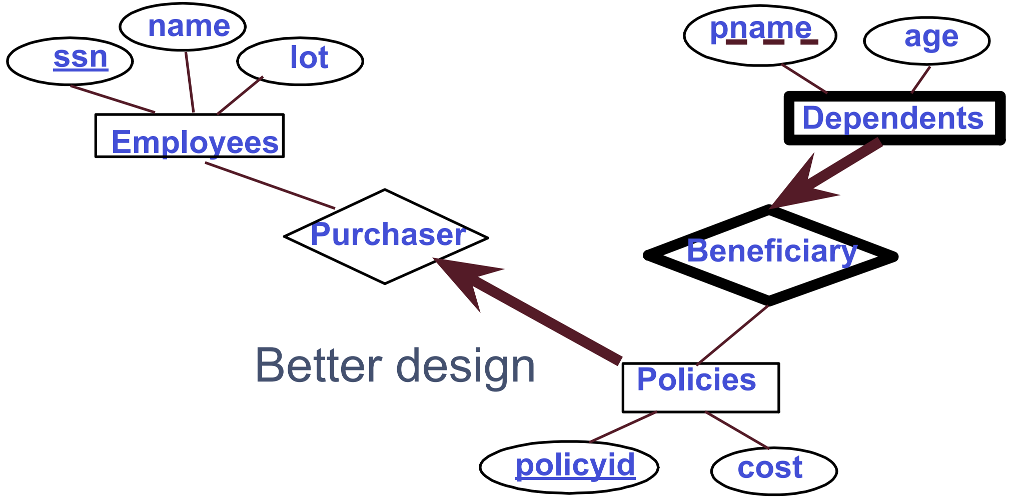
Binary vs. Ternary Relationships

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!





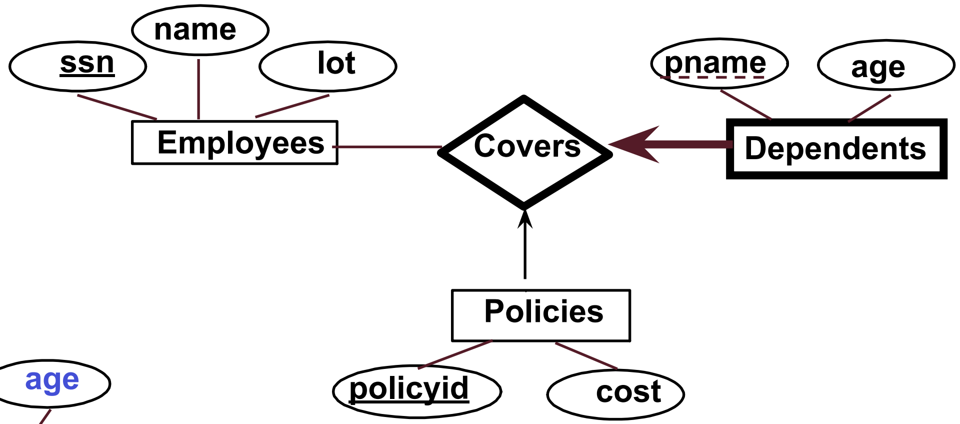
Better design

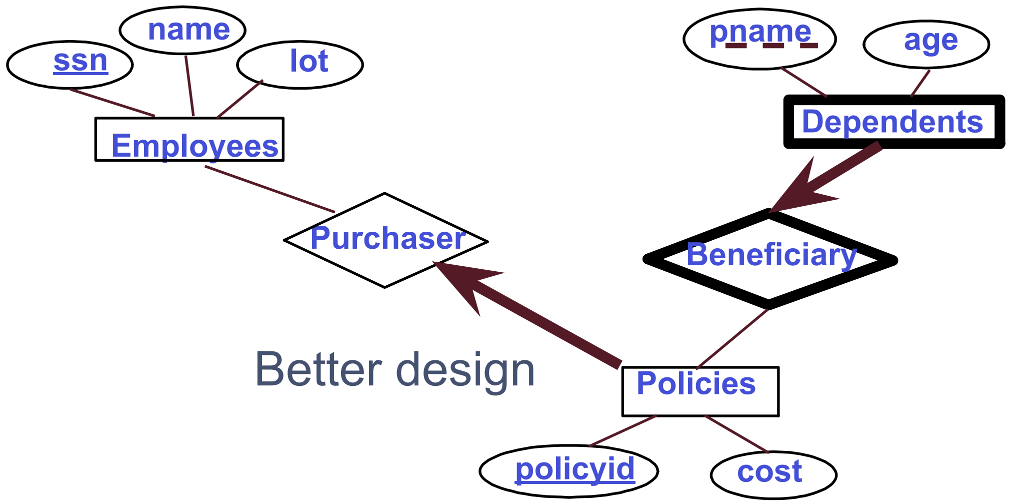
Binary vs. Ternary Relationships, cont

If each policy is owned by just 1 employee:

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

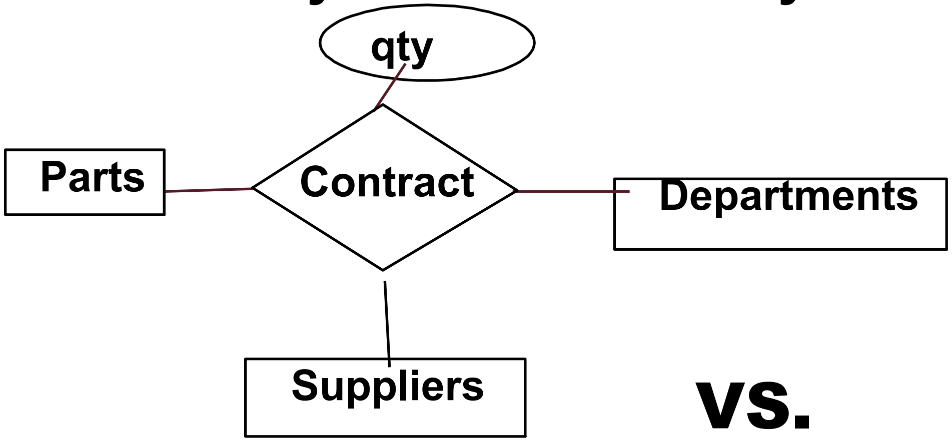
Think through ***all*** the constraints in the 2nd diagram!





Better design

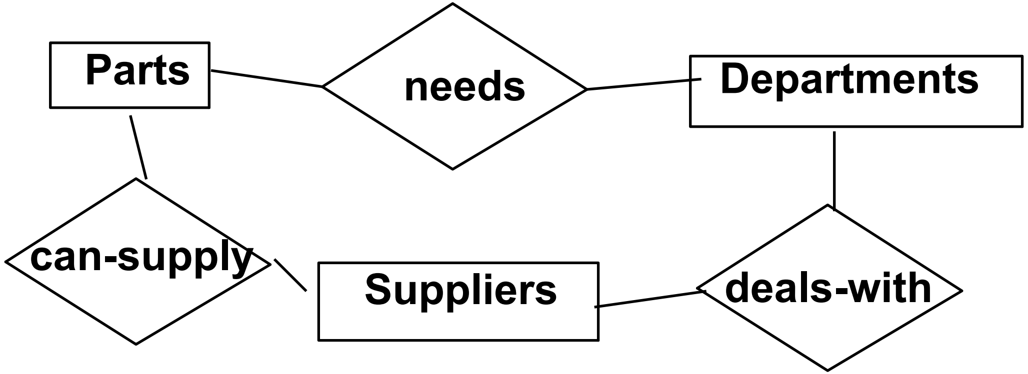
Binary and Ternary Relationship (cont)



S “can-supply” P, D “needs” P, and D “deals-with” S does not imply that D has agreed to buy P from S.

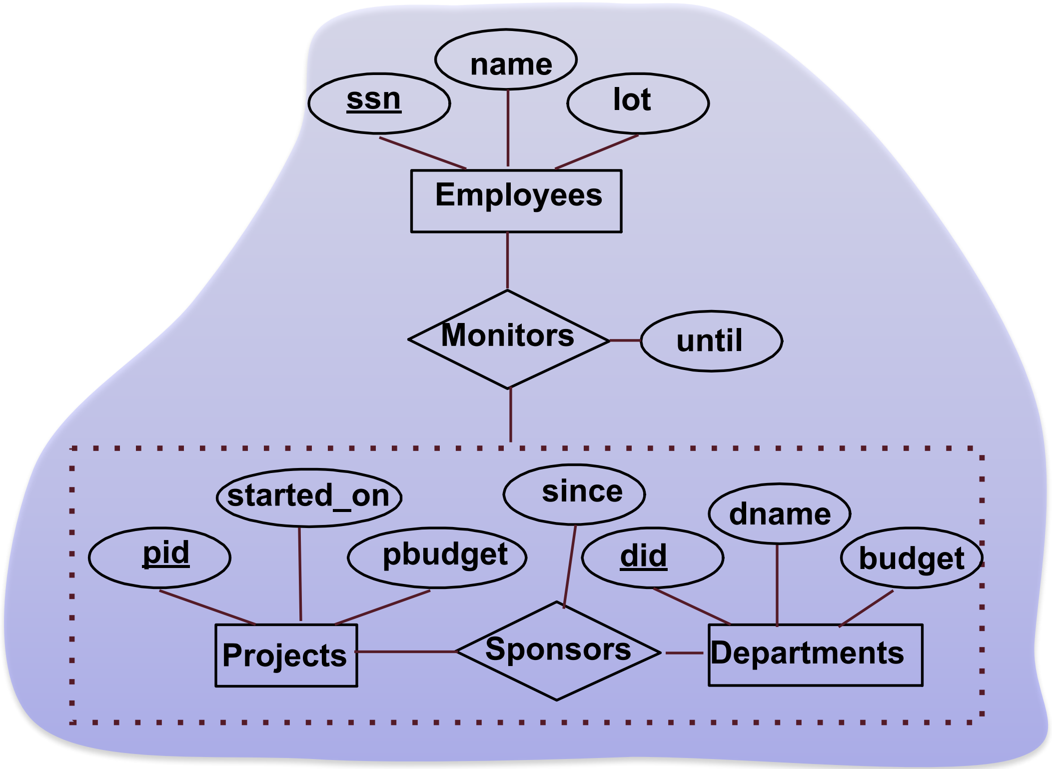
How do we record *qty*?

vs.

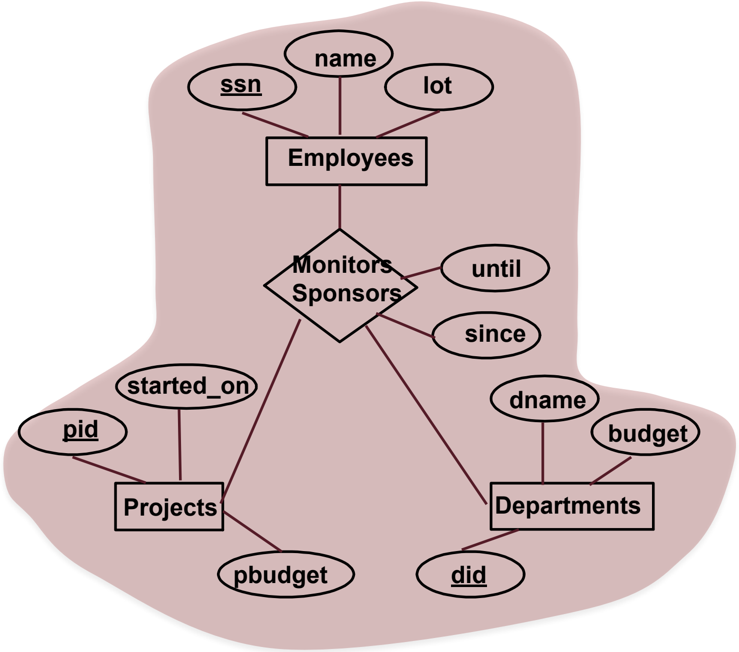


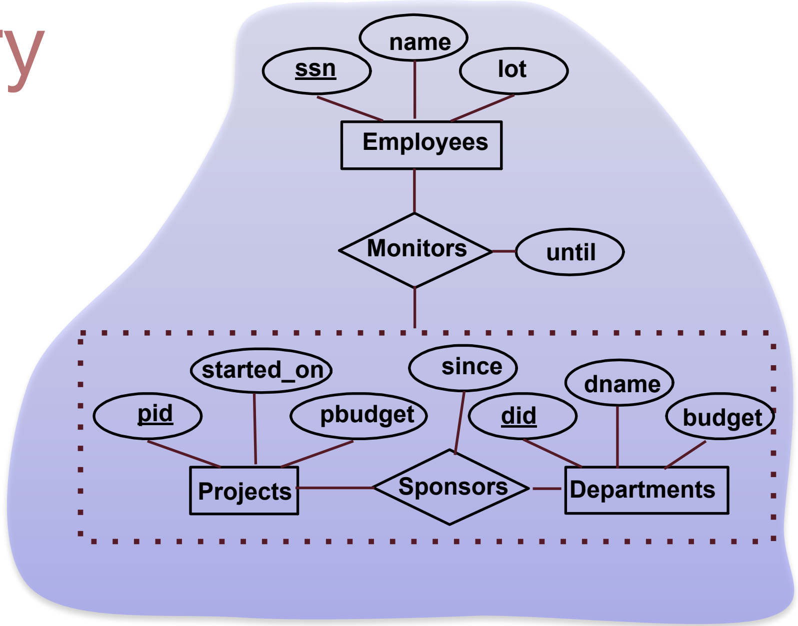
Aggregation

Allows relationships to have relationships.



Aggregation vs. Ternary





Conceptual Design Using the ER Model

- ER modeling can get tricky!
- Design choices:
  - **Entity** or **attribute**?
  - **Entity** or **relationship**?
  - Relationships: **Binary or ternary? Aggregation?**
- ER Model goals and limitations:
  - Lots of semantics can (and should) be captured.
  - Some constraints cannot be captured in ER.
    - We'll refine things in our logical (relational) design

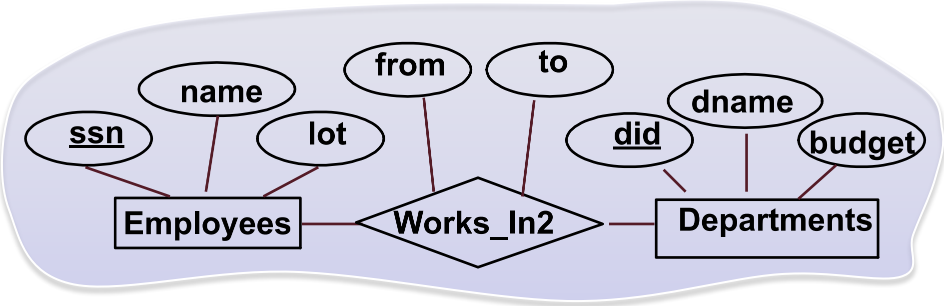
Entity vs. Attribute

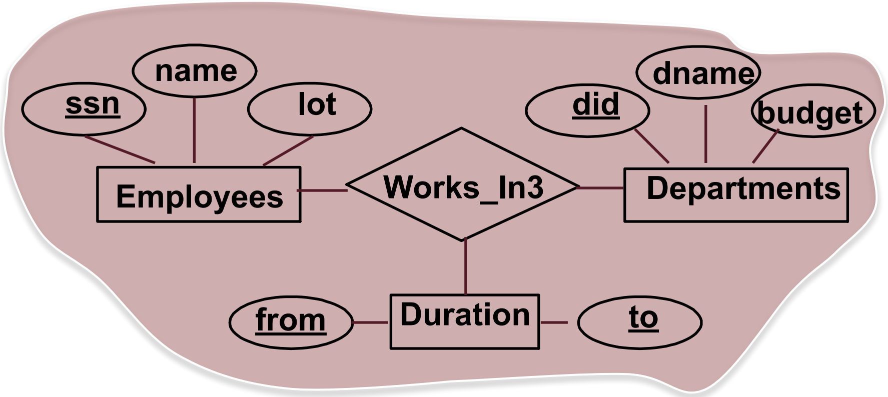
- "Address":
  - attribute of Employees?
  - Entity of its own?
- It depends! Semantics and usage.
  - Several addresses per employee?
    - must be an entity
    - atomic attribute types (no set-valued attributes!)
  - Care about structure? (city, street, etc.)
    - must be an entity!
    - atomic attribute types (no tuple-valued attributes!)

Entity vs. Attribute (Cont.)

- Works_In2: employee cannot work in a department for >1 period.

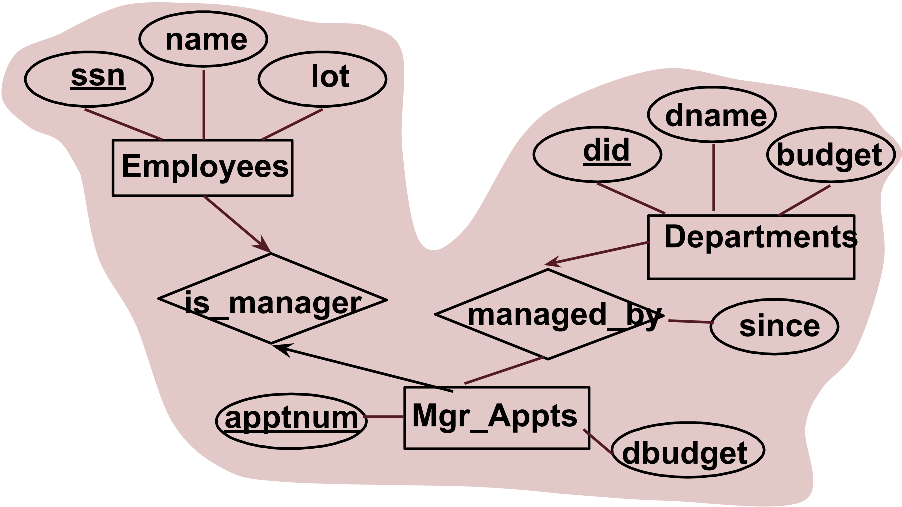
- Like multiple addresses per employee!

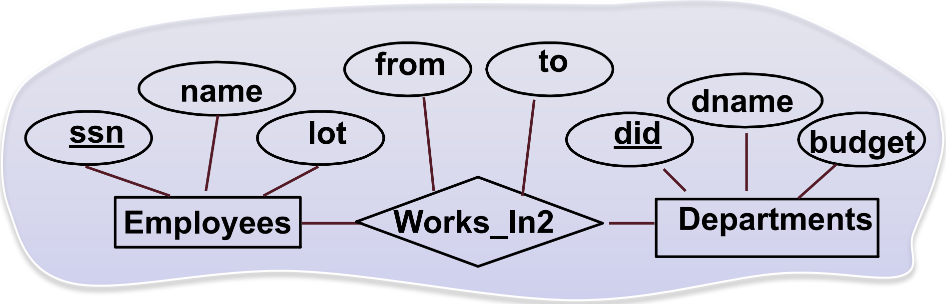




Entity vs. Relationship

- Separate discretionary budget (dbudget) for each dept.
- What if manager's dbudget covers all managed depts
  - Could repeat value
  - But redundancy = problems
- Better design:





E-R Diagram as Wallpaper

- Very common for them to be wall-sized

Steps in Database Design, Part 4

- Requirements Analysis
  - user needs; what must database do?
- Conceptual Design
  - *high level description (often done w/ER model)*
  - ORM encourages you to program here
- **Logical Design**
  - **translate ER into DBMS data model**
  - **ORMs often require you to help here too**
- Schema Refinement
  - consistency, normalization
- Physical Design - indexes, disk layout
- Security Design - who accesses what, and how

Completed

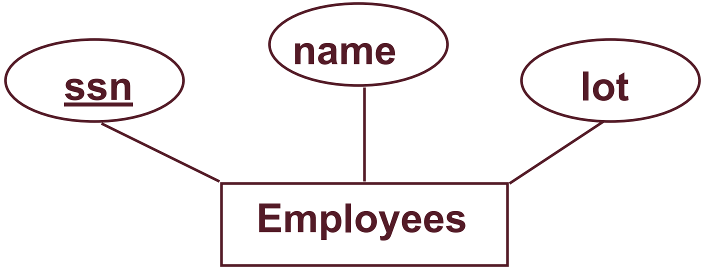
You are here

Converting ER to Relational

- Fairly analogous structure
- But many simple concepts in ER are subtle to specify in relations

Logical DB Design: ER to Relational

- Entity sets to tables.
  Easy.



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.

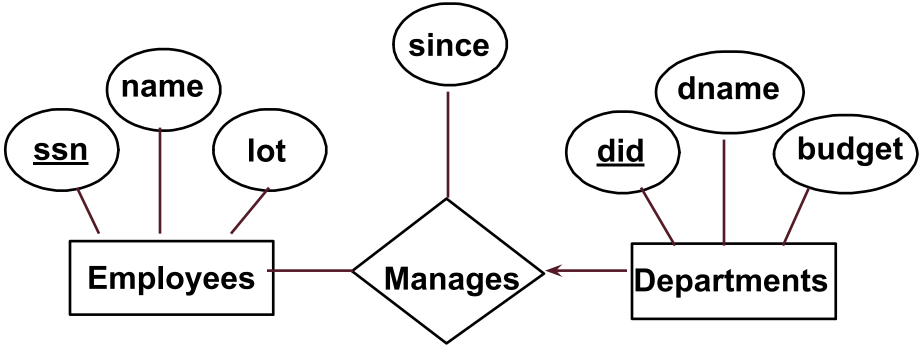
2) All descriptive attributes.

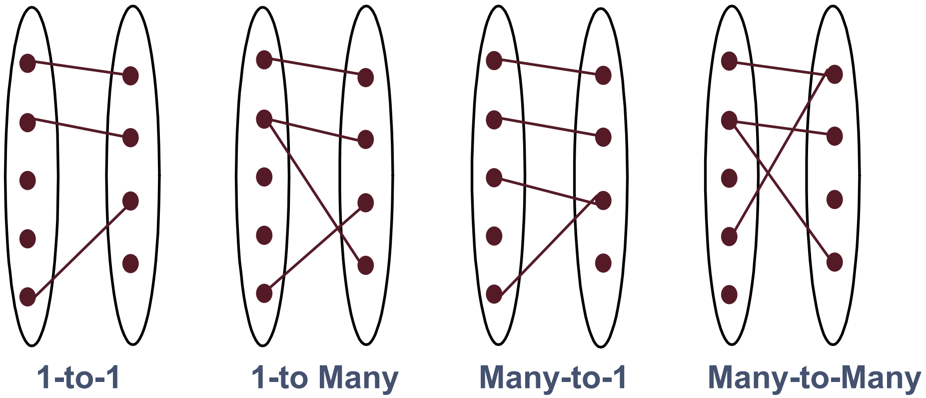
```
CREATE TABLE Works_In(
  ssn  CHAR(1),
  did  INTEGER,
  since  DATE,
  PRIMARY KEY (ssn, did),
  FOREIGN KEY (ssn)
    REFERENCES Employees,
  FOREIGN KEY (did)
    REFERENCES Departments)
```

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

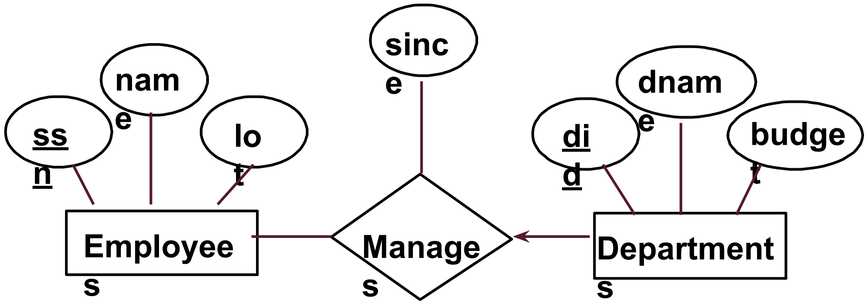
Review: Key Constraints

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



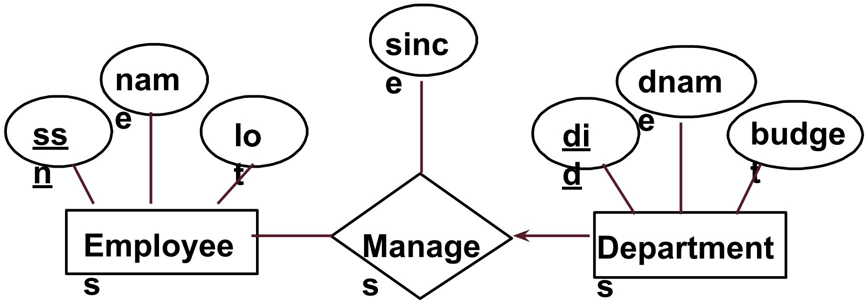


Translating ER with Key Constraints




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

Translating ER with Key Constraints, cont



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

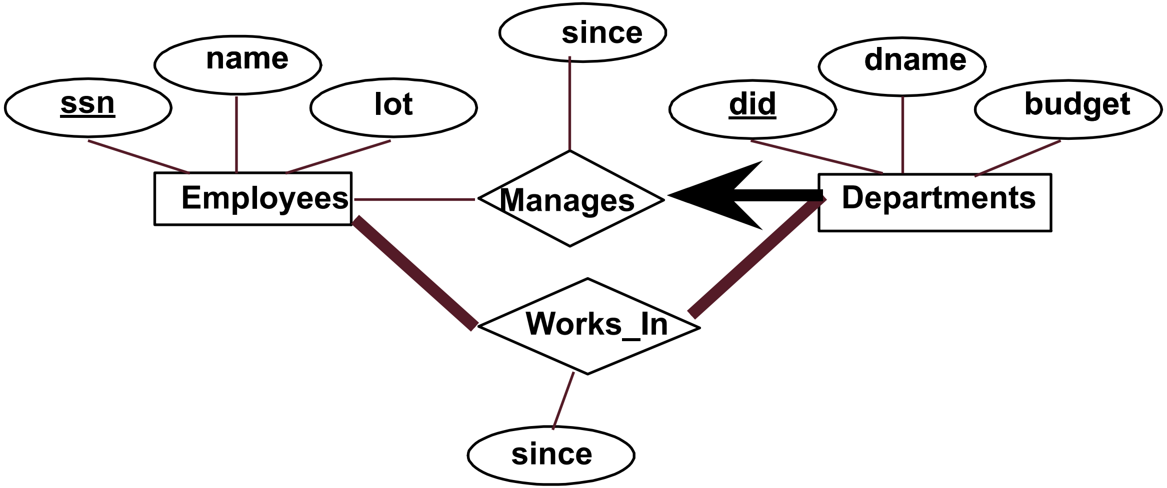
```
CREATE TABLE  Manages(
 ssn  CHAR(11),
 did  INTEGER,
 since  DATE,
 PRIMARY KEY  (did),
 FOREIGN KEY (ssn)
   REFERENCES Employees,
 FOREIGN KEY (did)
   REFERENCES Departments)
```

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: Key+Participation Constraints

- Every department has one manager.
  - Every did value in Departments table must appear in a row of the Manages table (with a non-null ssn value!)



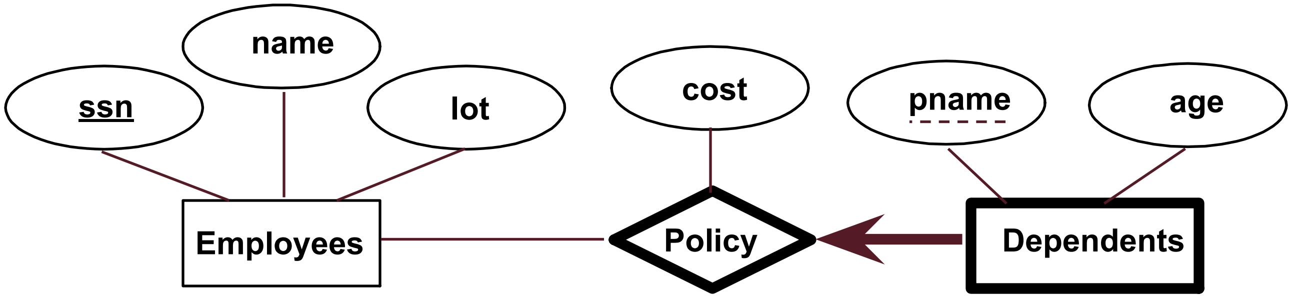
Participation Constraints in SQL

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

```
CREATE TABLE  Dept_Mgr(
   did  INTEGER,
   dname  CHAR(20),
   budget  REAL,
   ssn  CHAR(11) NOT NULL, -- total participation!
   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)
```

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 we think about 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 (see text if you're curious), and aggregation.

Summary of ER (Cont.)

- Basic integrity constraints
  - **key constraints**
  - **participation constraints**
- 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**. Many ways to model a given scenario!
- Analyzing alternatives can be tricky! Common choices include:
  - Entity vs. attribute, entity vs. relationship, binary or n-ary relationship, whether or not to use aggregation
- For good DB design: resulting relational schema should be analyzed and refined further.
  - Functional Dependency information
    + normalization coming in subsequent lecture.