Entity-Relationship Diagrams

Fall 2017, Lecture 3

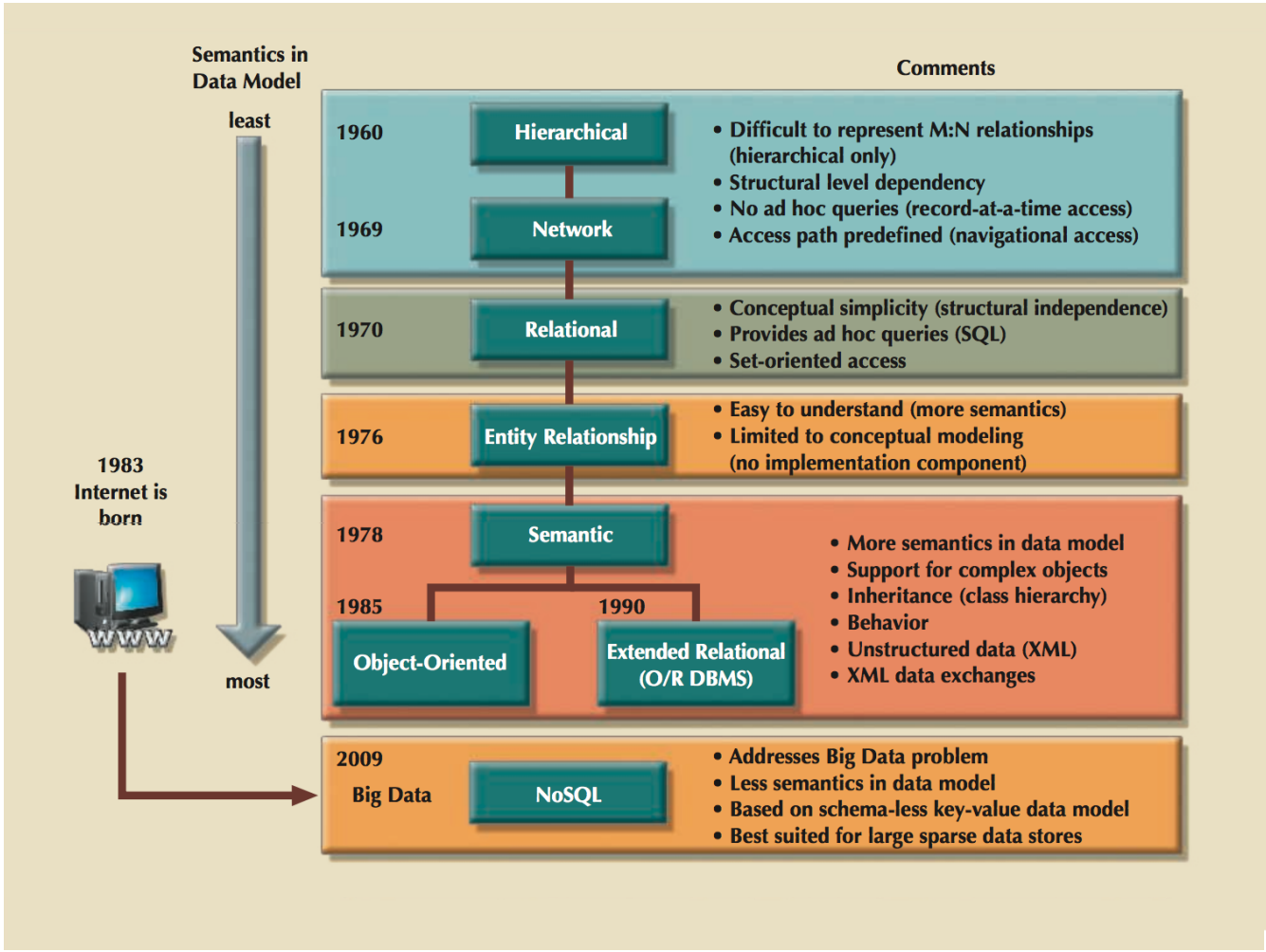
There is nothing worse than a sharp image of a fuzzy concept.

Ansel Adams

Recall: Relational Database Management

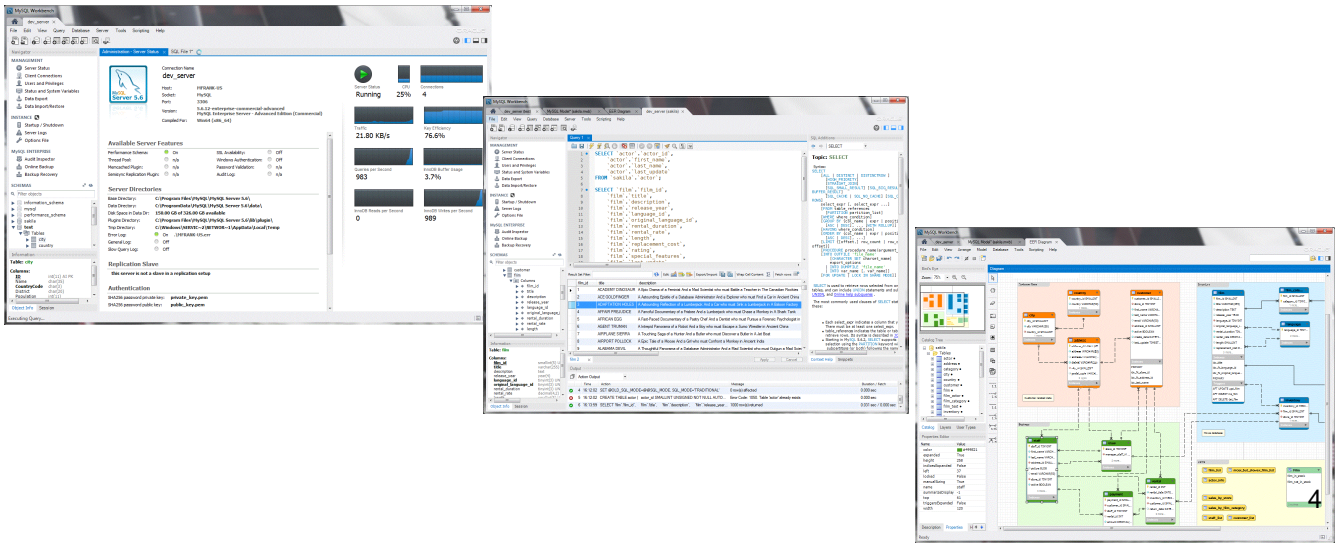
Relational DataBase Management Systems were invented to let you use one set of data in multiple ways, including ways that are unforeseen at the time the database is built and the 1st applications are written.

(Curt Monash, analyst/blogger)

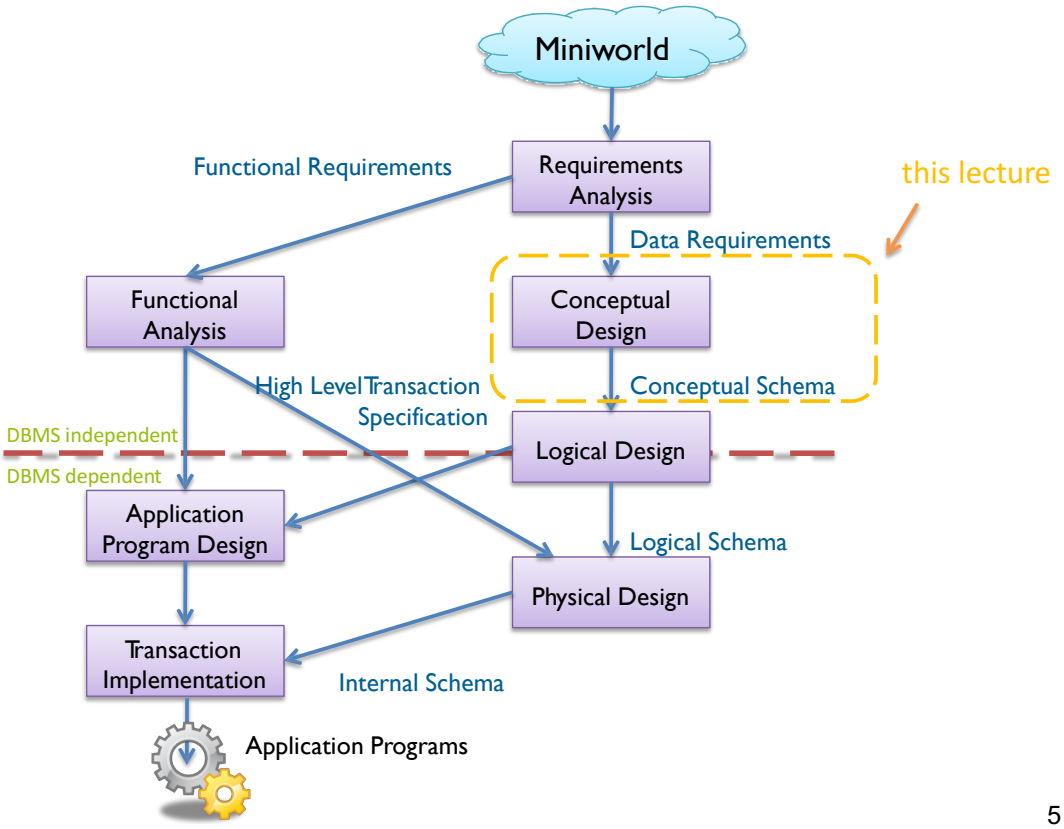


Software to be used in this Chapter…

- MySQL Workbench which can be downloaded from http://www.mysql.com/products/workbench/



Recap: Steps in Database Design



Phases of DB Design

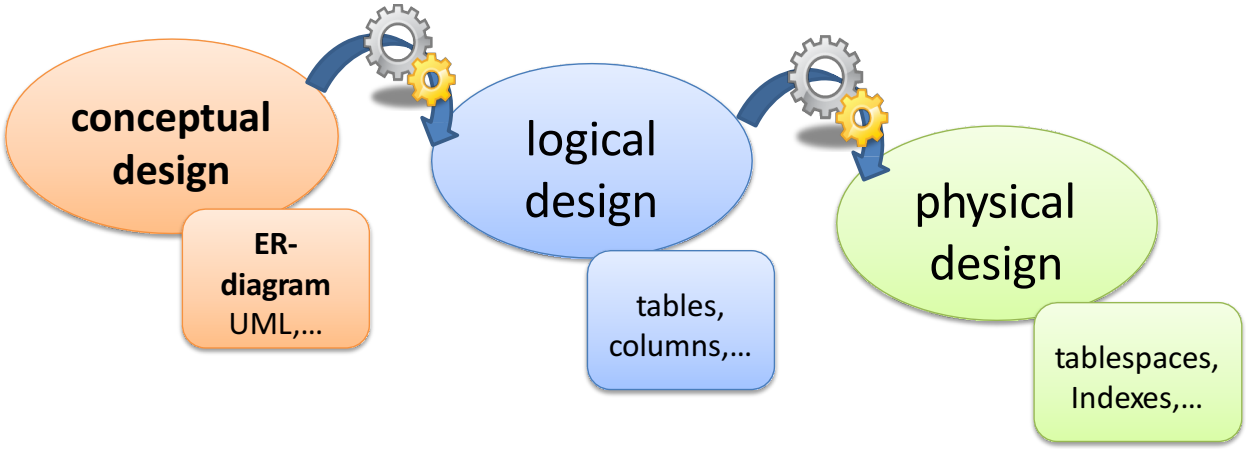
- Requirements Analysis
  - Database designers interview prospective users and stakeholders
  - Data requirements describe what kind of data is needed
  - Functional requirements describe the operations performed on the data
- Functional Analysis
  - Concentrates on describing high-level user operations and transactions
    - Does also not contain implementation details
    - Should be matched versus conceptual model

Phases of DB Design

- Conceptual Design
  - Transforms data requirements to conceptual model
  - Conceptual model describes data entities, relationships, constraints, etc. on high-level
    - Does not contain any implementation details
    - Independent of used software and hardware
- Logical Design
  - Maps the conceptual data model to the logical data model used by the DBMS
    - e.g. relational model, hierarchical model, …
    - Technology independent conceptual model is adapted to the used DBMS software
- Physical Design
  - Creates internal structures needed to efficiently store/manage data
    - Table spaces, indexes, access paths, …
    - Depends on used hardware and DBMS software

Phases of DBMS Design

- While modeling the data, three design phases have to be completed
  - The result of a phase serves as input to the next phase
  - Often, automatic transition is possible with some additional designer feedback



Example: DBA for Bank of America

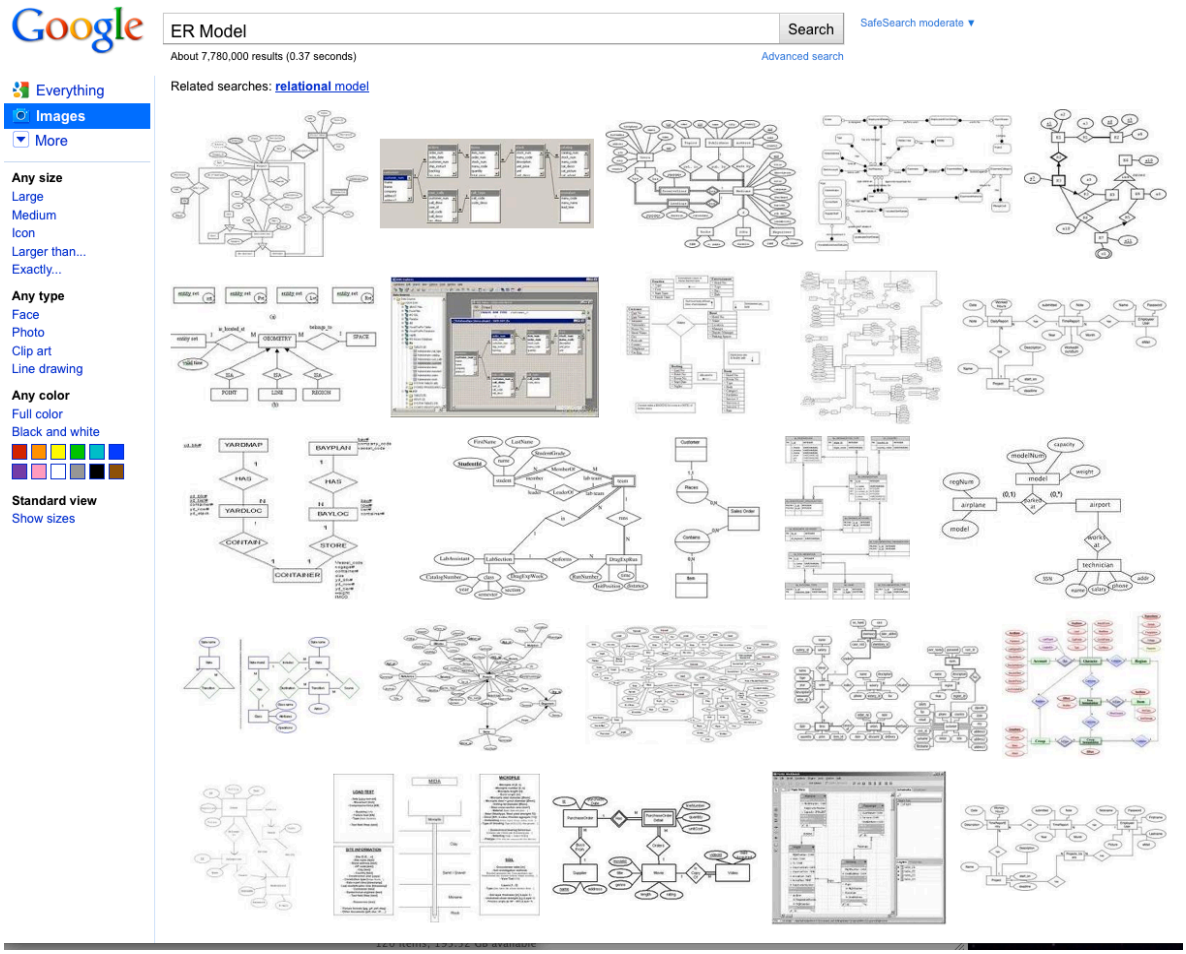
- Requirements Specification
  - Determine the requirements of clients (Database to store information about customers, accounts, loans, branches, transactions, …)
- Conceptual Design
  - Express client requirements in terms of E/R model.
  - Confirm with clients that requirements are correct.
  - Specify required data operations
- Logical Design
  - Convert E/R model to relational, object-based, XML-based,…
- Physical Design
  - Specify file organizations, build indexes

ER Modeling

- Traditional approach to **Conceptual Modeling**
  - Entity-Relationship Models (ER-Models)
    - Also known as Entity-Relationship Diagrams (ERD)
    - Introduced in 1976 by Peter Chen
    - Graphical representation
- Top-Down-Approach for modeling
  - Entities and Attributes
  - Relationships
  - Constraints
- Some derivates became popular
  - ER Crow's Foot Notation (Bachman Notation)
  - ER Baker Notation
  - Later: Unified Modeling Language (UML)

ER Modeling

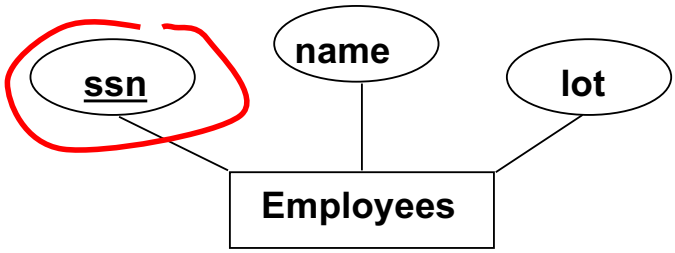
- What are the entities and relationships?
- What info about E's & R's should be in DB?
- What *integrity constraints* (*business rules*) hold?
- *ER diagram* is a representation of the `schema'
- Can map an ER diagram into a relational schema.
- Conceptual design is where the SW/data engineering *begins*



E-R Diagram as Wallpaper

- Very common for them to be wall-sized

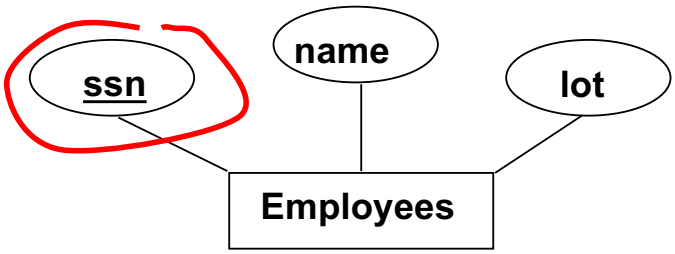
ER Model Basics



- *Entity*:
  - Real-world object, distinguishable from other objects.
  - Described using a set of *attributes*.
  - Has its own identity and represents just one thing

- *Entity Set*: A collection of similar entities. E.g., all employees.
  - All entities in an entity set have the same set of attributes. (Until we consider hierarchies, anyway!)
  - Each entity set has a *key (underlined)*.
  - Each attribute has a *domain*.

ER Model Basics



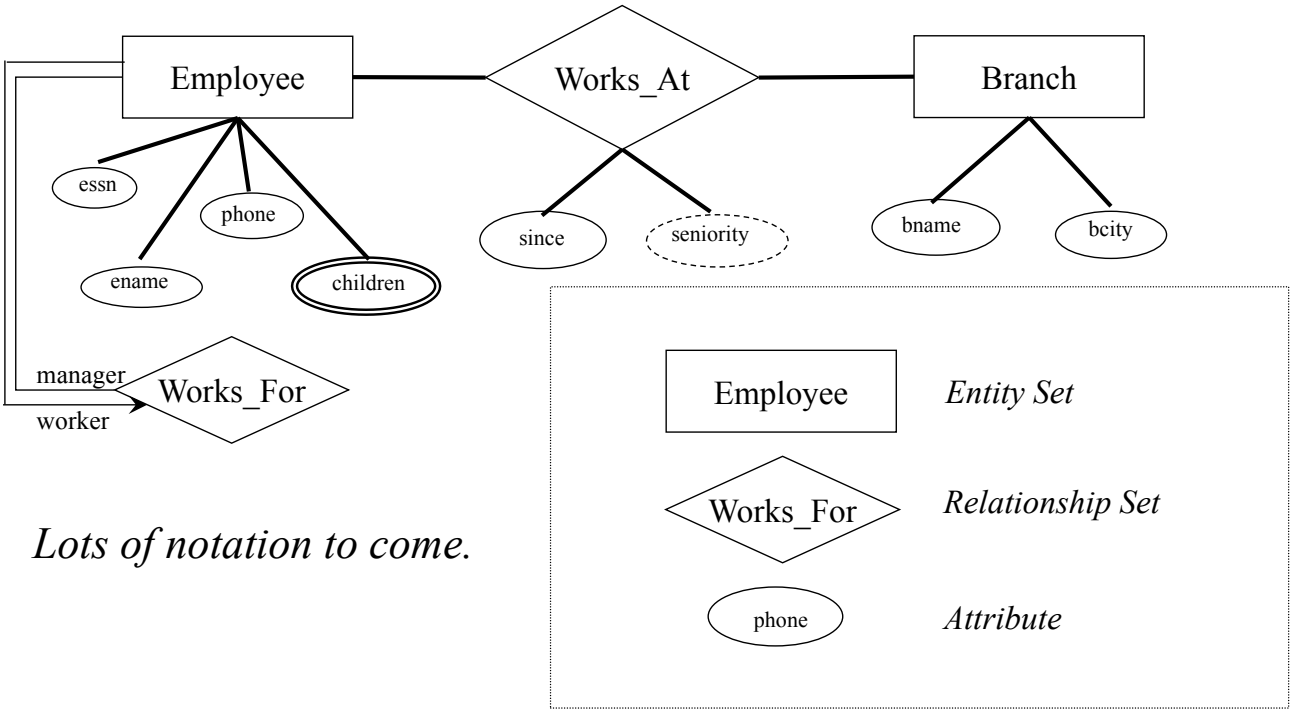
ssn	name	lot
123-22-3666	Attishoo	48
231-31-5368	Smiley	22
131-24-3650	Smethurst	35

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


E/R Diagrams

- In an entity-relationship diagram:
  - Entity set = rectangle.
  - Attribute = oval, with a line to the rectangle representing its entity set.

E/R Data Model: An Example

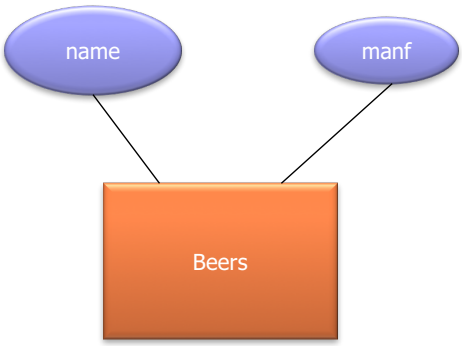


Lots of notation to come.

In other words …

- *Entity* = "thing" or object.
- *Entity set* = collection of similar entities.
  - Similar to a class in object-oriented languages.
- *Attribute* = property of (the entities of) an entity set.
  - Attributes are simple values, e.g. integers or character strings, not structs, sets, etc.
  - Example: name of an employee, color of a car, balance of an account, location of a house,…

Example:



- Entity set Beers has two attributes, name and manf (manufacturer).
- Each Beers entity has values for these two attributes, e.g. (Corona, Grupo Modelo)

Entity Relationship Diagram Symbols

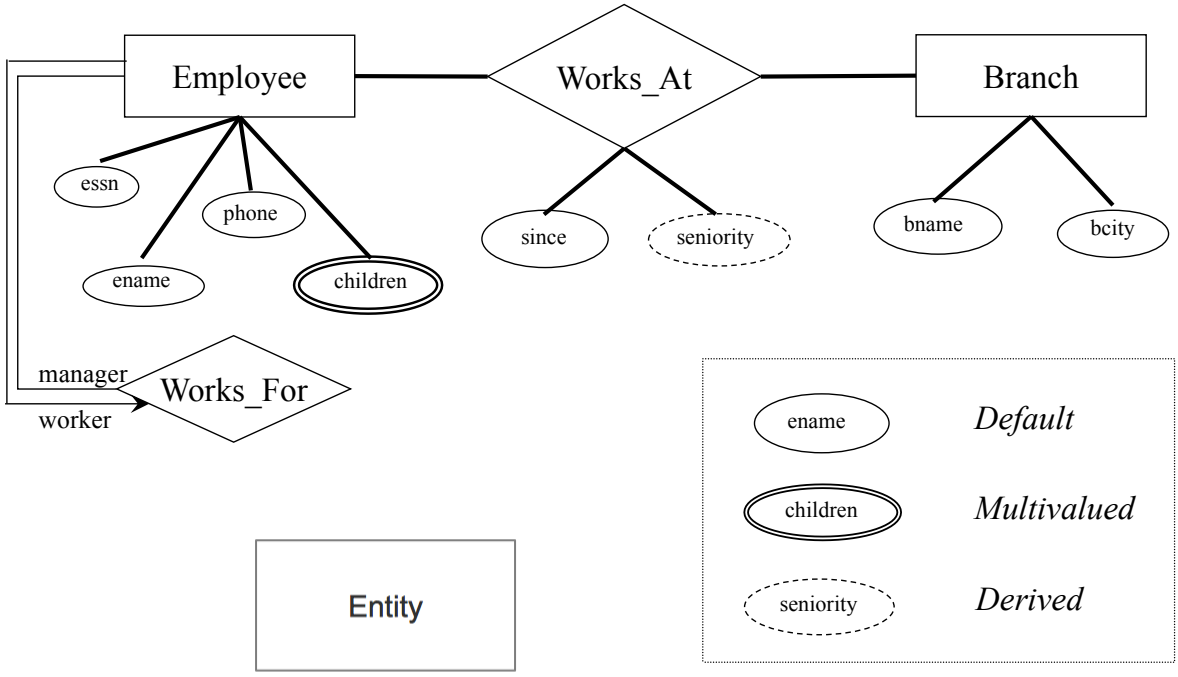
Entity	An entity is represented by a rectangle which contains the entity's name.
Weak Entity	An entity that cannot be uniquely identified by its attributes alone. The existence of a weak entity is dependent upon another entity called the owner entity.
Attribute	In the Chen notation, each attribute is represented by an oval containing atributte's name
Derived attribute	An attribute whose value is calculated (derived) from other attributes.
Multivalue attribute	An attribute that can have many values (there are many distinct values entered for it in the same column of the table).

Entity Relationship Diagram Symbols

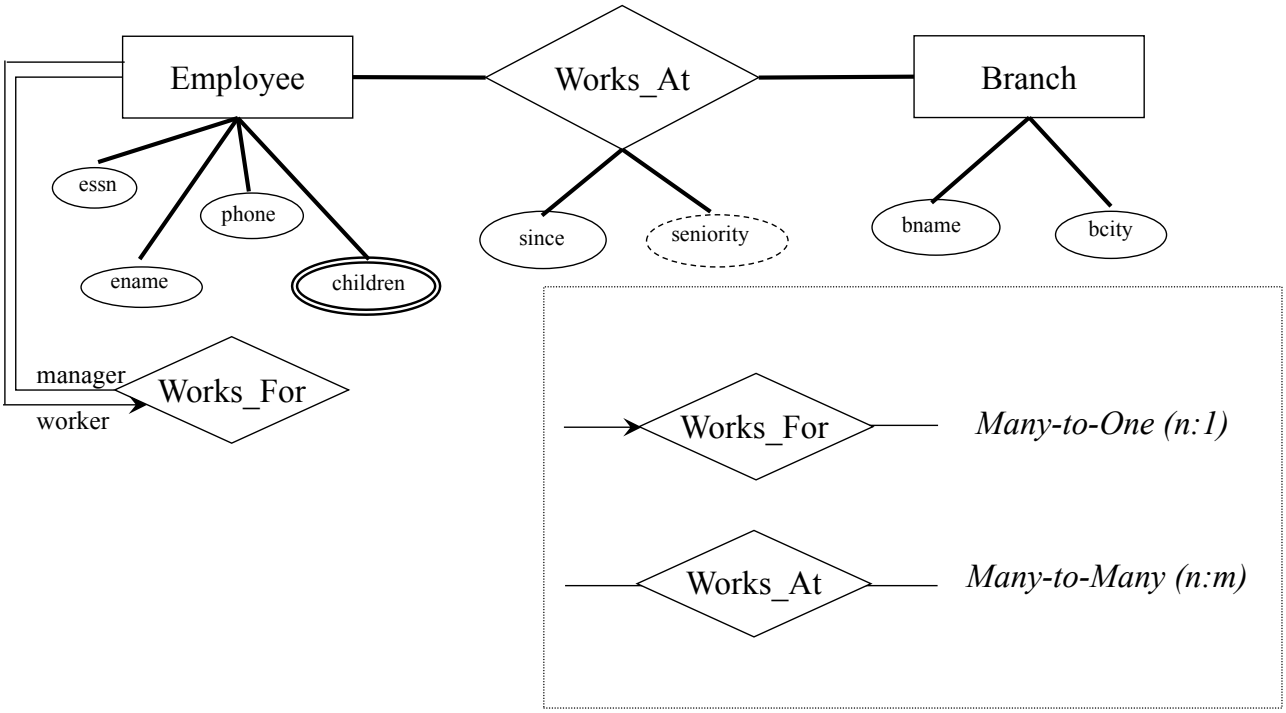
A relationship where entity is existence-independent of other entities, and PK of Child doesn't contain PK component of Parent Entity

A relationship where Child entity is existence-dependent on parent, and PK of Child Entity contains PK component of Parent Entity.

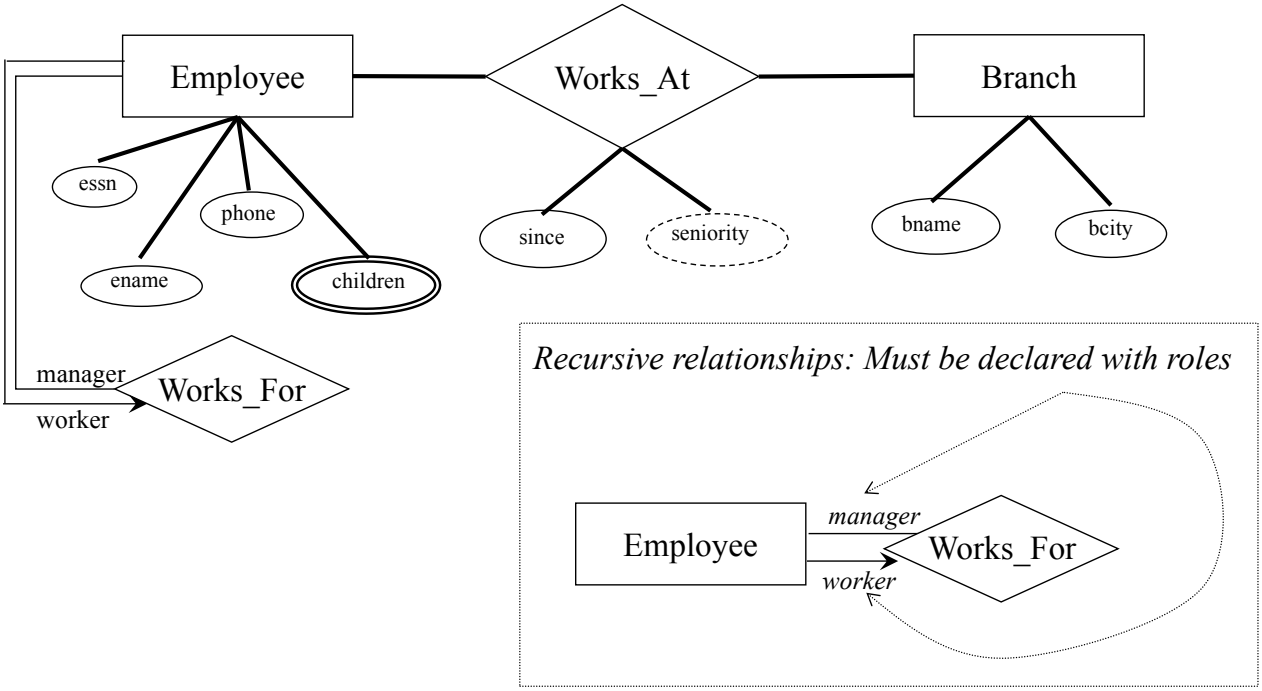
E/R Data Model: Types of Attributes



E/R Data Model: Types of relationships



E/R Data Model: Recursive relationships



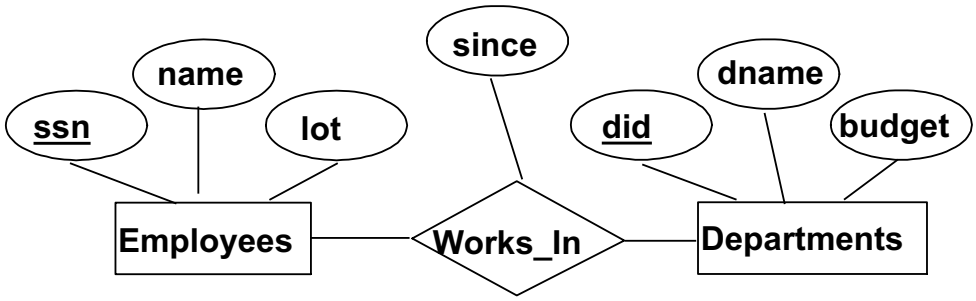
Business Rules

- A **business rule** is "a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behavior of the business...rules prevent, cause, or suggest things to happen"

- Entity-relationship diagrams are used to document rules and policies of an organization

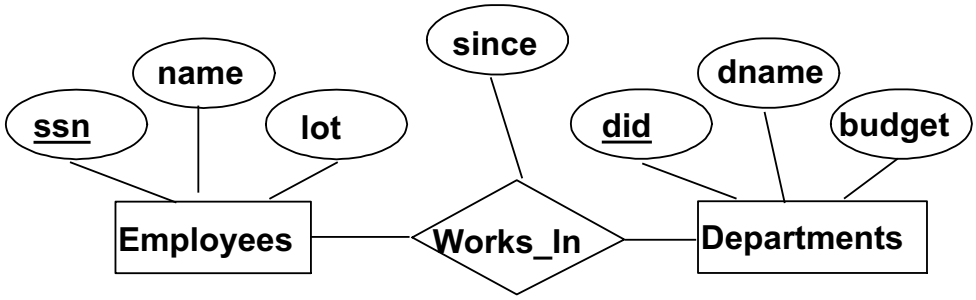
In fact, documenting rules and policies of an organization that govern data is exactly what data modeling is all about.

Relationships



- *Relationship*: Association among two or more entities. E.g., Attishoo works in Pharmacy department.
  - relationships can have their own attributes.
- *Relationship Set*: Collection of similar relationships.
  - An *n*-ary relationship set *R* relates *n* entity sets $E_1 \ldots E_n$; each relationship in *R* involves entities $e_1 \in E_1, \ldots, e_n \in E_n$

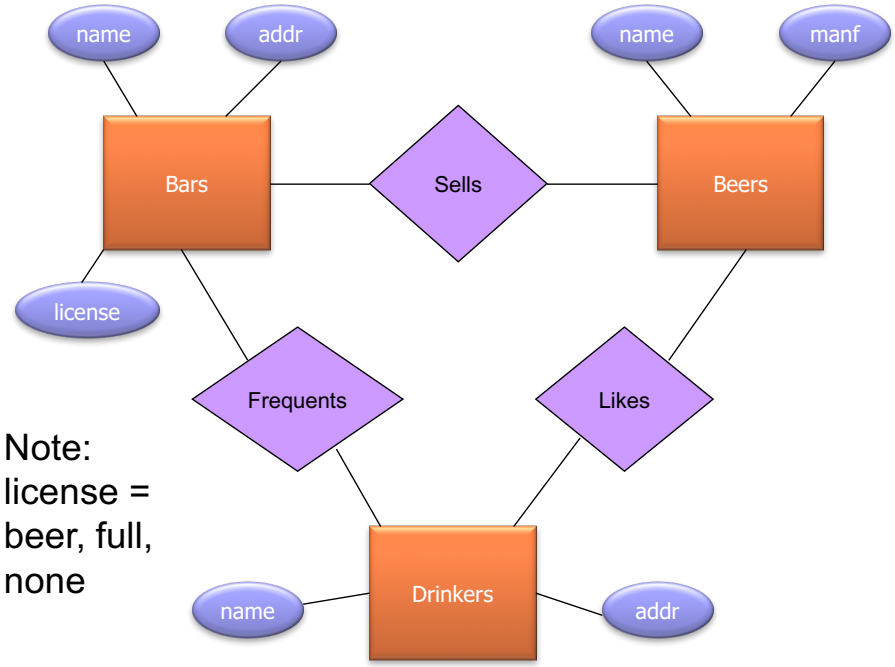
Relationships




```
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)
```

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

Example: Relationships



Bars sell some beers.

Drinkers like some beers.

Drinkers frequent some bars.

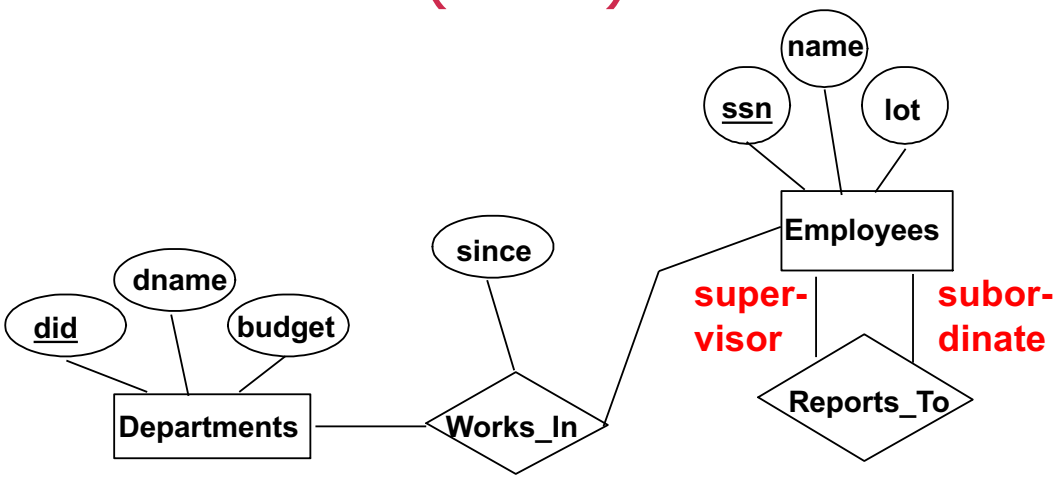
Note: license = beer, full, none

Example: Relationship Set

- For the relationship Sells, we might have a relationship set like:

Bar	Beer
Joe's Bar	Bud
Joe's Bar	Miller
Sue's Bar	Bud
Sue's Bar	Pete's Ale
Sue's Bar	Bud Lite

ER Model Basics (Cont.)

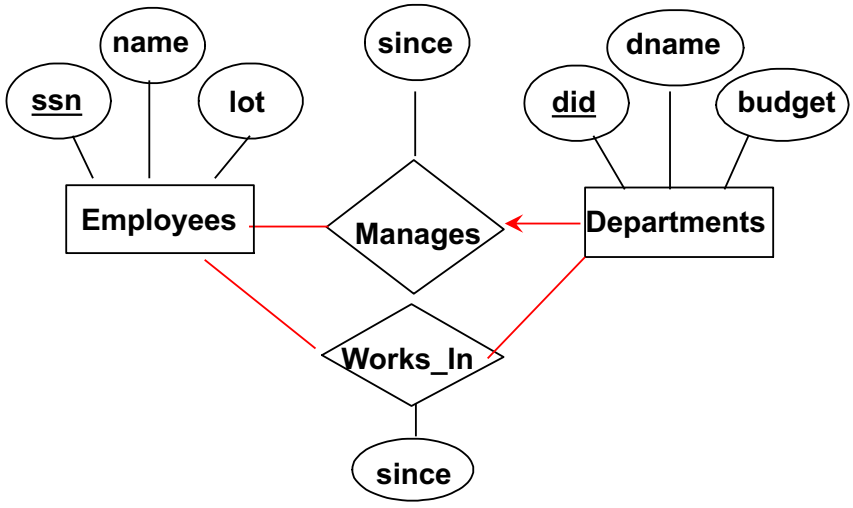


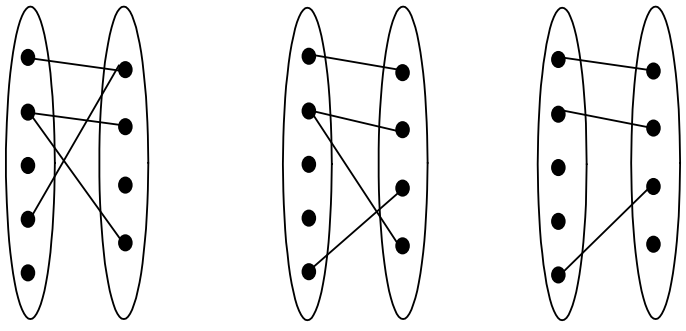
- Same entity set can participate in different relationship sets, or in different "roles" in the same set.

Key Constraints

An employee can work in many departments; a dept can have many employees.

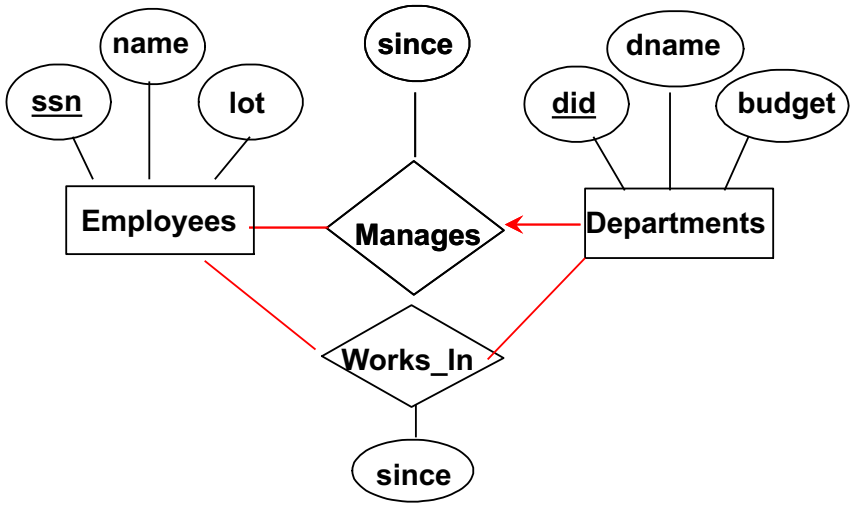
In contrast, each dept has at most one manager, according to the *key constraint* on Manages.





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

Key Constraints



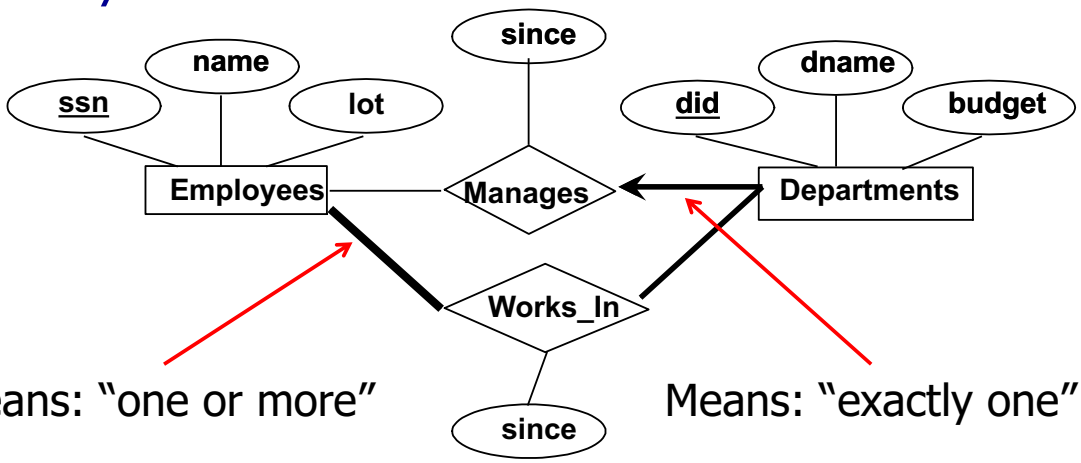
An employee can work in many departments; a dept can have many employees.

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

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


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 "one or more"



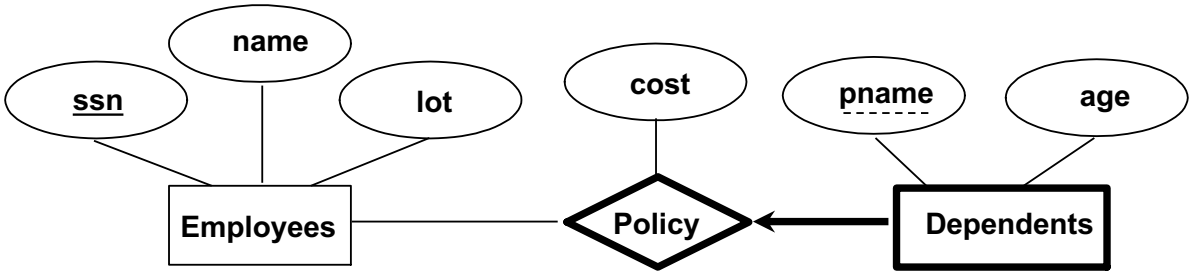
Means: "one or more"

Means: "exactly one"

Weak Entities

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

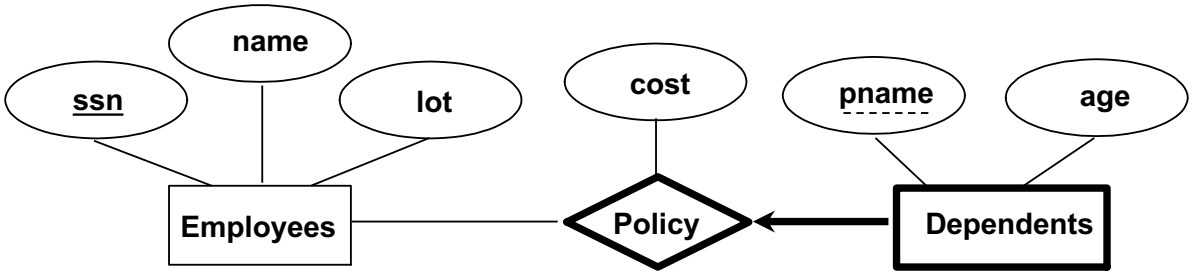
- Owner entity set and weak entity set must participate in a one-to-many relationship set (one owner, many weak entities).
- Weak entity set must have total participation in this *identifying* relationship set.



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

Weak Entities

```
CREATE TABLE     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)
```

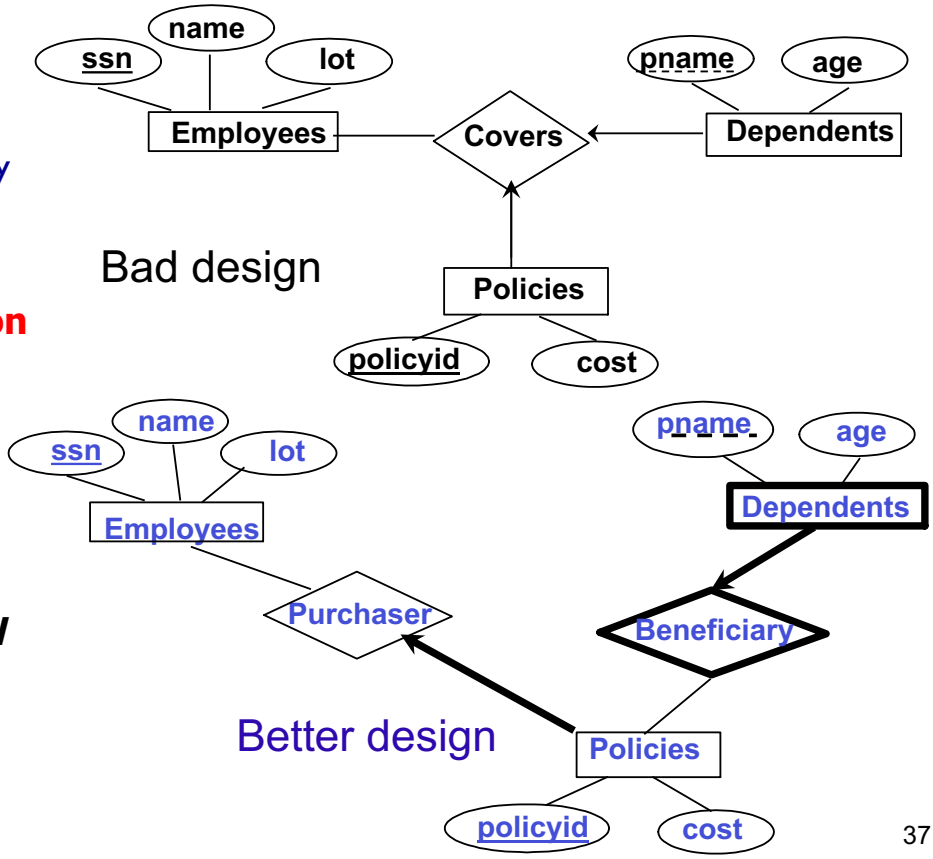


Binary vs. Ternary Relationships

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 (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?

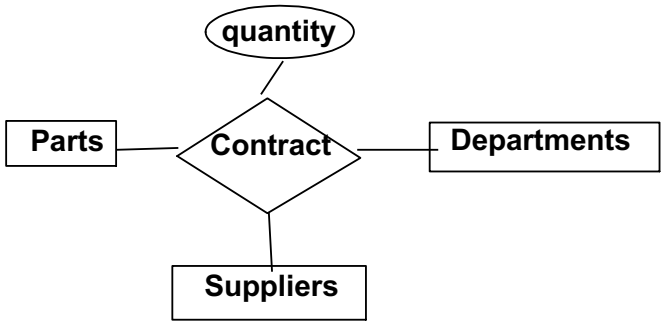
```
CREATE TABLE Policies (
  policyid INTEGER,
  cost REAL,
  ssn CHAR(11) NOT NULL,
  PRIMARY KEY (policyid).
  FOREIGN KEY (ssn) REFERENCES Employees,
    ON DELETE CASCADE)

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

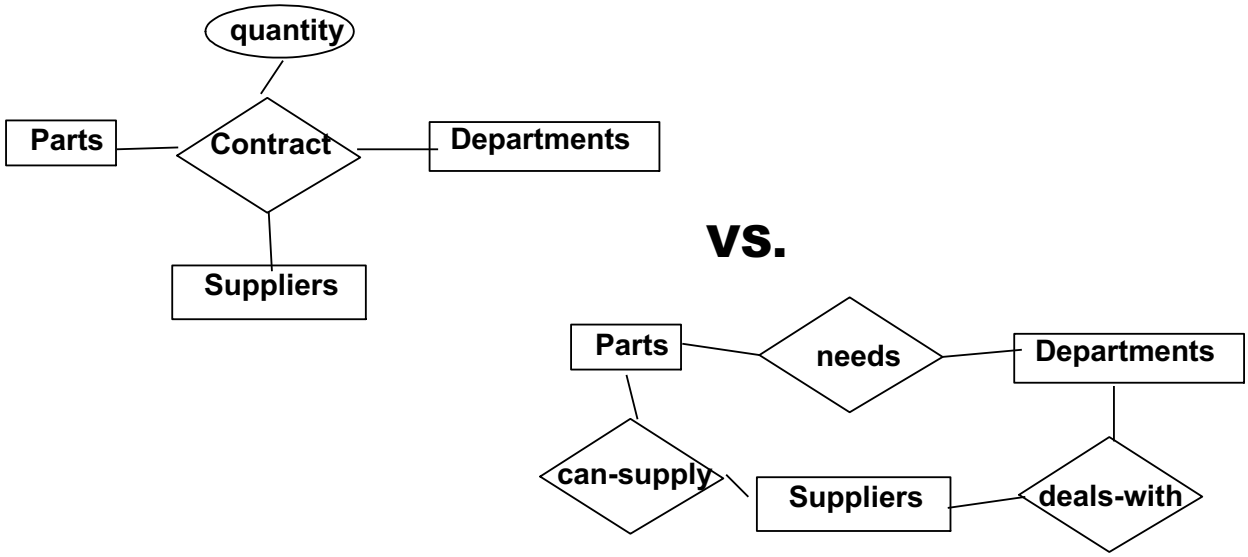
Binary vs. Ternary Relationships (Contd.)

- 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.

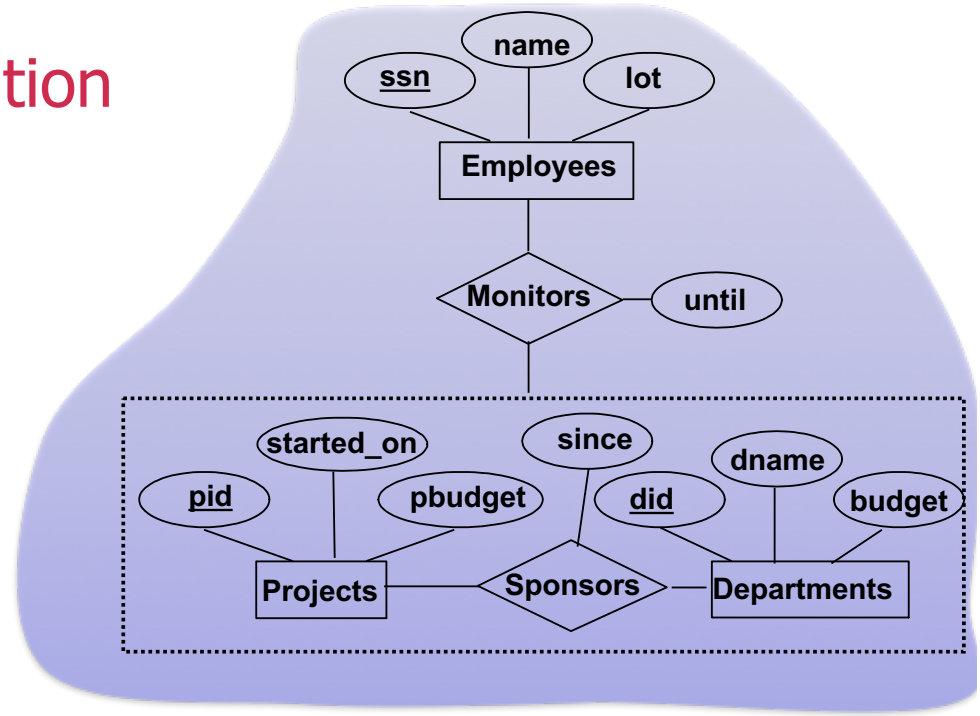


Binary vs. Ternary Relationships (Contd.)



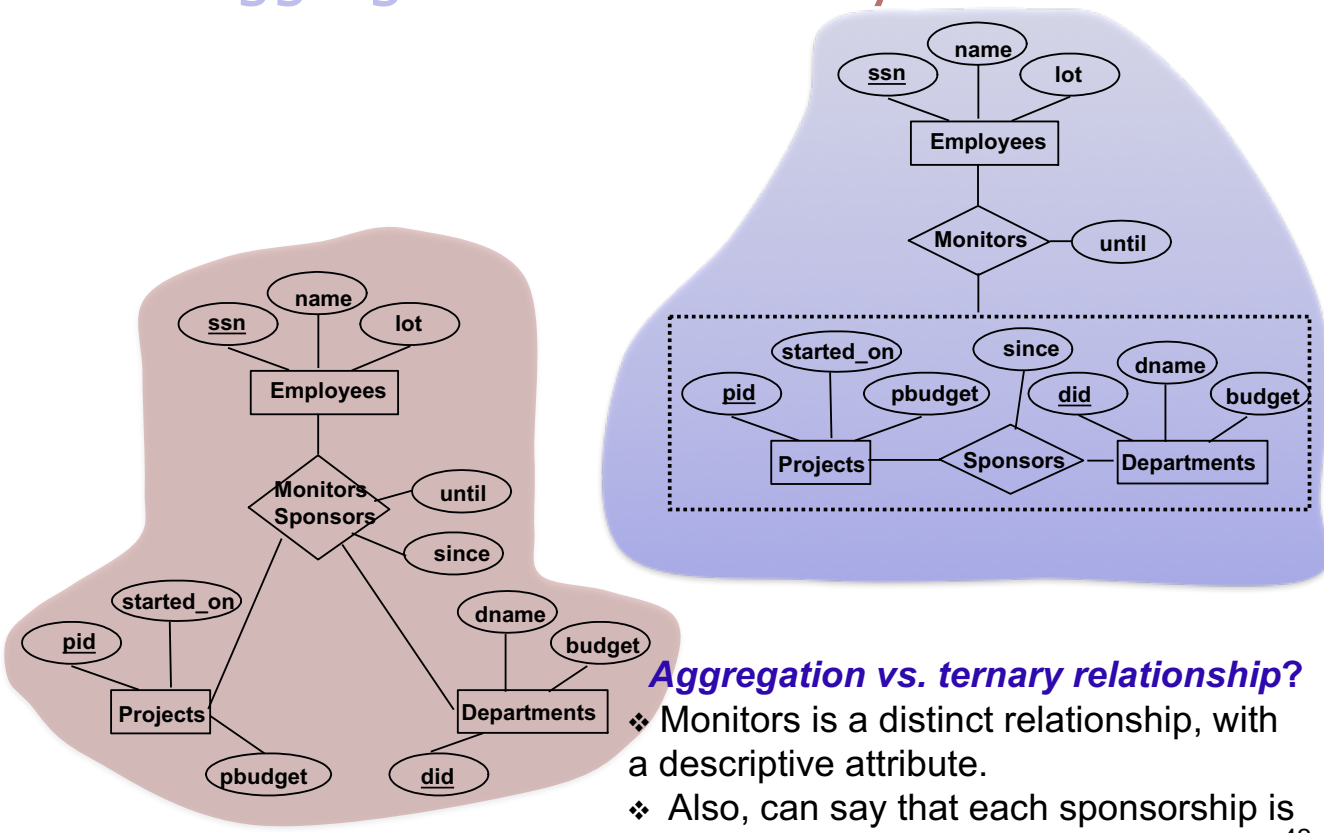
- S "can-supply" P, D "needs" P, and D "deals-with" S does not imply that D has agreed to buy P from S.
- How do we record *qty*?

Aggregation



Allows relationships with *relationship sets*.

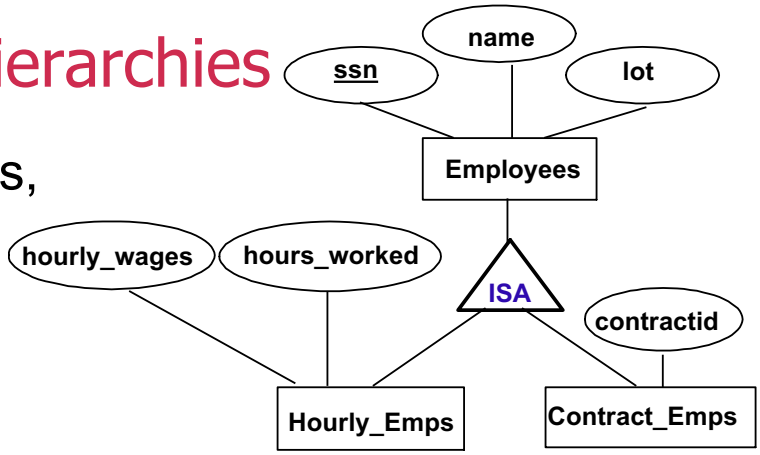
Aggregation vs. Ternary



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



- 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 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

Conceptual Design Using the ER Model

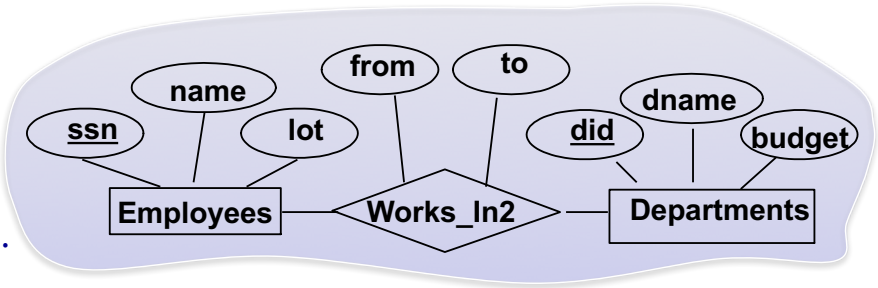
- ER modeling *can* get tricky!
- Design choices:
  - Should a concept be modeled as an entity or an attribute?
  - Should a concept be modeled as an entity or a relationship?
  - Identifying relationships: Binary or ternary? Aggregation?
- Note constraints of the ER Model:
  - A lot of data semantics 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

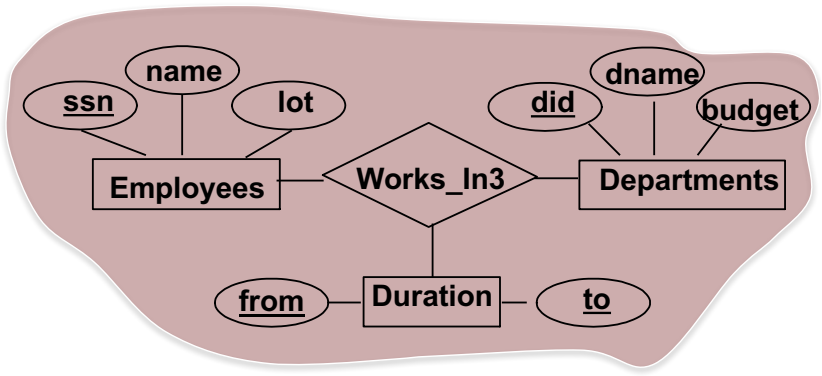
- "Address":
  - attribute of Employees?
  - Entity of its own?
- It depends! Semantics and usage.
  - Several addresses per employee?
    - must be an entity!
    - atomic attribute types (no set-valued attributes!)
  - Care about structure? (city, street, etc.)
    - must be an entity!
    - atomic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

Entity vs. Attribute (Cont.)

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

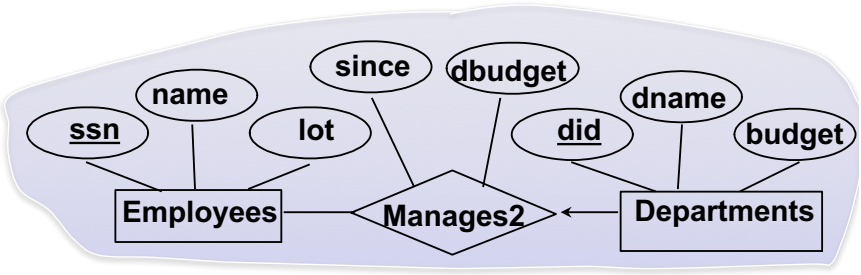


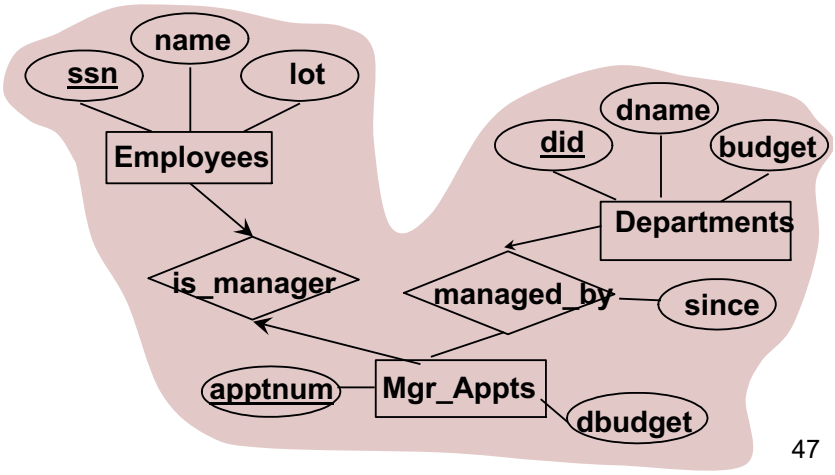
- Like multiple addresses per employee!



Entity vs. Relationship

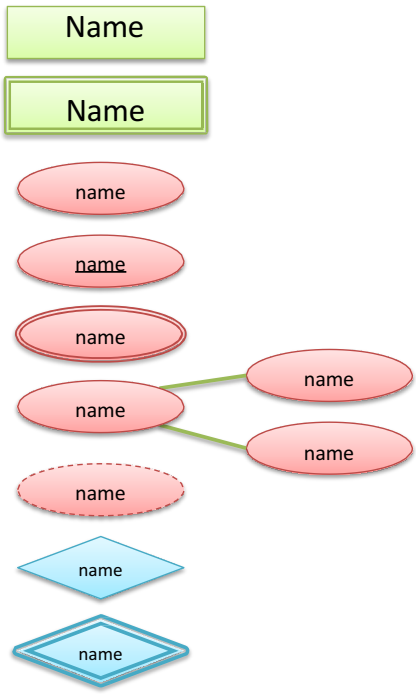
- Separate discretionary budget (dbudget) for each dept.
- What if manager's dbudget covers all managed depts
  - Could repeat value
  - But redundancy = problems
- Better design:





Model Review

- Entity Type
- Weak Entity Type
- Attribute
- Key Attribute
- Multi-valued Attribute
- Composite Attribute
- Derived Attribute
- Relationship Type
- Identifying Rel.Type



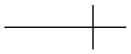
Model Review (continued)

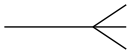
- Entities and Entity Set (boxes)
- Relationships and Relationship sets (diamonds)
  - binary
  - n-ary
- Key constraints (1-1,1-M, M-M, arrows on 1 side)
- Participation constraints (bold for Total)
- Weak entities - require strong entity for key
- Aggregation - an alternative to n-ary relationships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

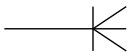
Alternative Notations

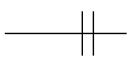
- Chen Model
  - 1 to represent one.
  - M to represent many

- Crow's Foot


One

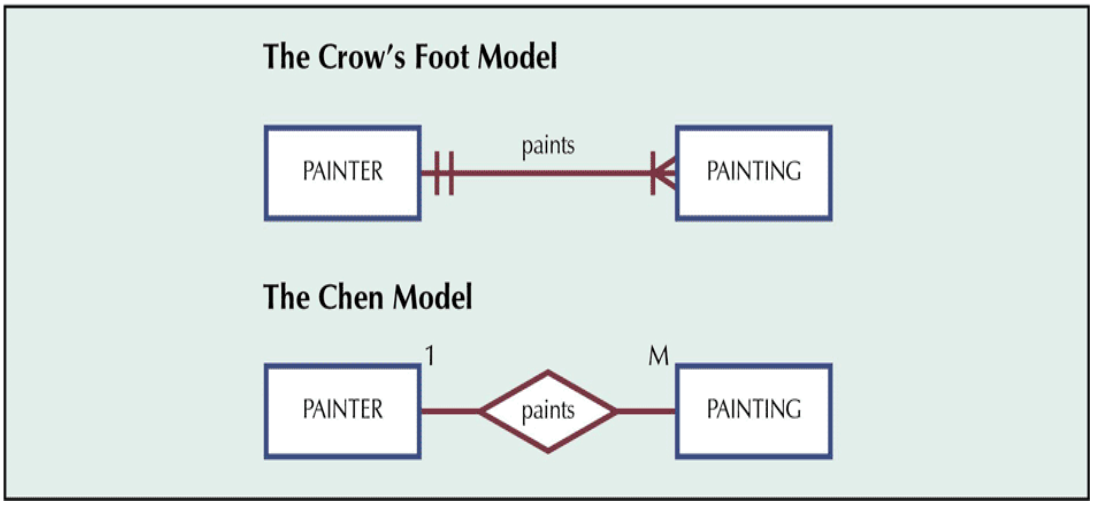

many


One or many


Mandatory one , means (1,1)

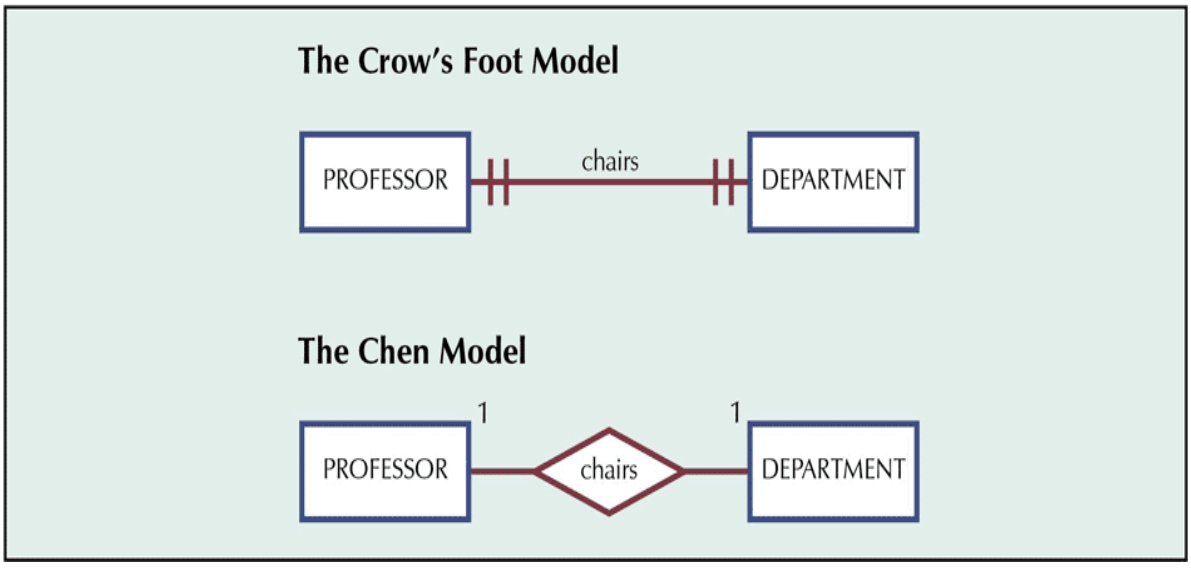
Symbol	Meaning
	One—Mandatory
	Many—Mandatory
	One—Optional
	Many—Optional

Crow's Foot Notation: Example



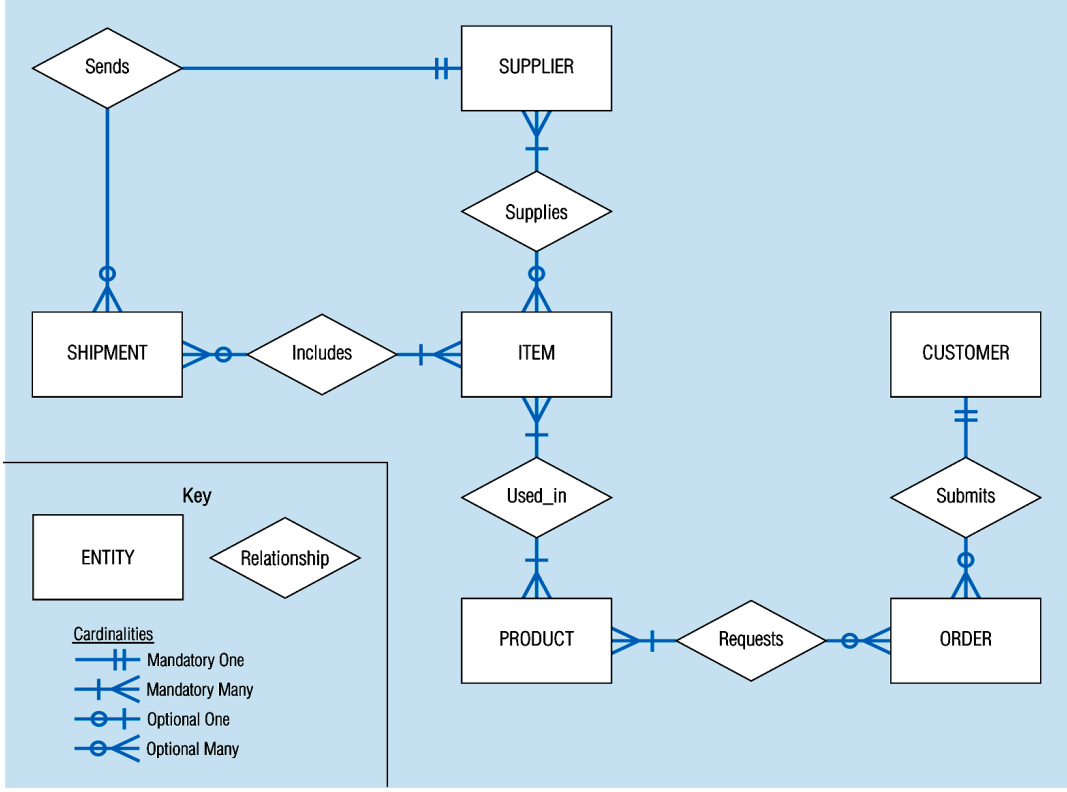
The 1: M relationship between PAINTER and PAINTING

Crow's Foot Notation: Example



The 1:1 Relationship Between PROFESSOR and DEPARTMENT

Crow's Foot Notation: Example



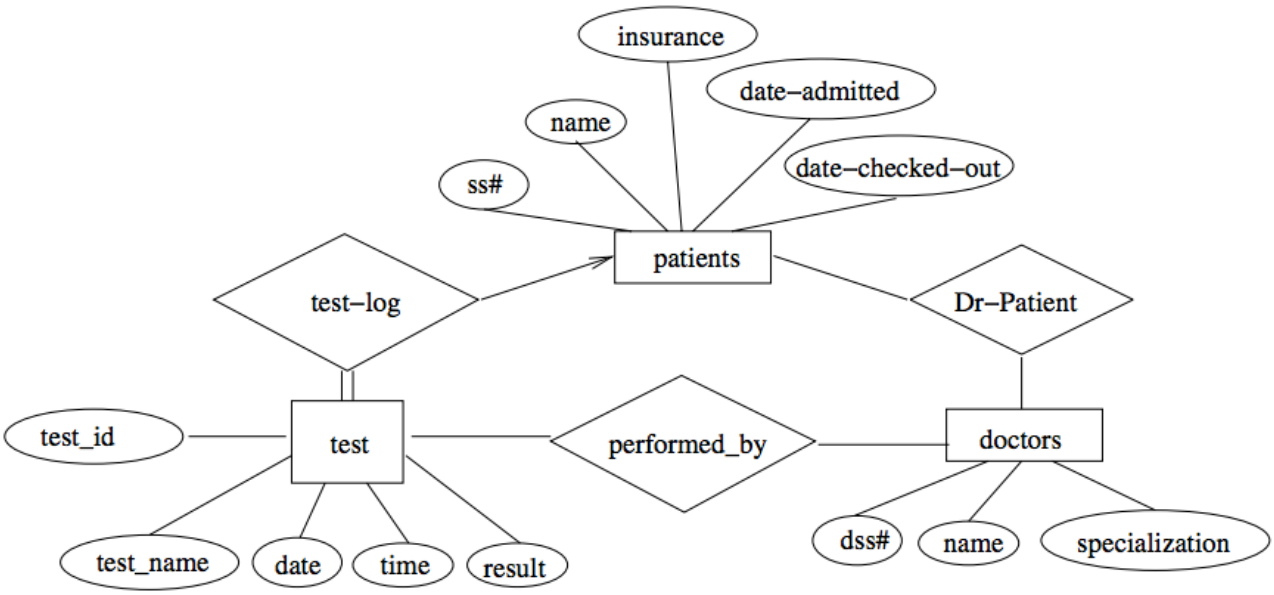
Crow's Foot Notation: Summary

DEPARTMENT DepartmentName BudgetCode OfficeNumber	DEPARTMENT entity; DepartmentName is identifier; BudgetCode and OfficeNumber are attributes.
A B	1:1, nonidentifying relationship. A relates to zero or one B; B relates to exactly one A. Identifier and attributes not shown.
A B	1:N, nonidentifying relationship. A relates to one or many Bs; B relates to zero or one A. Identifier and attributes not shown.
A B	Many-to-many, nonidentifying relationship. A relates to zero or more Bs; B relates to one or more As.
A B	1:N identifying relationship. A relates to zero, one, or many Bs. B relates to exactly one A. Identifier and attributes not shown. For identifying relationships, the child must always relate to exactly one parent. The parent may relate to zero, one, many, or a combination of these minimum cardinalities.
A C D	A is supertype, C and D are exclusive subtypes. Discriminator not shown. Identifier and attributes not shown.
A C D	A is supertype, C and D are inclusive subtypes. Identifier and attributes not shown.

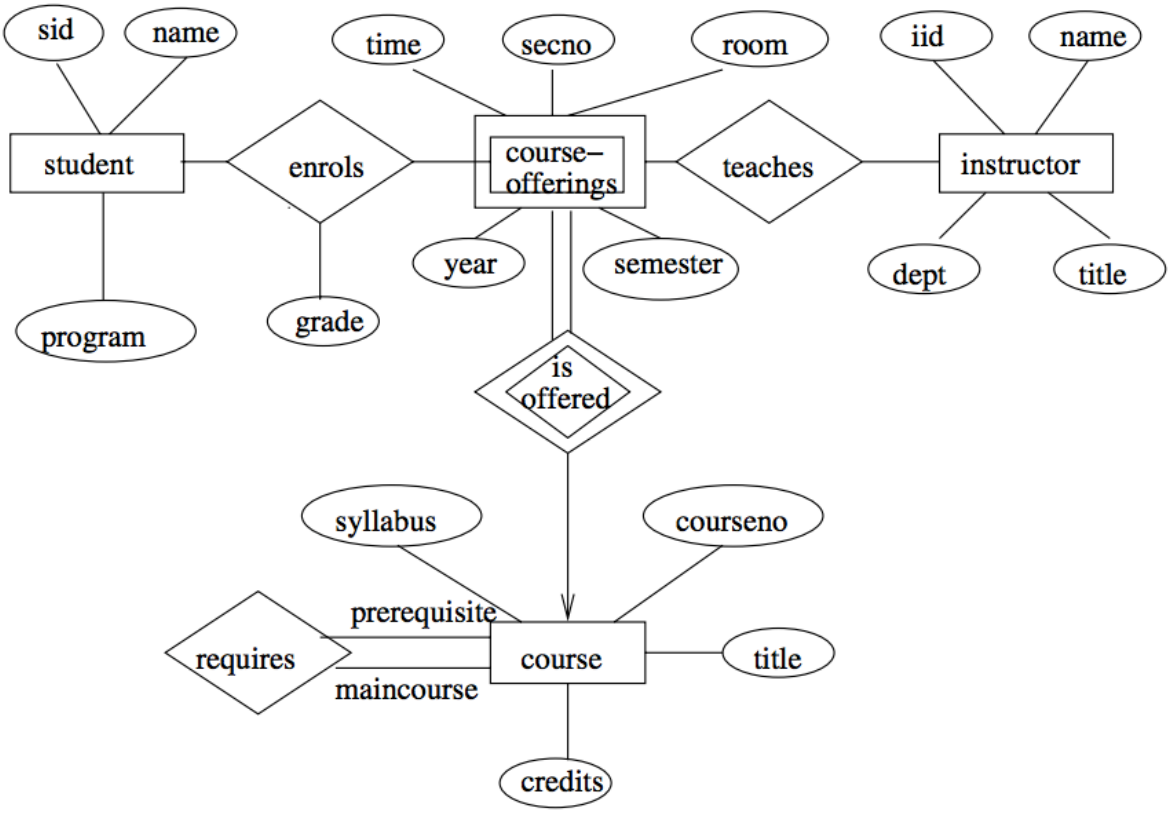
Review (continued)

- 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.

What does this ER Diagram model?

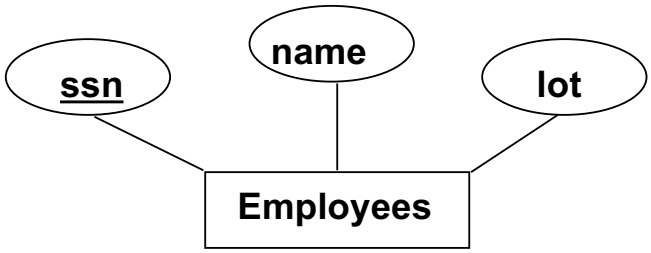


And this?



Logical DB Design: ER to Relational

- Entity sets to tables.




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

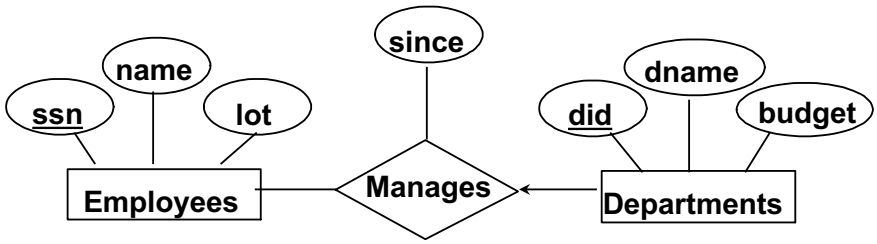
Relationship Sets to Tables

- 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
111-22-3333	51	1/1/91
123-22-3666	56	3/3/93
231-31-5368	51	2/2/92

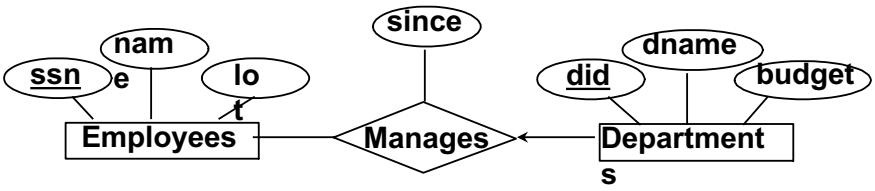
Translating ER with Key Constraints



One way to translate the Manages Relationship:

```
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



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

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


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)
```

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)
```


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.

An Example

- We want to model a simple university database
  - In our database, we have students.
  - They have a name, a registration number, and a course of study.
  - The university offers lectures. Each lecture may be part of some course of study in a certain semester. Lectures may have other lectures as prerequisites.
  - They have a title, provide a specific number of credits and have an unique ID
  - Each year, some of these lectures are offered by a professor at a certain day at a fixed time in a specific room. Students may register for that lecture.
  - Professors have a name and are member of a specific department.

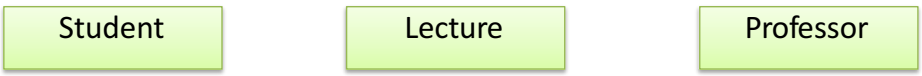
An Example

- How to start? What to do?
  - Find the basic **entity types**
  - Find the **attributes** of entities
    - Decide to which entity an attribute should be assigned
    - Which attributes are key attributes?
    - Some attributes are better modeled as own entities, which ones?
  - Define the **relationship types**
    - Which role do entities play?
    - Do relationships require additional entity types?
    - Are the relationships total? Identifying? Are weak entities involved?
    - What are the cardinalities of the relationship type?

An Example

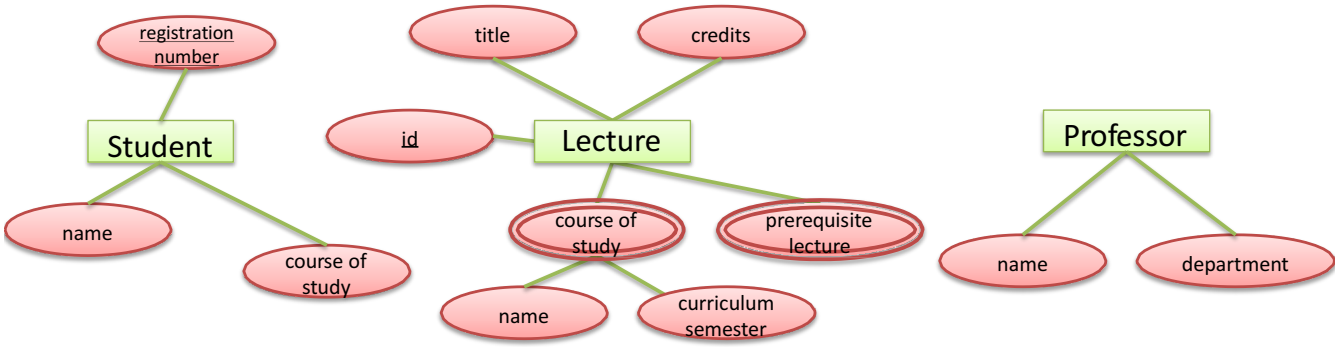
- Which are our entity types?
  - In our database, we have students. They have a name, a registration number and a course of study.
  - The university offers lectures. Each lecture may be part of some course of study in a certain semester. Lectures may have other lectures as prerequisites. They have a title, provide a specific number of credits and have unique ID
  - Each year, some of these lectures are offered by a professor at a certain day at a fixed time in a specific room. Students may attend that lecture.
  - Professors have a name and are member of a specific department.

An Example



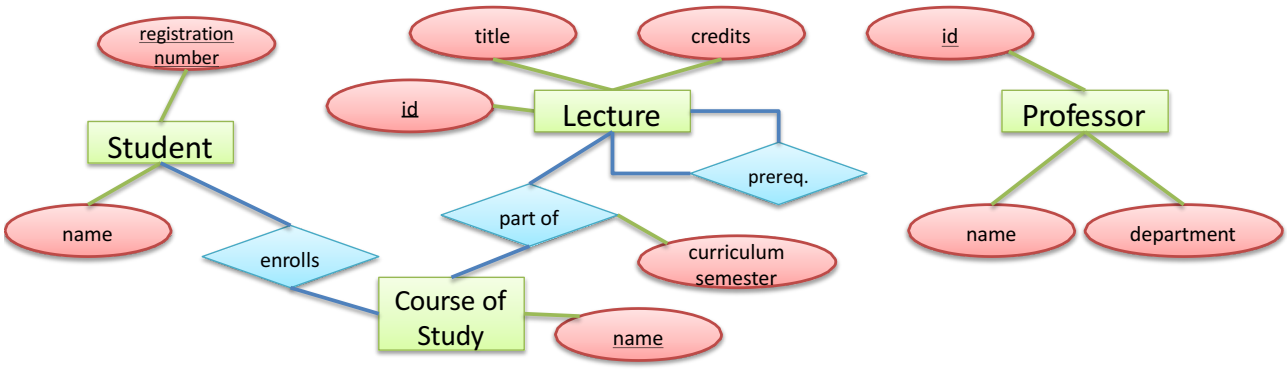
- What attributes are there?
  - In our database, we have students. They have a name, a registration number and a course of study.
  - The university offers lectures. Each lecture may be part of some course of study in a certain semester. Lectures may have other lectures as prerequisites. They have a title, provide a specific number of credits and have unique ID
  - Professors have a name and are member of a specific department.

An Example



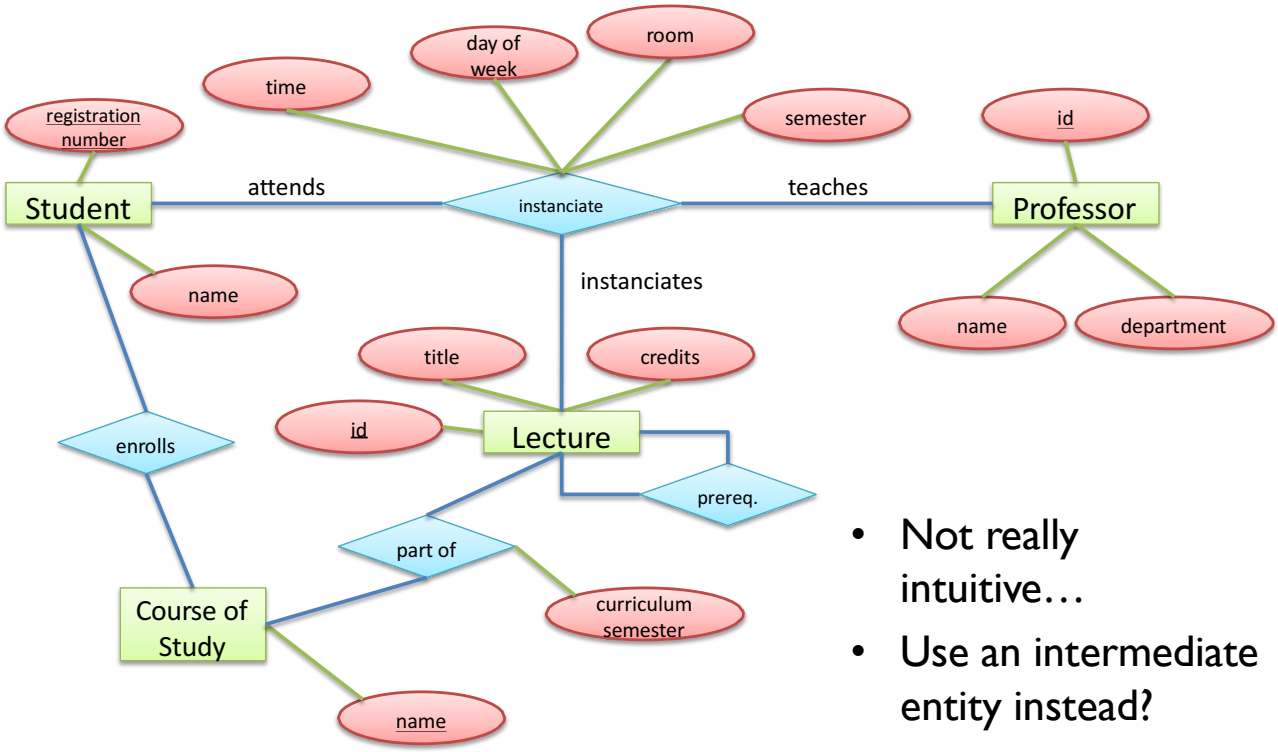
- First try…
  - This model is **really crappy**!
  - "Course of study" does not seem to be an attribute
    - Used by student and lecture. Even worse, lecture refers to a course of study in a specific curriculum semester.
    - Use additional entity type with relationships!
  - "Prerequisite lecture" also is not a good attribute
    - Prerequisite lectures are also lectures. Use a relationship instead!
  - "Professor" does not have key attributes

An Example



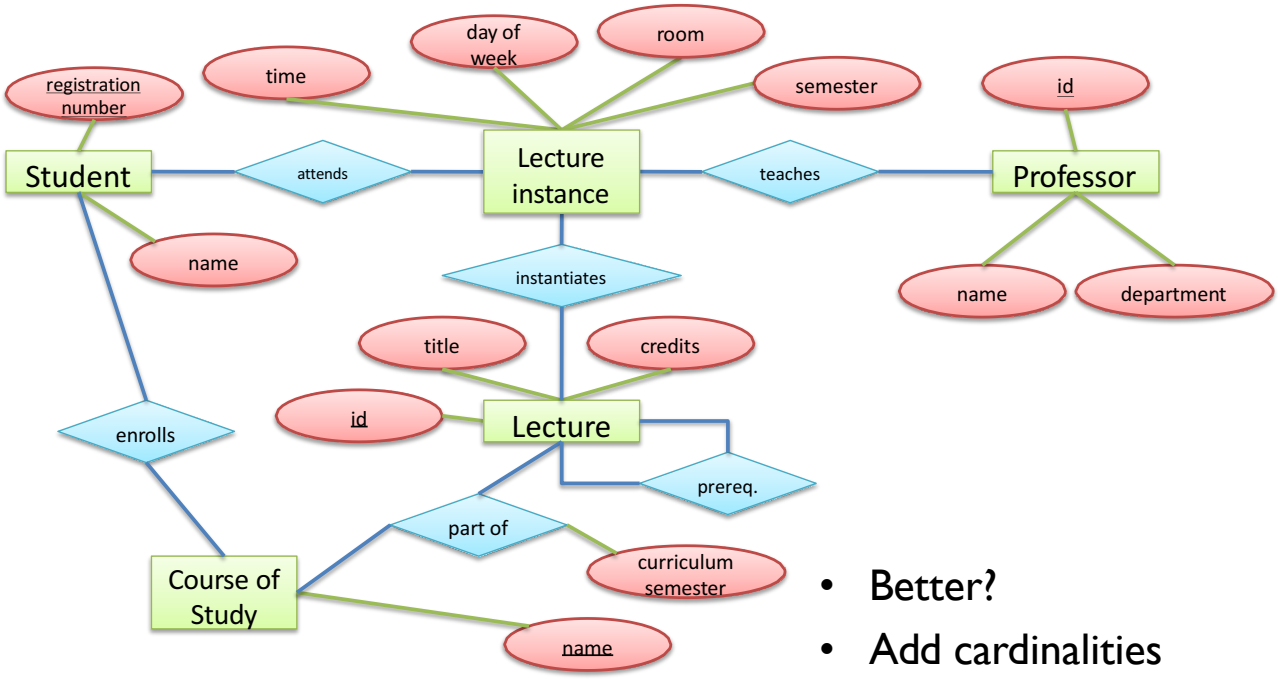
- Second try
  - Professors use a **surrogate** key now
    - Key is automatically generated and has no meaning beside unique identification
  - Course of study is an entity type now
- Which entity types are additionally related?
  - "Each year, some lectures of the pool of all lectures are offered by a professor at a certain day at a fixed time in a specific room. Students may attend that lecture."

An Example



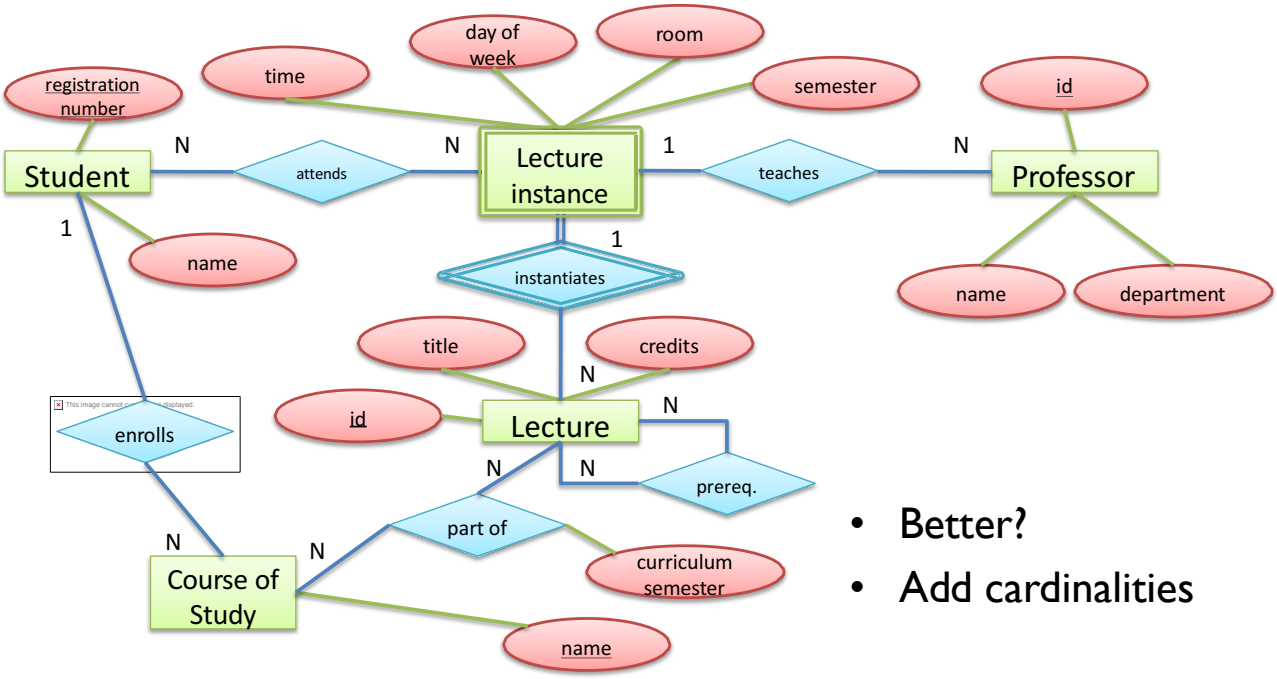
- Not really intuitive…
- Use an intermediate entity instead?

An Example



- Better?
- Add cardinalities
- Add total and identifying annotations

An Example



- Better?
- Add cardinalities

An Example

- Modeling is not that simple
- Many possible (and also correct) ways of modeling the same miniworld
  - Some are more elegant, some are less elegant
- Models alone are not enough, they need to be documented
  - What are the meanings of the attributes? The meanings of the relationships?

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

ER & Relational Model: Summary

- ER is a high-level model that is typically not directly implemented but is "user-friendly"
- Relational Model: A tabular representation of data.
- Simple and intuitive, currently the most widely used
  - Object-relational and XML extensions in most products
- Integrity constraints
  - Specified by the DB designer to capture application semantics.
  - DBMS prevents violations.
  - Some important ICs:
    - primary and foreign keys
    - Domain constraints
- Powerful query languages:
  - SQL is the standard commercial one
    - DDL - Data Definition Language
    - DML - Data Manipulation Language

Summary

- 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.