Logical Database Design I

Entity-Relation Models

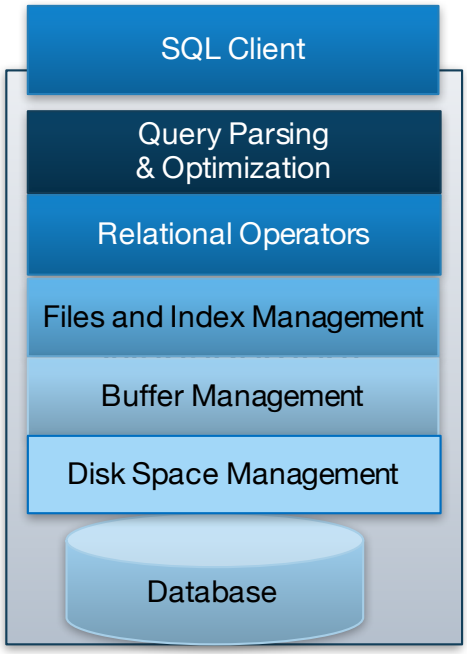
Alvin Cheung

Fall 2022

Reading: R&G Chapter 2

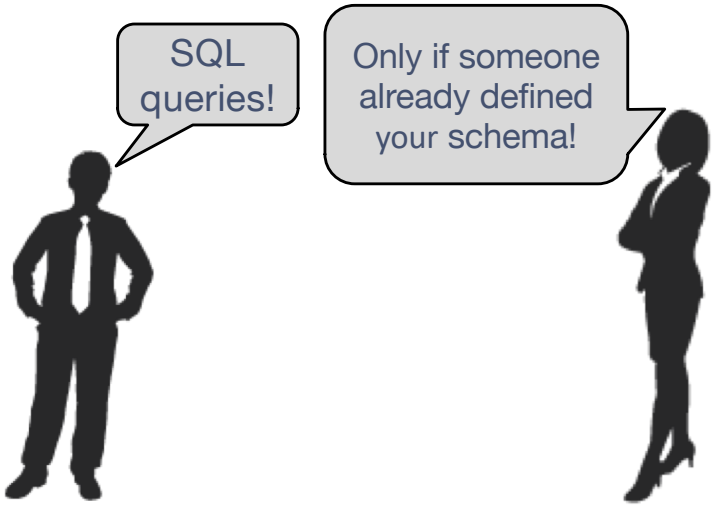
Architecture of a DBMS

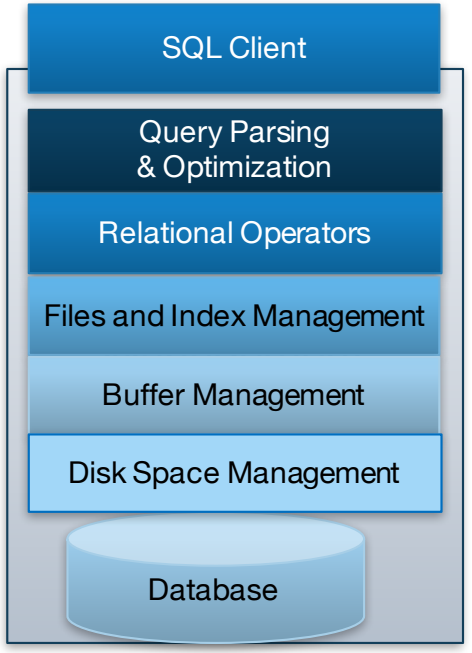
- Gives us a good sense of how to build a DBMS
- How about using one?



Architecture of a DBMS, Pt 2

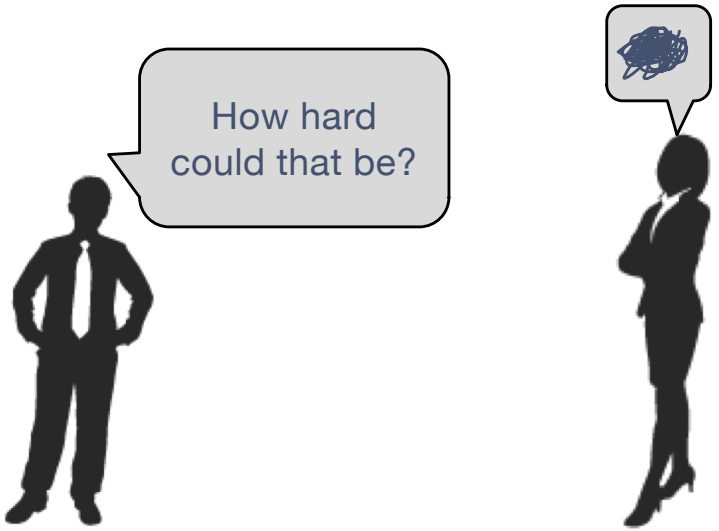
- Gives us a good sense of how to build a DBMS
- How about using one?

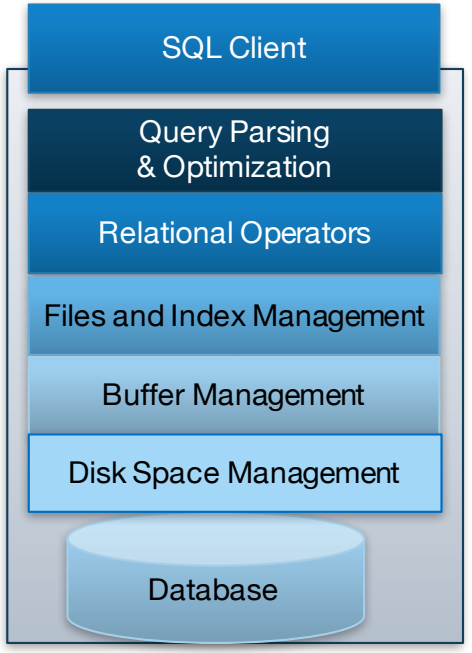




Architecture of a DBMS, Pt 3

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

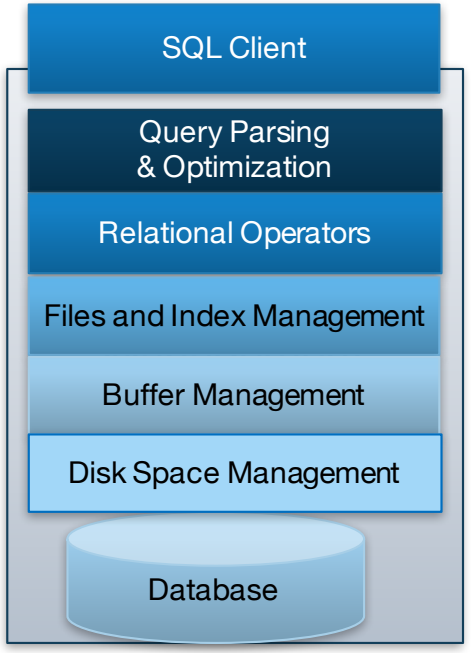




Design of a Database

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

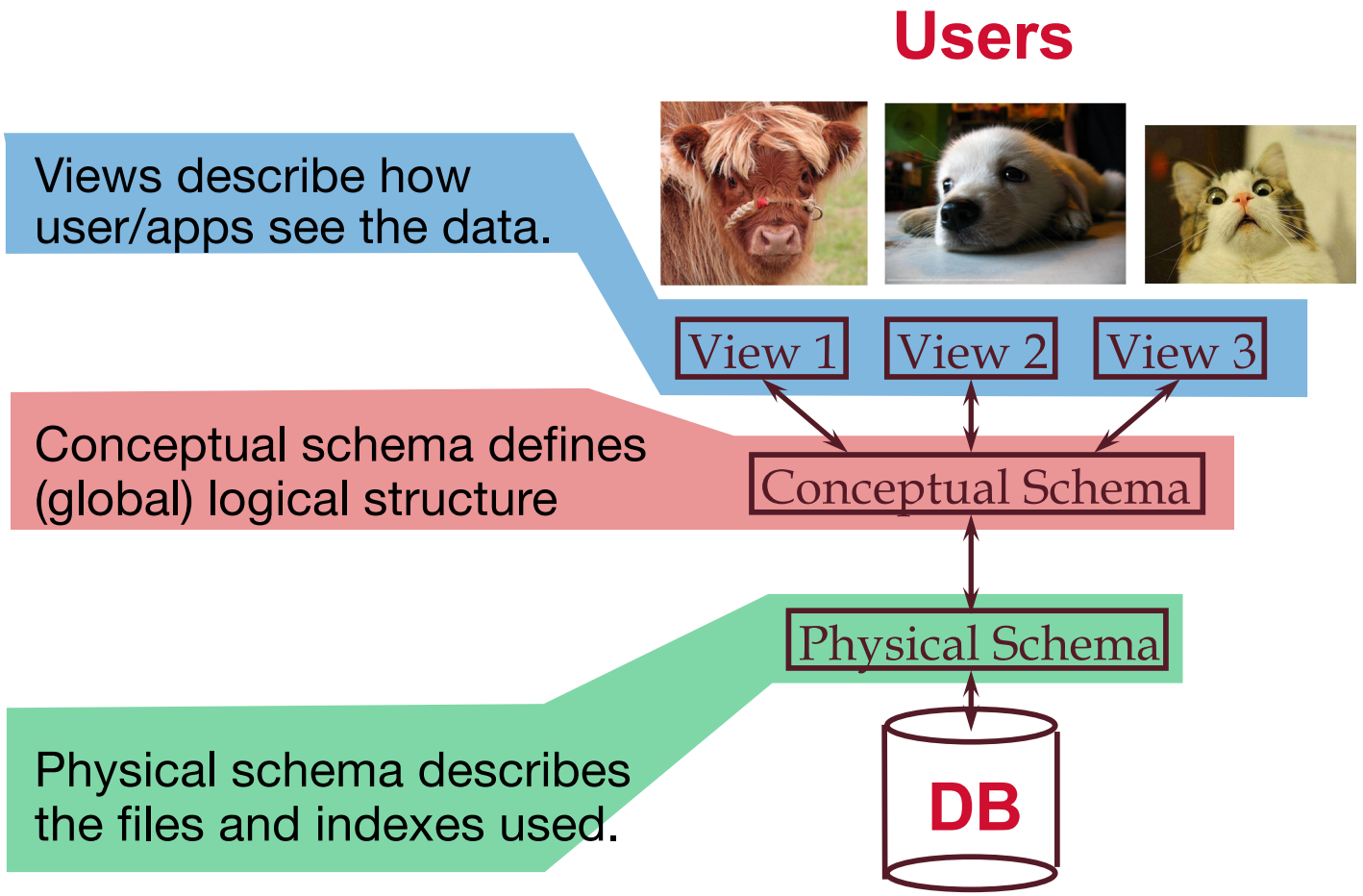
- Today let's talk about how to design a database
  - Not a database system
  - Let's start with what we know… data models



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: Various Schemas

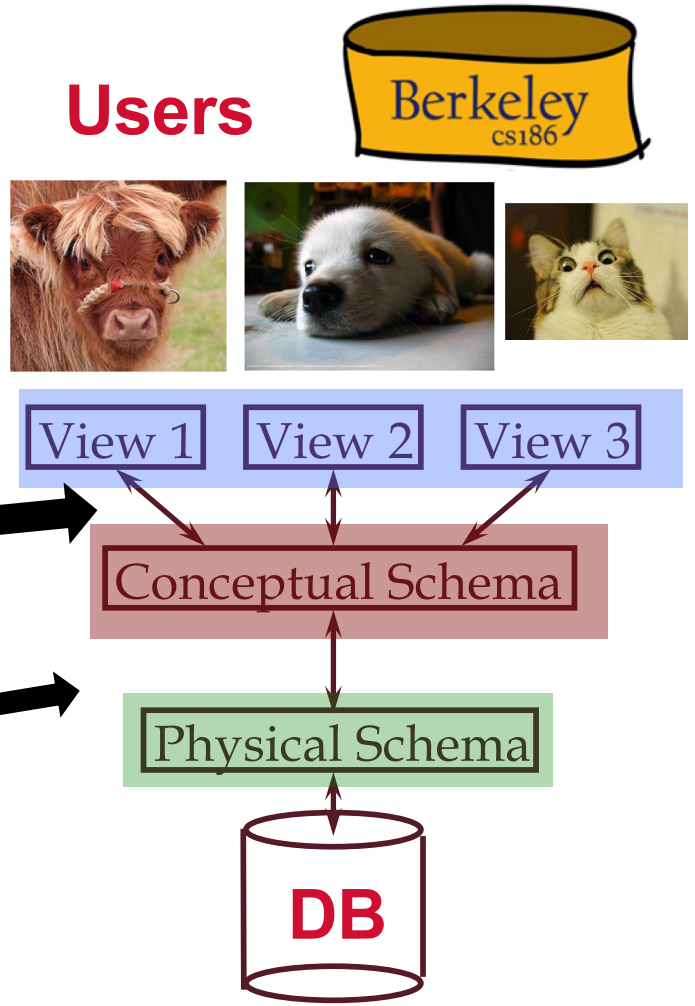


Example: University Database

- **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/View schema**:
  - `Course_info (cid text, enrollment integer)`
    - Group by query on `Enrolled`

Data Independence

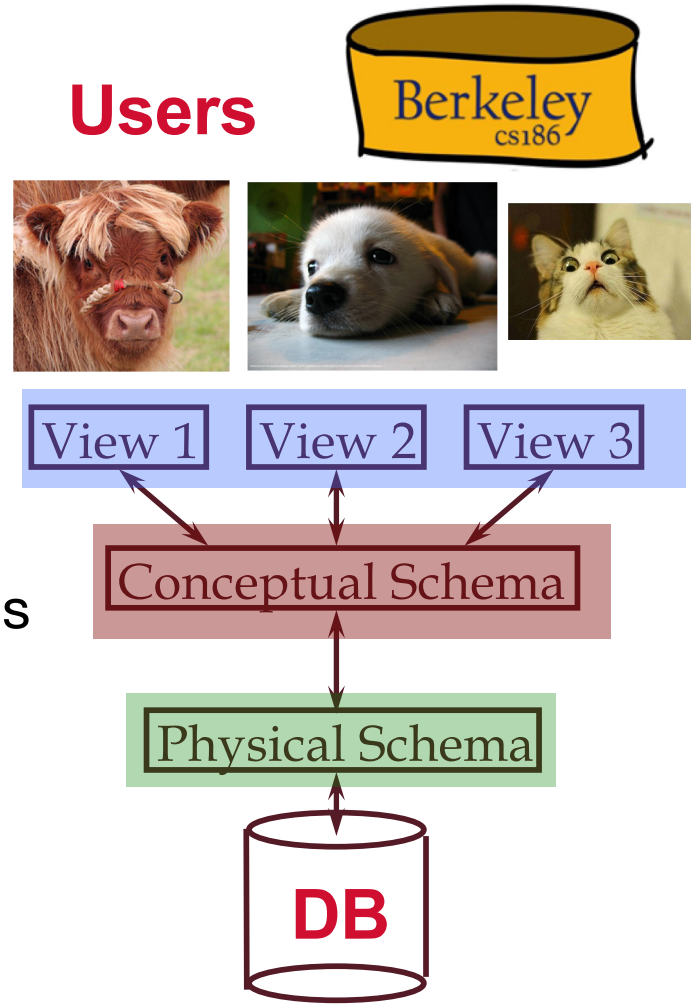
- Insulate apps from structure of data
- **Logical data independence:**
  - Maintain views when logical structure [schema] changes
  - E.g., if we split a relation into two, can simply change view query, apps that use views don't need to be rewritten
- **Physical data independence:**
  - Maintain logical structure when physical structure changes
  - E.g., if we add an index, queries to conceptual schema don't need to be rewritten



Data Independence

- Insulate apps from structure of data

- **Logical data independence:**
- **Physical data independence:**
- Q: Why important for DBMS?
- Because databases and their associated apps persist over long periods of time!
  - Applications, hardware may change, …
  - E.g., Many banks are still running old database systems
  - Modularity is key



Data Models

- Relational Model:
  - A collection of relations
  - Easy to implement in a database
  - Easy to provide both logical and physical data independence
  - Harder to reason about
- Today: Entity-Relational (ER) Model
  - A collection of entities and relations
  - Harder to implement in a database directly
  - But easier to reason about
- So we will talk about the ER Model and how to translate it into the Relational Model
- Let's talk about the workflow of database design…

Many Steps in Database Design!

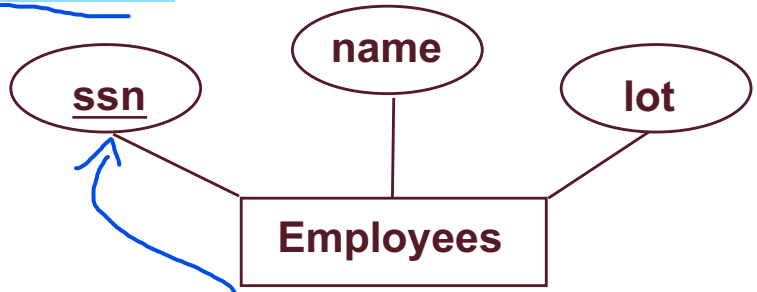
- **Requirements Analysis**
  - Translating user needs; what must database do? what must it capture?
- **Conceptual Design [Needs => ER Diagram]**
  - *high level visual description of data(often done w/ER model)*
  - Object-Relational Mappings (ORMs: Hibernate, Rails, Django, etc) encourage you to program here [essentially ER models]
- **Logical Design [ER Diagram => Relations]**
  - translate ER into DBMS data model
  - ORMs often require you to help here too
  - can be partially automated
- **Schema Refinement [Relations => Better Relations]**
  - consistency, normalization [add constraints, break or merge relations]
- **Physical Design [Storing Relations]**
  - indexes, disk layout

- **Orthogonal: Security Design [Relational Access Control]**
  - who accesses what, and how


We are here

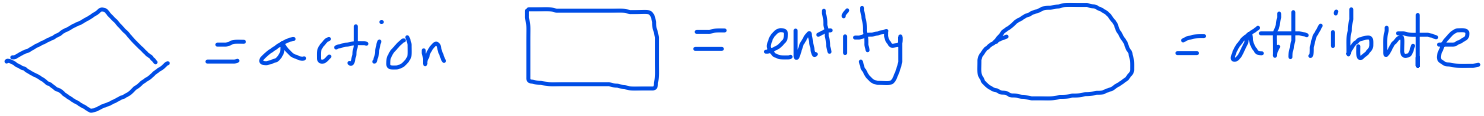
ER Model Basics: Entities

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

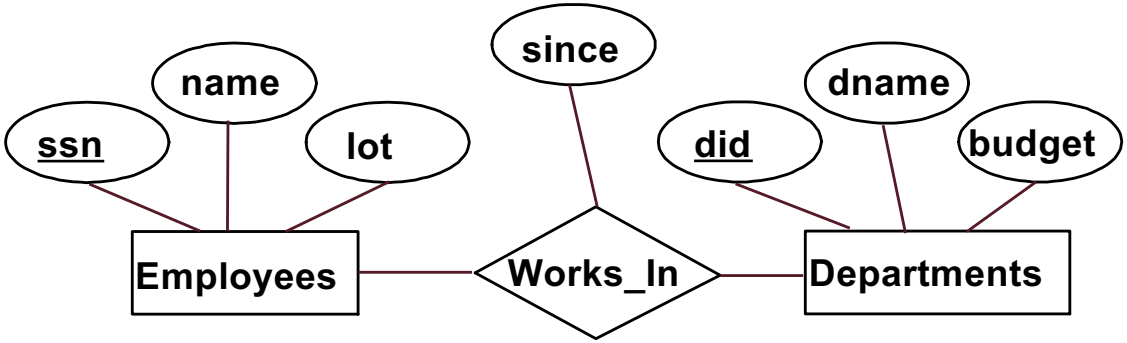


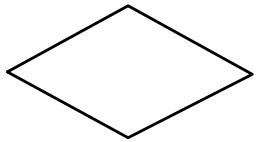
- **Entity Set**: A collection of similar entities.
  - E.g., all employees.
  - All entities in an entity set have the same attributes.
  - Each entity set has a primary key (underlined)
  - Each attribute has a domain

unique



ER Model Basics: Relationships





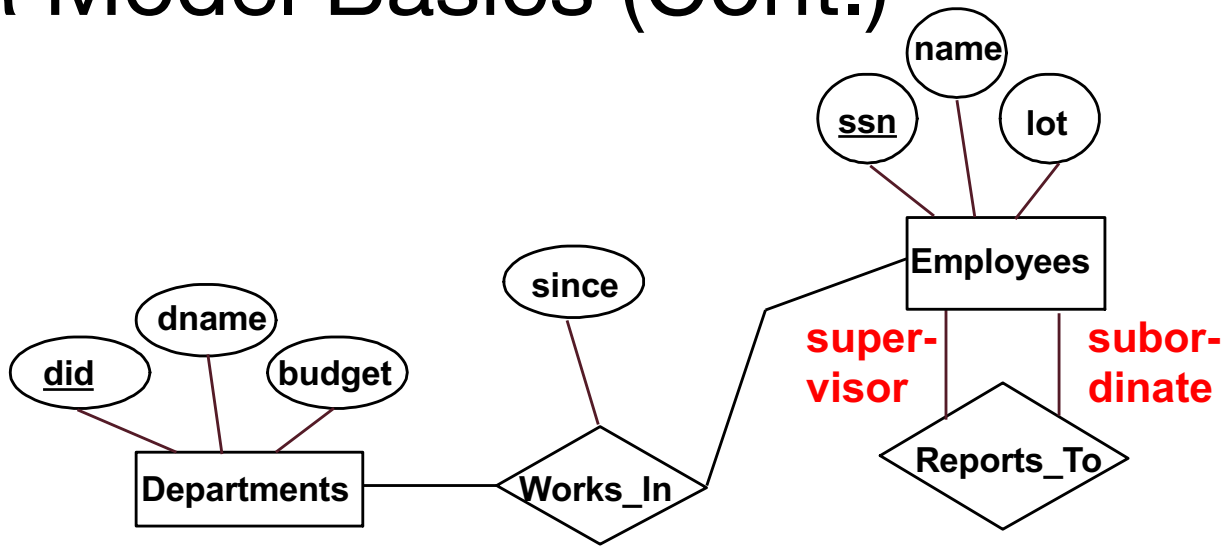
Relationship: Association among two or more entities.

- E.g., Jenny (employee) 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 ... E_n; each relationship in R involves entities $e_1 \in E_1$, ..., $e_n \in E_n$

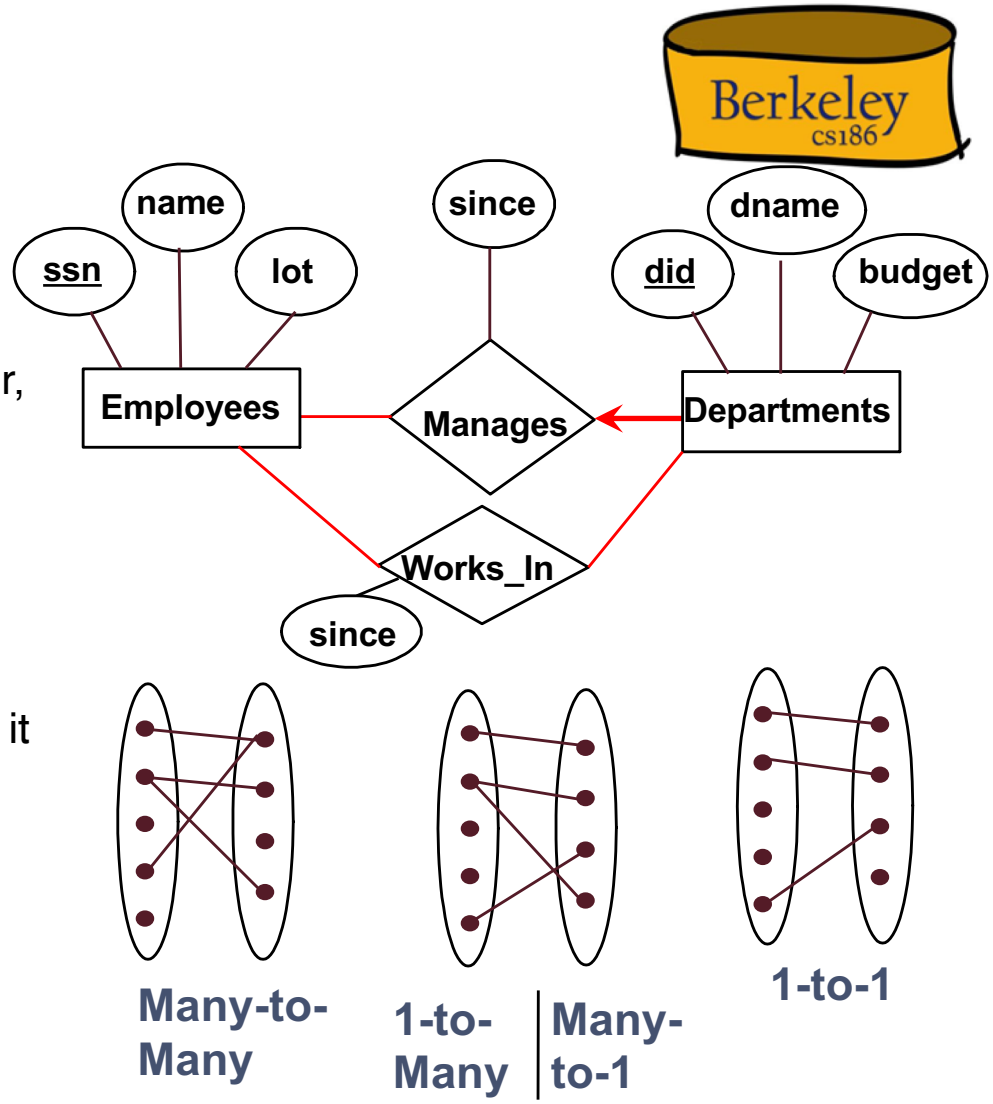
ER Model Basics (Cont.)



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

Key Constraints

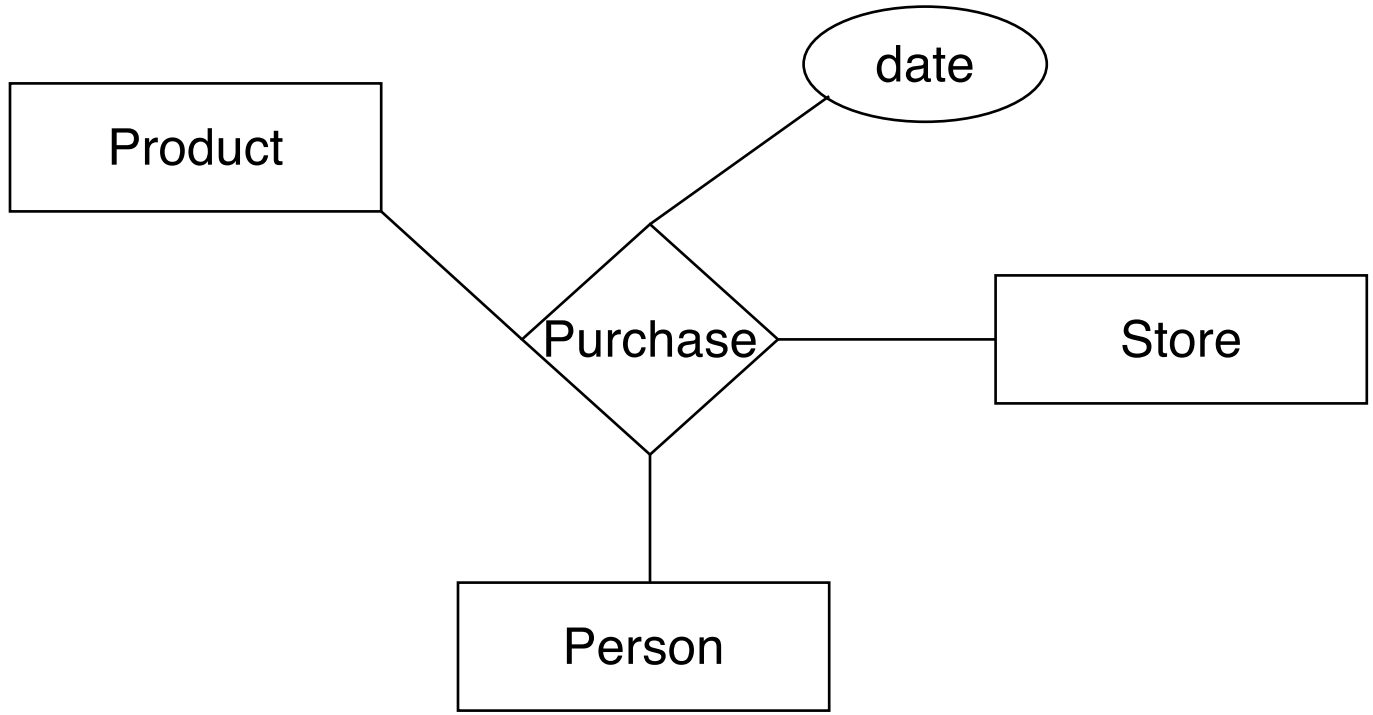
- An employee can work in **many** departments; a dept can have **many** employees.
- In contrast, each dept has **at most one** manager, according to the *key constraint* on **Dept** in the **Manages** relationship set. Equivalently:
  - Each dept participates at most once in this relationship
  - Each dept has at most one emp. managing it
- A **key constraint** gives a 1-to-many/many-to-1 relationship.

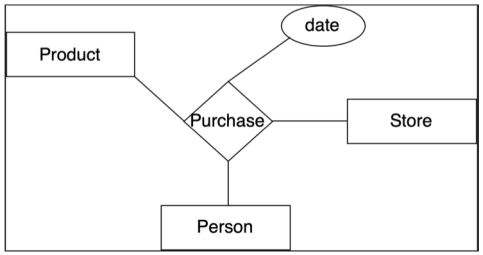


Some ER Modeling Tools Require 2-way Relationships

Do we need multi-way relationships or do 2-way (binary) relationships suffice?

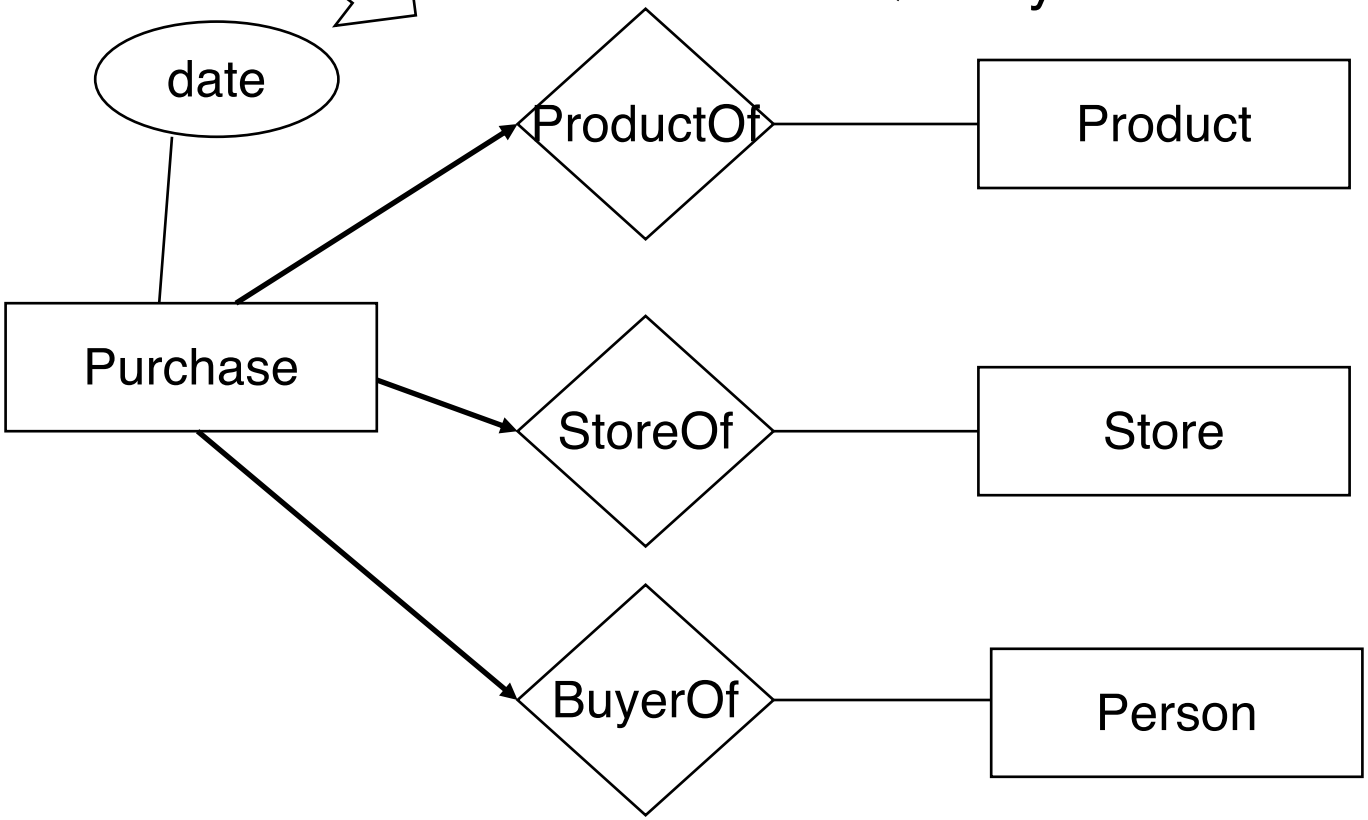
How would we convert this into binary?





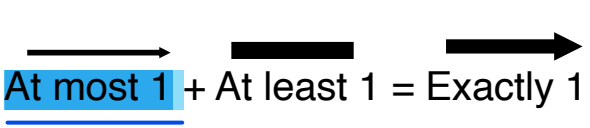
Converting Multiway Relationships to Binary

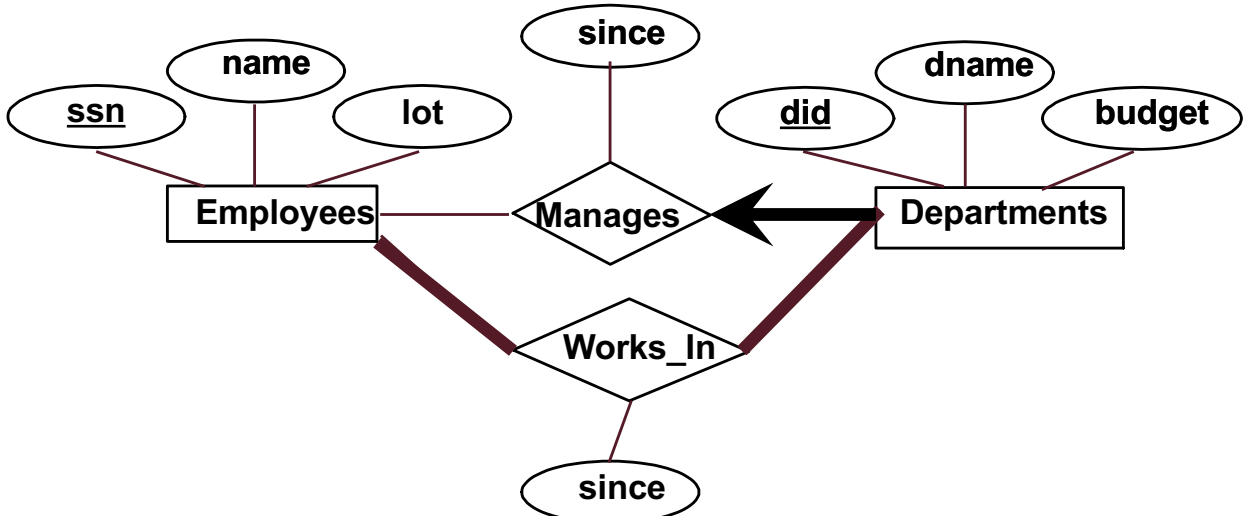
Q: Why the arrows?



Participation Constraints

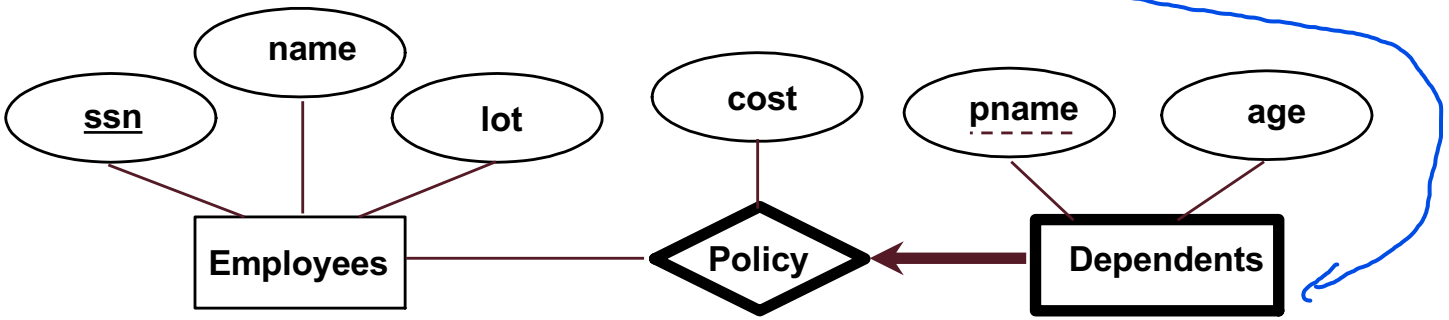
- Does every employee work in a department?
  - If so: a **participation constraint**
  - participation of Employees in Works_In is **total** (vs. partial)
  - Basically means that every employee participates in "**at least one**".
- Likewise, what if every department has an employee working in it?
- Likewise, what if every department has a manager?
  - Along with the arrow (at most one), this means exactly one





Weak Entities

- A **weak entity** can be identified uniquely only by considering the primary key of other (owner) entities.
  - Owner entity set(s) and weak entity set must participate in a one-to-many relationship set (one owner, many weak entities)
    - Each empl. can have multiple dep. Each dep is associated w. at most 1 empl.
  - Weak entity set must have total participation in this relationship set.
    - Each dep is associated with at least one empl.



- Weak entities have only a "partial key" (dashed underline) [pname] (+ [ssn] = full key)

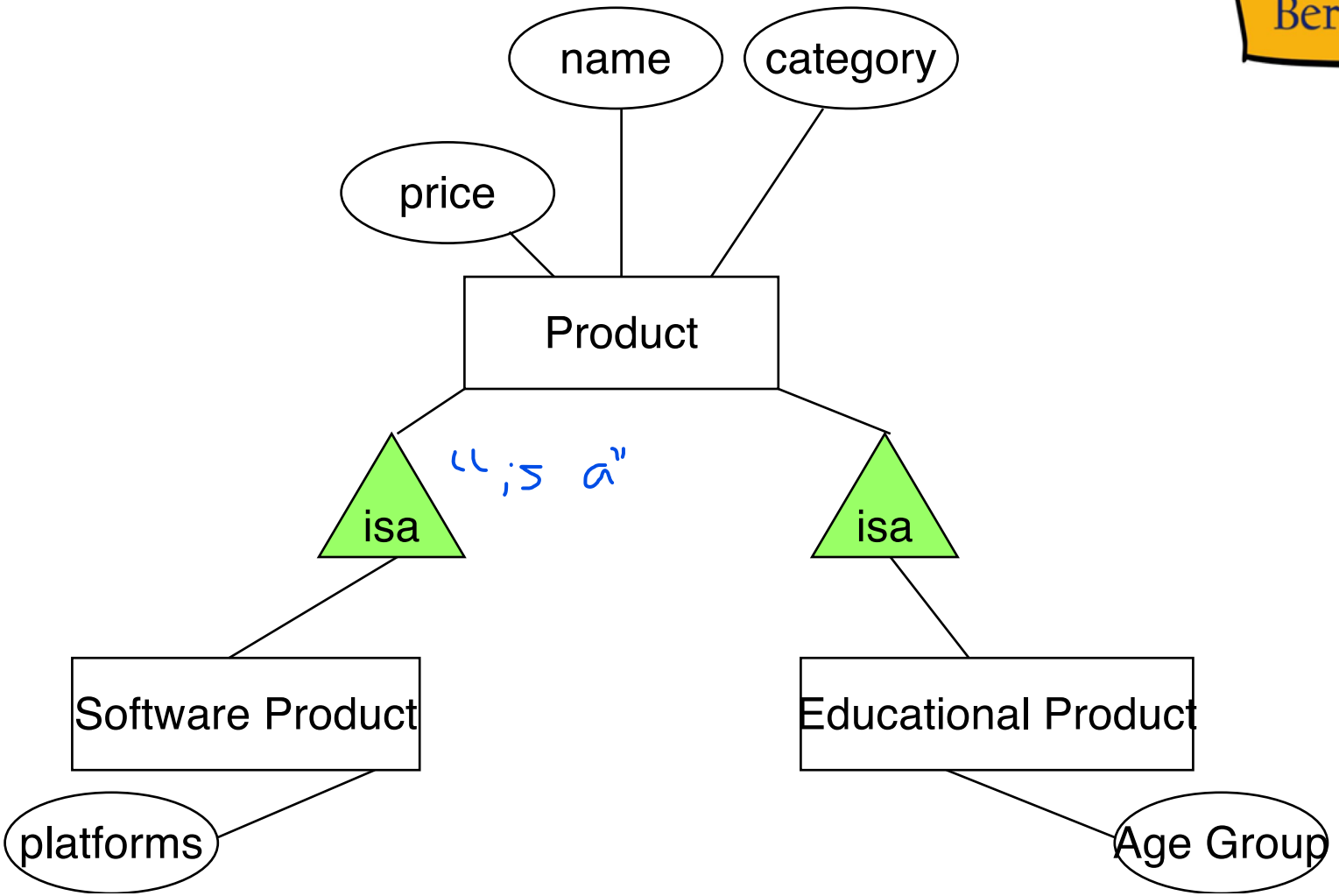
Recap: ER Diagrams – Constraints

- Key constraint
  - at most one
- Participation constraint
  - at least one
- Key constraint with total participation
  - exactly one
- Non-key partial participation
  - 0 or more (no restrictions)

Relationships: Summary

- Modeled as a mathematical set
- Binary and multi-way relationships
- Converting a multi-way one into many binary ones
- Constraints on the degree of the relationship
  - many-one, one-one, many-many
  - participation constraints
  - limitations of arrows
- Attributes of relationships
  - not necessary, but useful
- Weak entity sets & supporting relationships

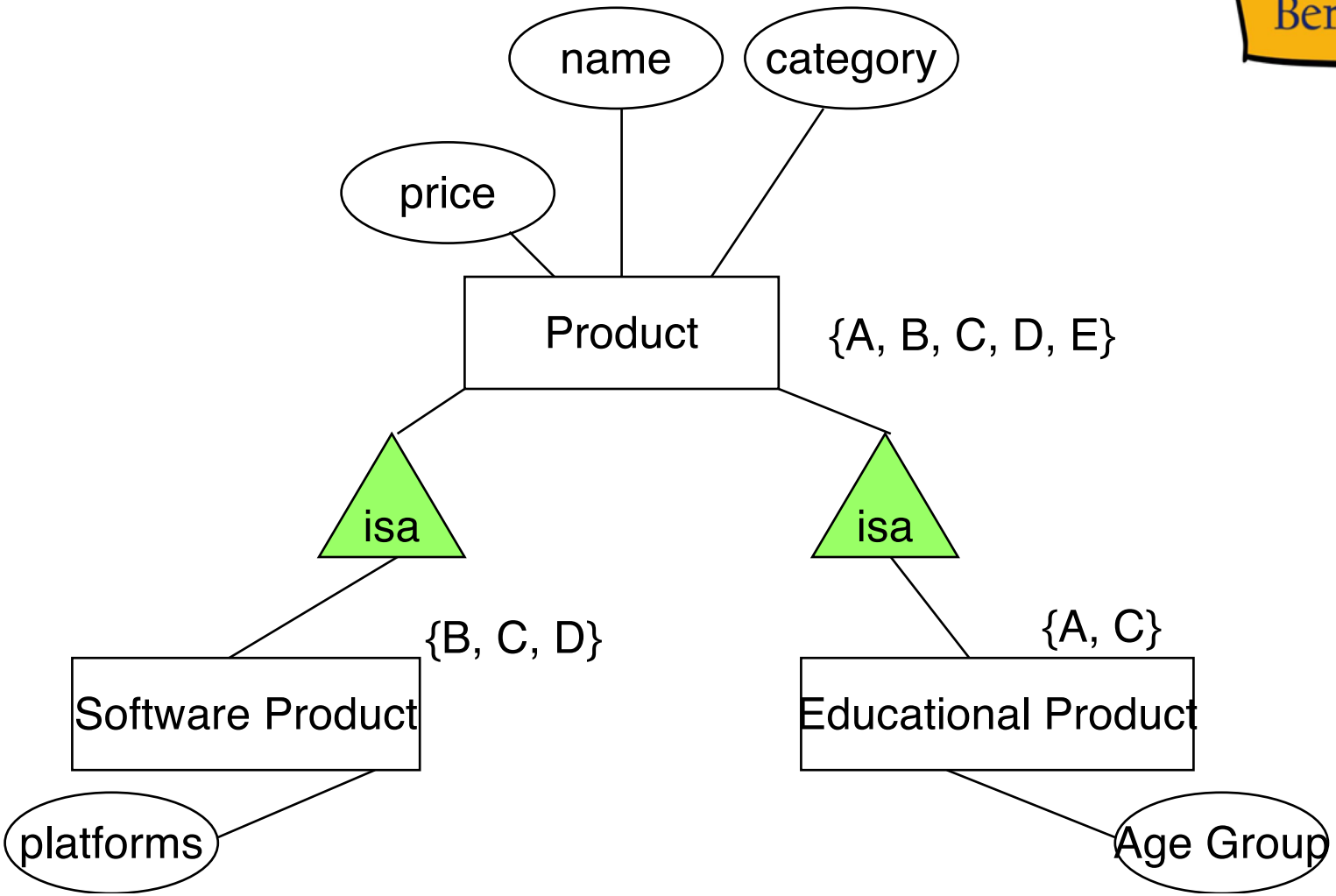
Subclasses in ER Diagrams



Subclasses

- "Isa" triangles indicate the subclass relationship.
  - Point to the superclass.
- Subclasses form a tree.
  - I.e., no "multiple inheritance".
- Why subclasses?
  - Unnecessary to add redundant properties to the root entity set that don't apply to many of the entities

Subclasses in ER Diagrams



Conceptual Design Using the ER Model

- ER modeling can get tricky!
- Design choices:
  - **Entity** or **attribute**?
  - **Entity** or **relationship**?
  - Relationships: **Binary or ternary?**
- 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

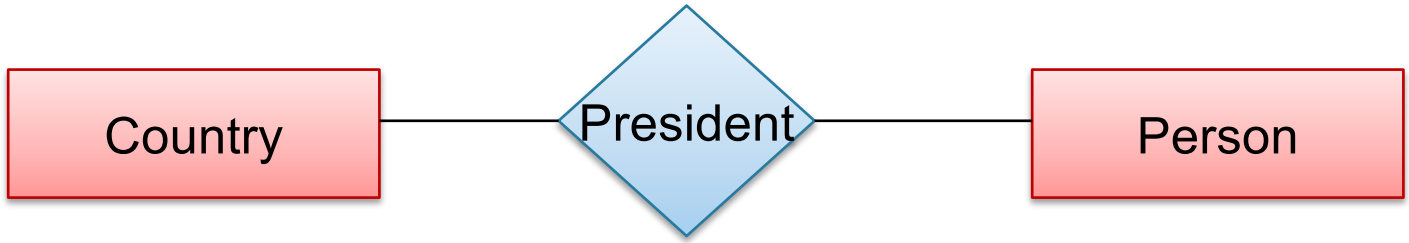
E-R Diagram as Wallpaper

- Very common for them to be wall-sized

Translating ER diagrams to relations

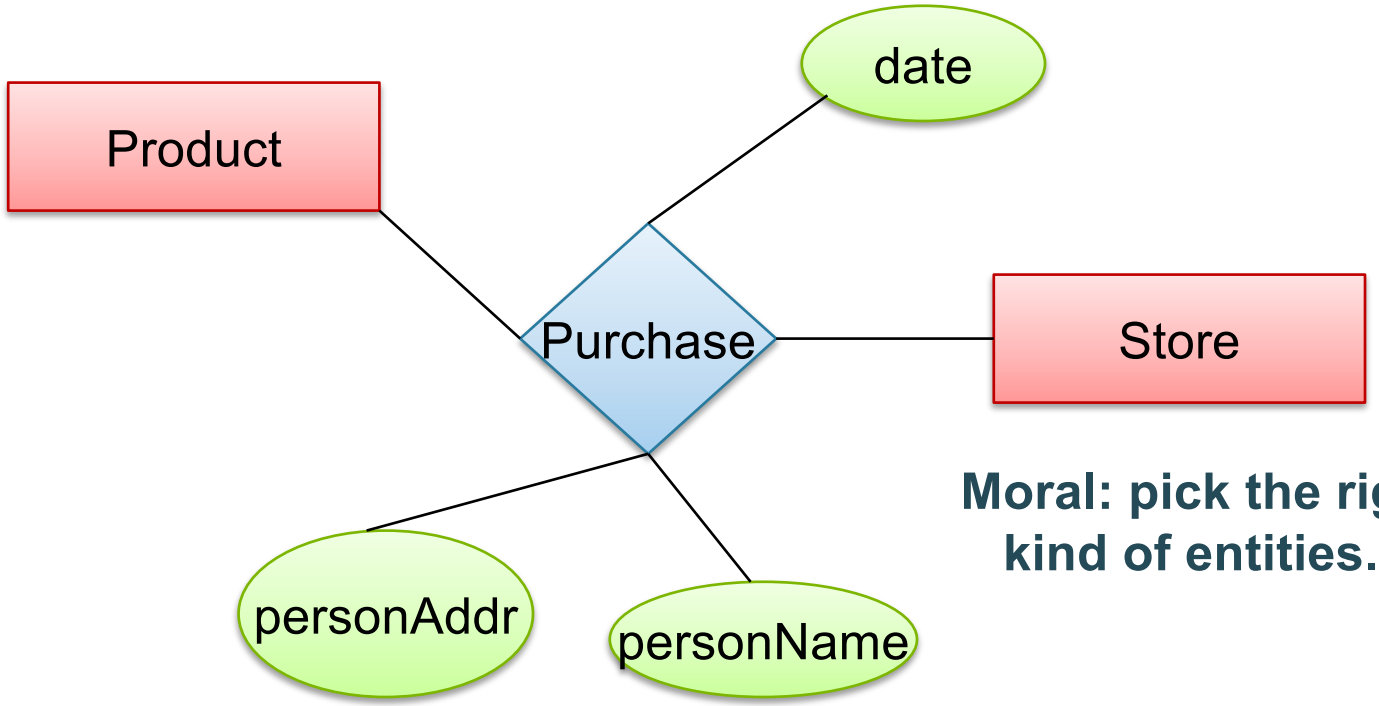
Design Principles

What's wrong?



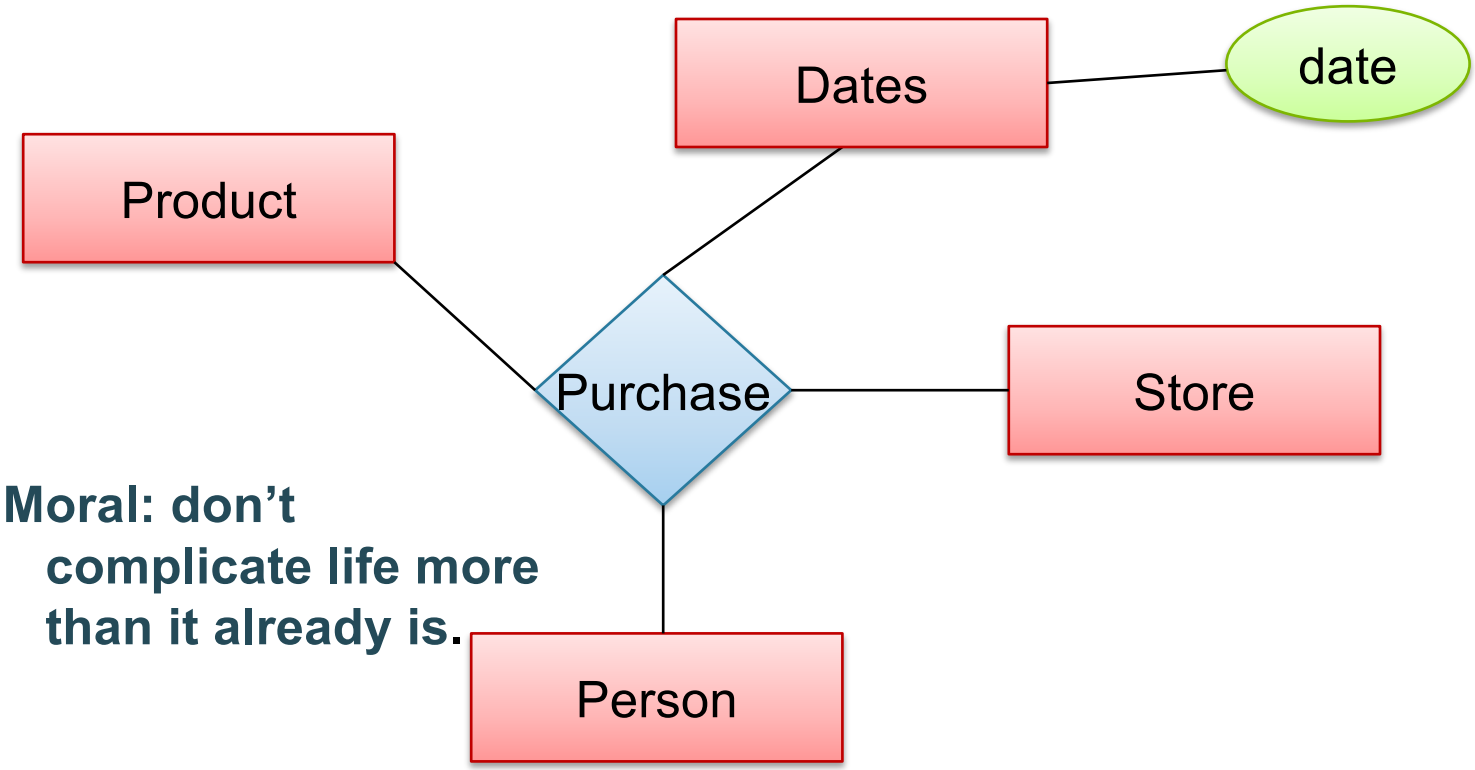
Moral: Be faithful to the specifications of the application!

Design Principles: What's Wrong?



Moral: pick the right kind of entities.

Design Principles: What's Wrong?



Moral: don't complicate life more than it already is.

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

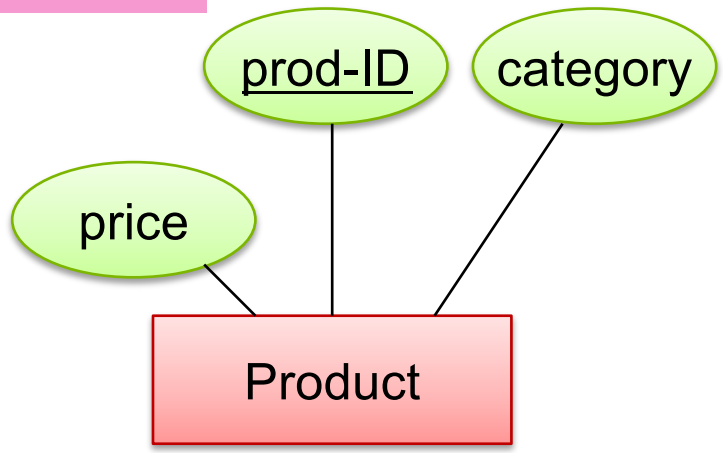
We are here

Ruby on Rails

Converting ER to Relations

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

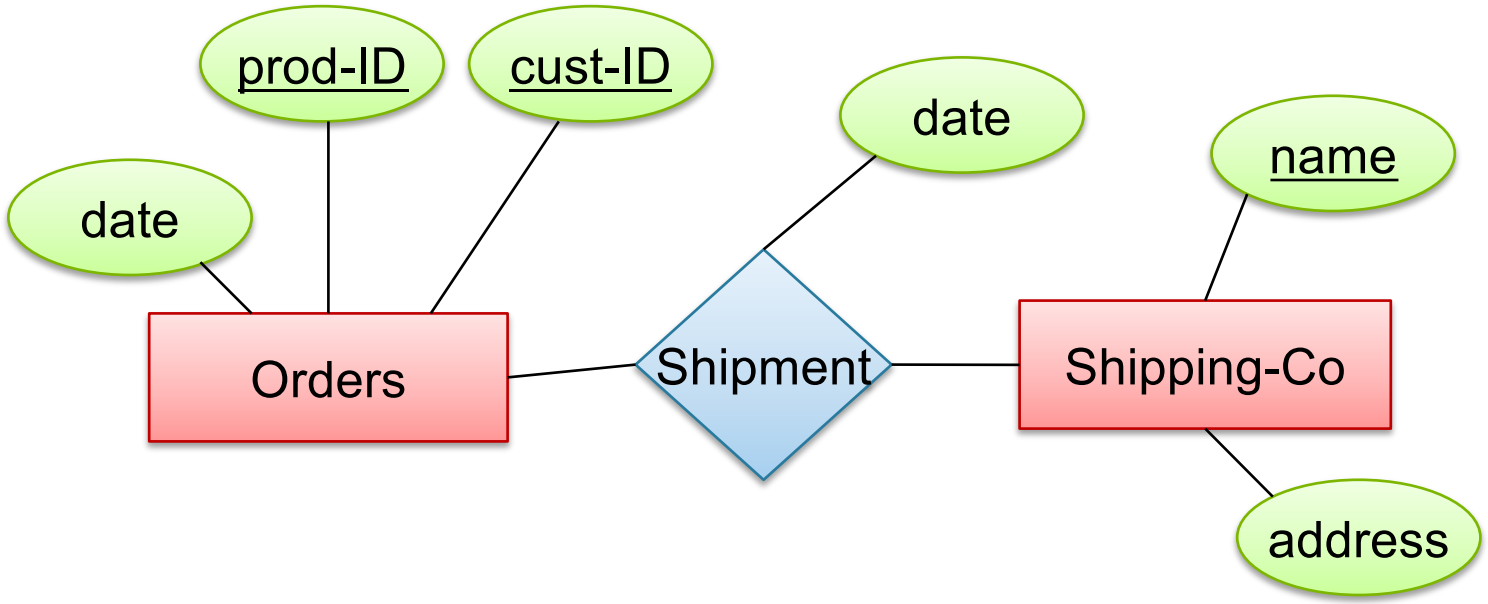
Entity Set to Relation



Product(prod-ID, category, price)

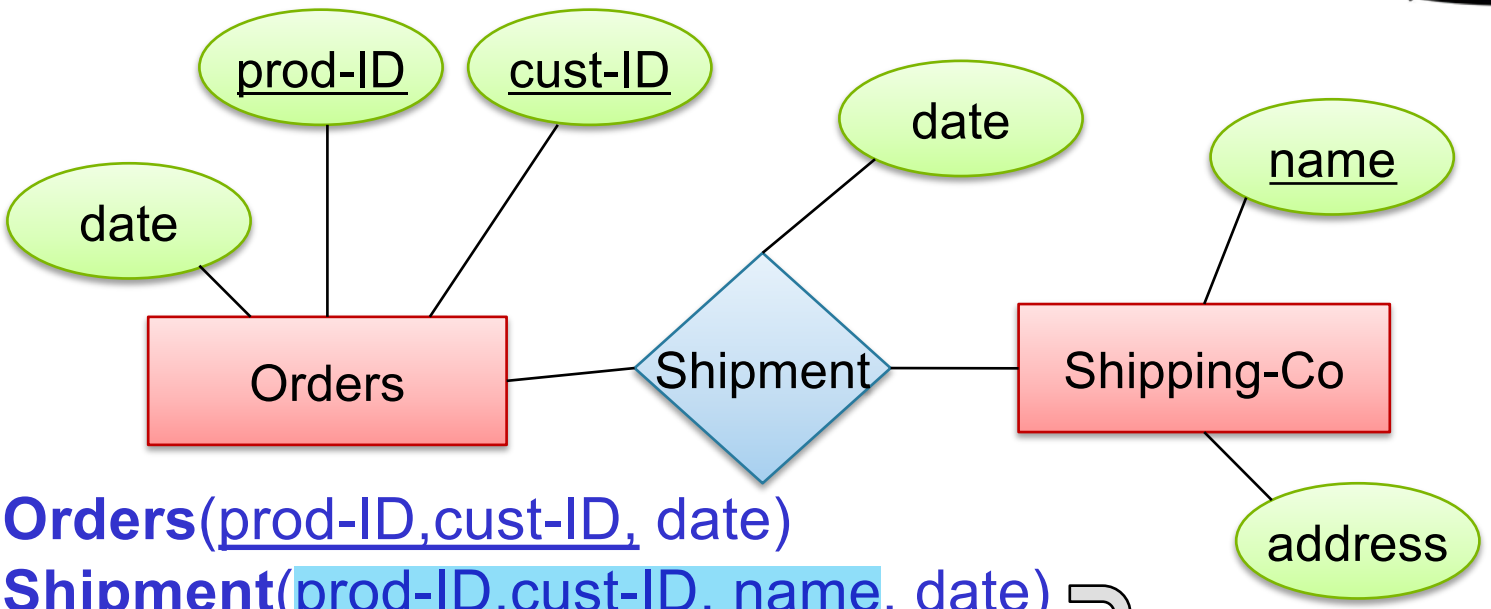
prod-ID	category	price
Gizmo55	Camera	99.99
Pokemn19	Toy	29.99

N-N Relationships to Relations



Represent this in relations

N-N Relationships to Relations



Orders(prod-ID,cust-ID, date)

Shipment(prod-ID,cust-ID, name, date)

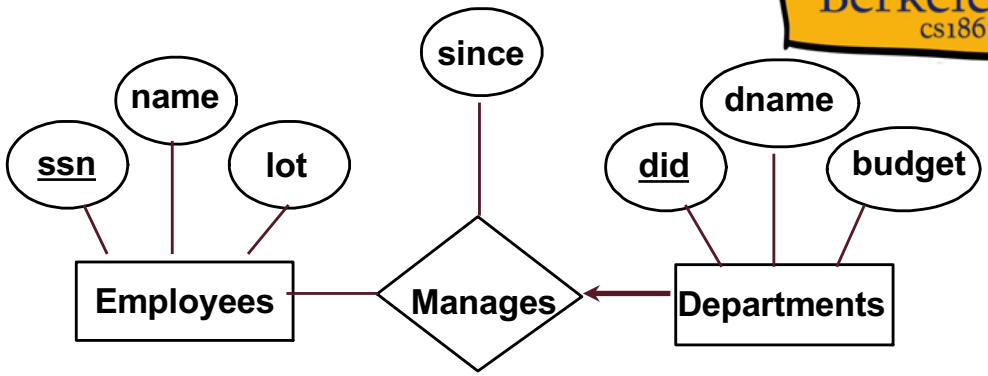
Shipping-Co(name, address)

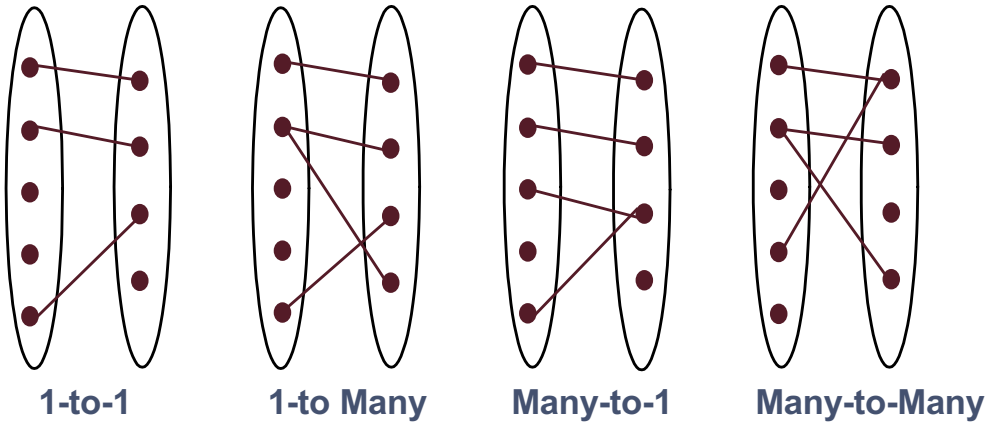
prod-ID	cust-ID	name	date
Gizmo55	Joe12	UPS	4/10/2011
Gizmo55	Joe12	FEDEX	4/9/2011

Note that keys from each participating entity set appear as foreign keys. This set of attributes forms a ***superkey*** for the relation.

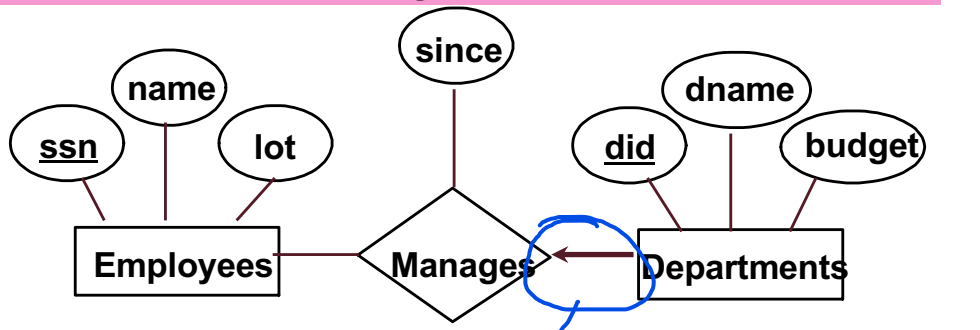
Review: Key Constraints

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



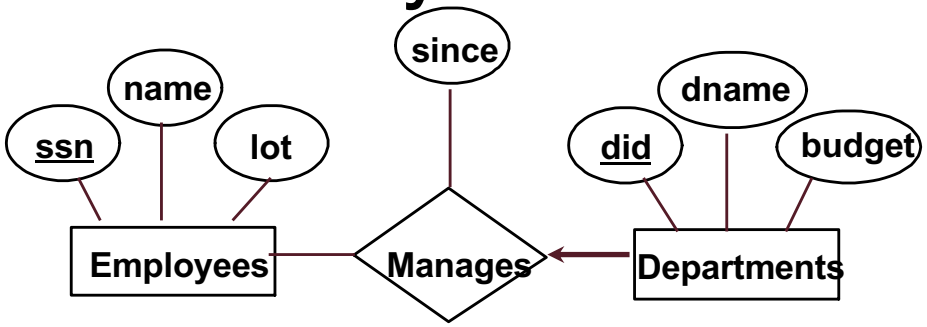


Translating ER with Key Constraints




```
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



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

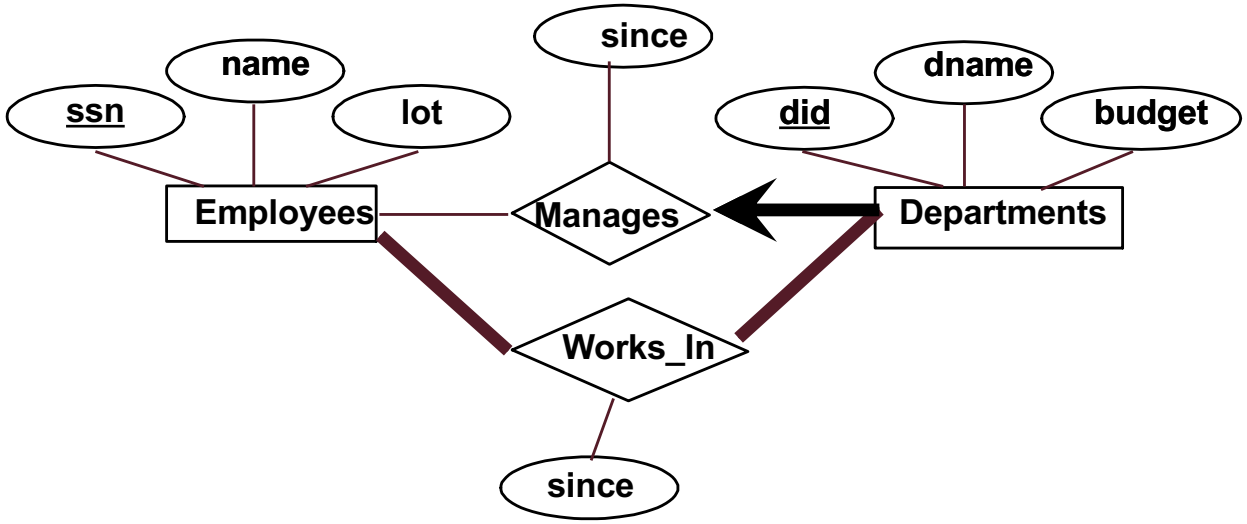
```
CREATE TABLE  Manages(
 ssn  CHAR(11),
 did  INTEGER,
 since  DATE,
 PRIMARY KEY (did),
 FOREIGN KEY (ssn)
   REFERENCES Employees,
 FOREIGN KEY (did)
   REFERENCES Departments)
```

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` must appear in a row of `Manages` (with a non-null `ssn`!)

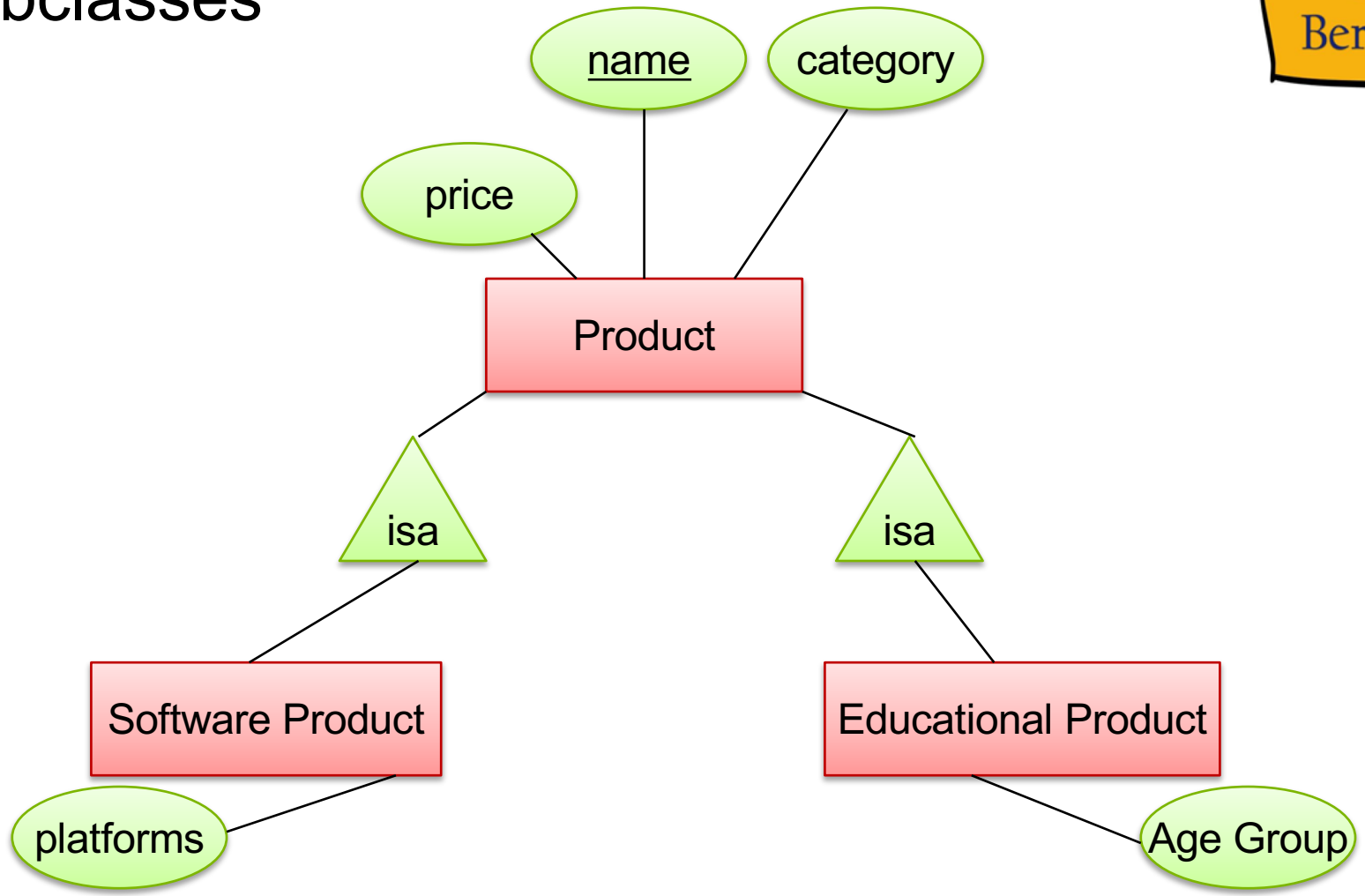


Participation Constraints in SQL

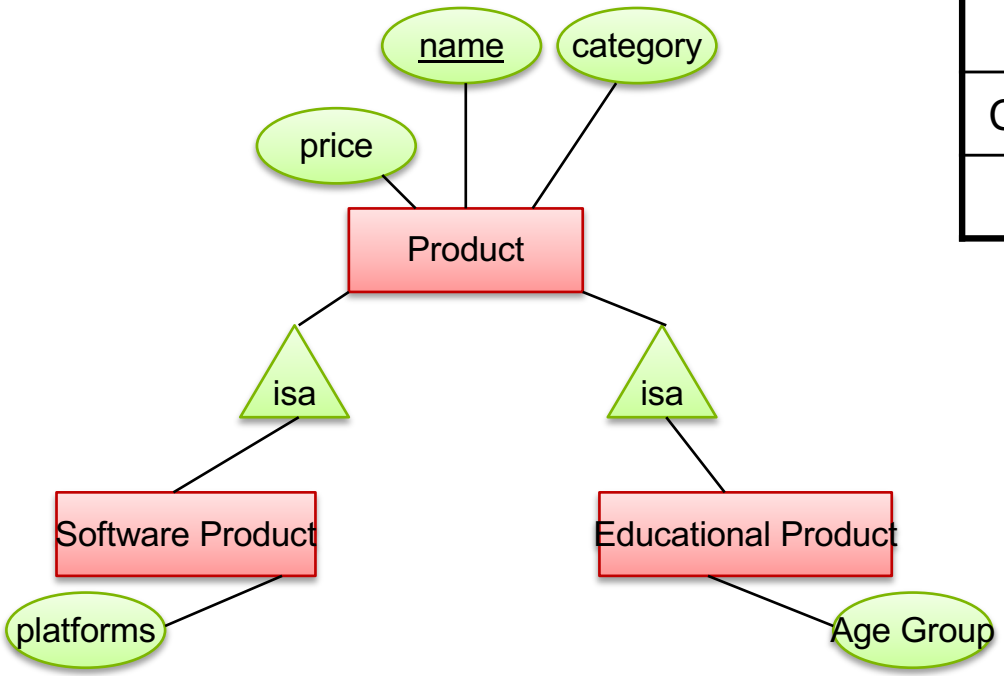
- Participation constraints with one entity set is translated as a binary relationship
- Hard to model more complicated constraints
  - Need to CHECK constraints which we'll learn later.

```
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)
```

Subclasses


name
category
price
Product
isa
isa
Software Product
Educational Product
platforms
Age Group

Subclasses to Relations



Other ways to convert are possible

Product

Name	Price	Category
Gizmo	99	gadget
Camera	49	photo
Toy	39	gadget

Sw.Product

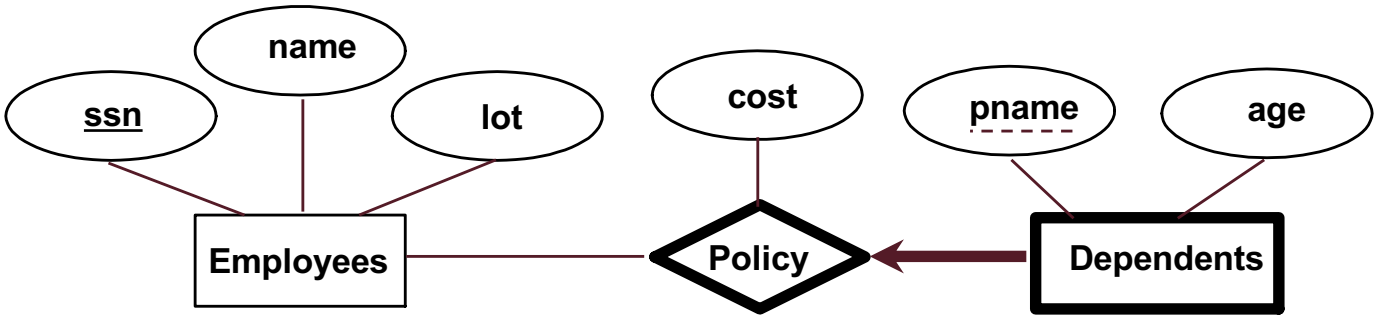
Name	platforms
Gizmo	unix

Ed.Product

Name	Age Group
Gizmo	toddler
Toy	retired

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)
```

we also delete owner entity when deleting weak entity

Summary of ER

- 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 easily analyzable:
  - Capture functional dependencies
  - Can be *normalized*