CSEP 544: Lecture 02

Relational Query Languages
and Database Design

Homework 1

- Due next Tuesday, October 20, 11pm
- Please note update using SQL Azure
  – Use shared account (login in your email)
  – Database is already there, just run queries
- Create your own SQL Azure instance
  – Extra credit in HW1
  – Required for HW4

Homework 3

I know it's far into the future, but…

- We will use Amazon Web Service
- You need to get a $100 student's pass
  http://aws.amazon.com/grants
  - Use your uw.edu email address

Brief Review of 1st Lecture

- Database = collection of related files
- Physical data independence
- SQL:
  - Select-from-where
  - Nested loop semantics
  - Group by (you read the slides, right?)
  - Advanced stuff: nested queries, outerjoins

Big Data

What is it?

Big Data

What is it?

- Gartner report*
  - High Volume
  - High Variety
  - High Velocity

* http://www.gartner.com/newsroom/id/1731916

Big Data

What is it? Stonebraker:

- Big volumes, small analytics
- Big analytics, on big volumes
- Big velocity
- Big variety

Big Data

- “Small analytics” = select/join/aggregate/groupby
  - Discuss: column-oriented databases, shared-nothing, Hive/ Hadoop
- “Big analytics” = linear algebra (R, ScalaPack)
  - Discuss: Sparse matrix multiplication = join/groupby
- High velocity = streaming data
  - Discuss: Streaming SQL engines, e.g. Microsoft’s Trill
- High variety = heterogeneous data models (XML, documents)
  - Discuss: ETL (“Extract Transform Load”)

Outline

- Relational Query Languages
  - Relational algebra
  - Recursion-free datalog with negation
  - Relational calculus
- Database Design
- Functional Dependencies and BCNF

- Suggested reading:
  Three Query Language Formalisms
  https://courses.cs.washington.edu/courses/cse344/12au/lectures/query-language-primer.pdf

1. Relational Algebra

- Used internally by the database engine to execute queries
- Book: chapter 4.2
- We will return to RA when we discuss query execution

1. Relational Algebra

The Basic Five operators:

- Union: $\cup$
- Difference: -
- Selection: σ
- Projection: Π
- Join: $\bowtie$

Running Example

Find all actors who acted both in 1910 and in 1940:

```
Q: SELECT DISTINCT a.fname, a.lname
   FROM   Actor a, Casts c1, Movie m1, Casts c2, Movie m2
   WHERE  a.id = c1.pid    AND c1.mid = m1.id
      AND a.id = c2.pid    AND c2.mid = m2.id
      AND m1.year = 1910       AND m2.year = 1940;
```

Two Perspectives

- Named Perspective:
  Actor(id, fname, lname)
  Casts(pid,mid)
  Movie(id,name,year)
- Unnamed Perspective:
  Actor = arity 3
  Casts = arity 2
  Movie = arity 3

Named perspective needs renaming operator: ρ

1. Relational Algebra (Details)

- Selection: returns tuples that satisfy condition
  - Named perspective: $\sigma_{\text{year = '1910'}}(\text{Movie})$
  - Unamed perspective: $\sigma_{3 \text{ = '1910'}}(\text{Movie})$

1. Relational Algebra (Details)

- Selection: returns tuples that satisfy condition
  - Named perspective: $\sigma_{\text{year = '1910'}}$(Movie)
  - Unamed perspective: $\sigma_{\text{3 = '1910'}}$ (Movie)

- Projection: returns only some attributes
  - Named perspective: $\Pi_{\text{fname,lname}}$(Actor)
  - Unnamed perspective: $\Pi_{2,3}$(Actor)

1. Relational Algebra (Details)

- Selection: returns tuples that satisfy condition
  - Named perspective: $\sigma_{\text{year = '1910'}}$(Movie)
  - Unamed perspective: $\sigma_{\text{3 = '1910'}}$ (Movie)

- Projection: returns only some attributes
  - Named perspective: $\Pi_{\text{fname,lname}}$(Actor)
  - Unnamed perspective: $\Pi_{\text{2,3}}$(Actor)

- Join: joins two tables on a condition
  - Named perspective: Casts $\bowtie_{\text{mid=id}}$ Movie
  - Unnamed perspectivie: Casts $\bowtie_{\text{2=1}}$ Movie

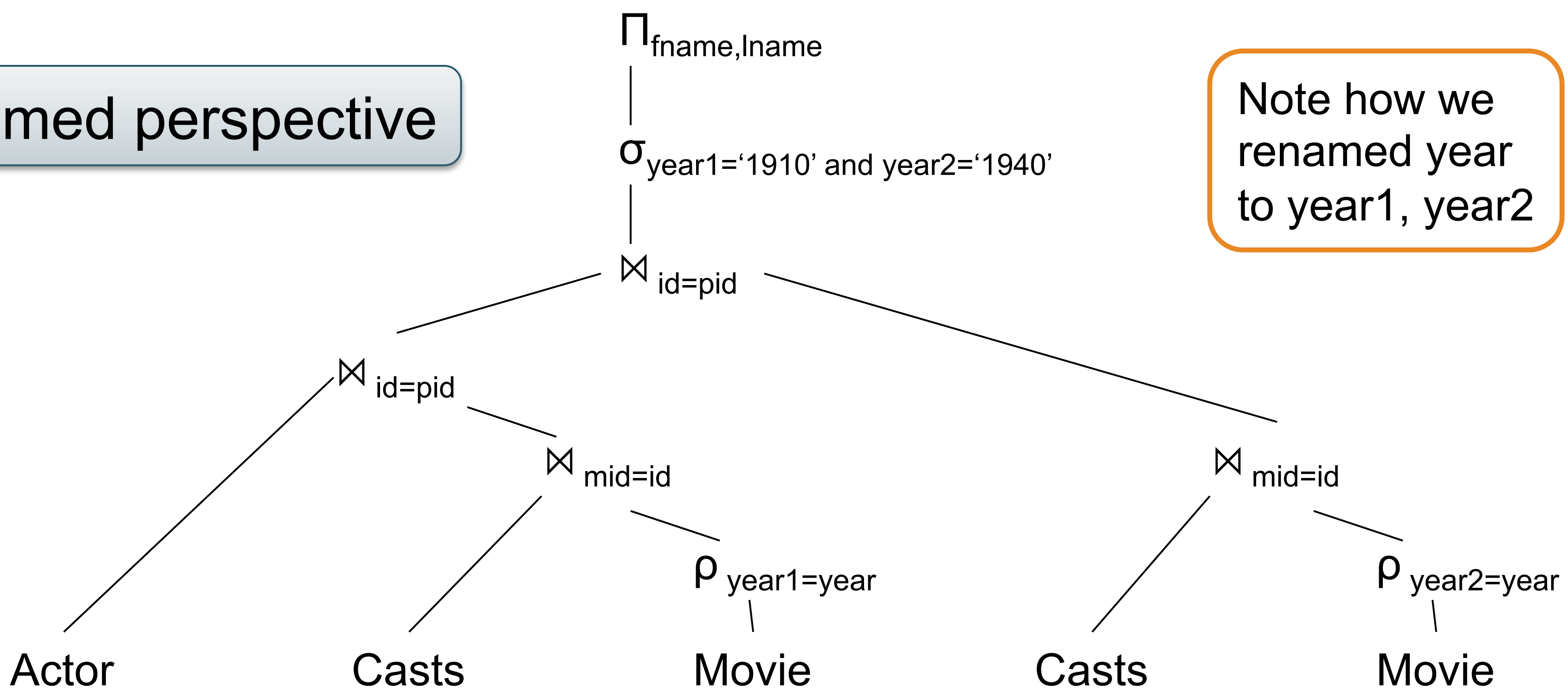
1. Relational Algebra Example

```
Q: SELECT DISTINCT a.fname, a.lname
     FROM   Actor a, Casts c1, Movie m1, Casts c2, Movie m2
     WHERE  a.id = c1.pid        AND c1.mid = m1.id
        AND  a.id = c2.pid        AND c2.mid = m2.id
        AND  m1.year = 1910   AND m2.year = 1940;
```

Actor(id, fname, lname)
Casts(pid,mid)
Movie(id,name,year)

Named perspective

Note how we renamed year to year1, year2

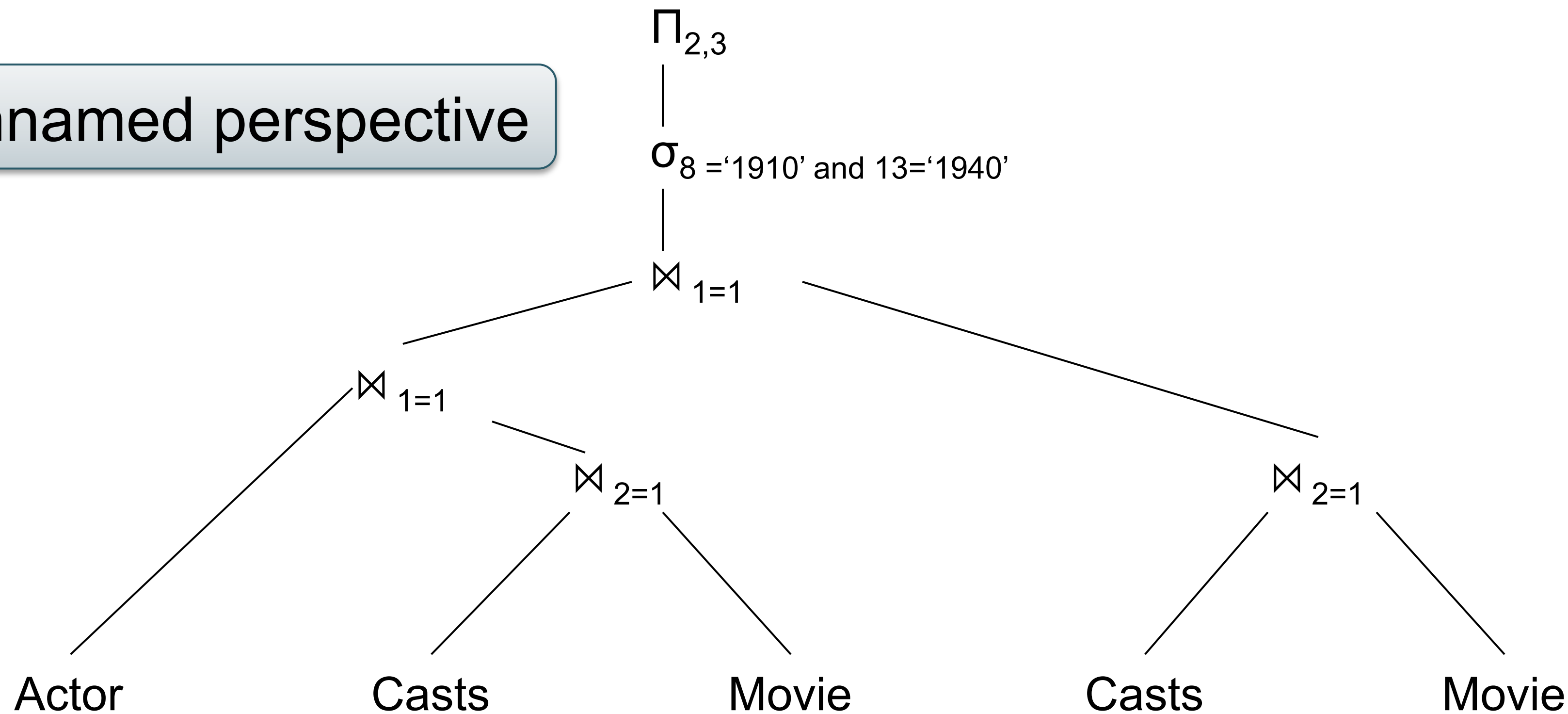


1. Relational Algebra Example

```
Q: SELECT DISTINCT a.fname, a.lname
     FROM   Actor a, Casts c1, Movie m1, Casts c2, Movie m2
     WHERE  a.id = c1.pid        AND c1.mid = m1.id
         AND  a.id = c2.pid        AND c2.mid = m2.id
         AND  m1.year = 1910   AND m2.year = 1940;
```

Actor(id, fname, lname)
Casts(pid,mid)
Movie(id,name,year)

Unnamed perspective



Joins and Cartesian Product

- Each tuple in R1 with each tuple in R2

$$\boxed{R1 \times R2}$$

- Rare in practice; mainly used to express joins

Cartesian Product (aka Cross Product)

Employee

Name	SSN
John	999999999
Tony	777777777

Dependent

EmpSSN	DepName
999999999	Emily
777777777	Joe

Employee $\times$ Dependent

Name	SSN	EmpSSN	DepName
John	999999999	999999999	Emily
John	999999999	777777777	Joe
Tony	777777777	999999999	Emily
Tony	777777777	777777777	Joe

Natural Join

$$\boxed{R1 \bowtie R2}$$

- Meaning: $R1 \bowtie R2 = \Pi_A(\sigma(R1 \times R2))$
- Where:
  - Selection σ checks equality of all common attributes
  - Projection eliminates duplicate common attributes

Natural Join Example

R

A	B
X	Y
X	Z
Y	Z
Z	V

S

B	C
Z	U
V	W
Z	V

R ⋈ S =

$\Pi_{ABC}(\sigma_{R.B=S.B}(R \times S))$

A	B	C
X	Z	U
X	Z	V
Y	Z	U
Y	Z	V
Z	V	W

Natural Join Example 2

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu

Voters V

name	age	zip
p1	54	98125
p2	20	98120

$P \bowtie V$

age	zip	disease	name
54	98125	heart	p1
20	98120	flu	p2

Natural Join

- Given schemas R(A, B, C, D), S(A, C, E), what is the schema of R ⋈ S ?

- Given R(A, B, C), S(D, E), what is R ⋈ S ?

- Given R(A, B), S(A, B), what is R ⋈ S ?

Theta Join

- A join that involves a predicate

$$\boxed{R1 \bowtie_\theta R2 \quad = \quad \sigma_\theta (R1 \times R2)}$$

- Here θ can be any condition
- For our voters/disease example:

$$P \bowtie_{P.zip = V.zip \text{ and } P.age < V.age + 5 \text{ and } P.age > V.age - 5} V$$

Equijoin

- A theta join where θ is an equality

$$\boxed{R1 \bowtie_{A=B} R2 \quad = \quad \sigma_{A=B} (R1 \times R2)}$$

- This is by far the most used variant of join in practice

Equijoin Example

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu

Voters V

name	age	zip
p1	54	98125
p2	20	98120

$P \bowtie_{P.age=V.age} V$

P.age	P.zip	disease	name	V.age	V.zip
54	98125	heart	p1	54	98125
20	98120	flu	p2	20	98120

Note:

Optional, drop the redundant **age**

Join Summary

- **Theta-join**: $R \bowtie_\theta S = \sigma_\theta(R \times S)$
  - Join of R and S with a join condition θ
  - Cross-product followed by selection θ
- **Equijoin**: $R \bowtie_\theta S = \pi_A(\sigma_\theta(R \times S))$
  - Join condition θ consists only of equalities
  - Projection π_A drops all redundant attributes
- **Natural join**: $R \bowtie S = \pi_A(\sigma_\theta(R \times S))$
  - Equijoin
  - Equality on **all** fields with same name in R and in S

So Which Join Is It ?

- When we write R ⋈ S we usually mean an equijoin, but we often omit the equality predicate when it is clear from the context

More Joins

- **Outer join**
  - Include tuples with no matches in the output
  - Use NULL values for missing attributes

- Variants
  - Left outer join
  - Right outer join
  - Full outer join

Outer Join Example

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu
33	98120	lung

AnnonJob J

job	age	zip
lawyer	54	98125
cashier	20	98120

$P \;⟕\; V$

age	zip	disease	job
54	98125	heart	lawyer
20	98120	flu	cashier
33	98120	lung	null

Some Examples

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,qty,price)
```

Q2: Name of supplier of parts with size greater than 10

$\pi_{sname}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{psize>10}\ (\text{Part}))$

Q3: Name of supplier of red parts or parts with size greater than 10

$\pi_{sname}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{psize>10}\ (\text{Part}) \cup \sigma_{pcolor=\text{'red'}}\ (\text{Part})\ )\ )$

Outline

- Relational Query Languages
  - Relational algebra
  - Recursion-free datalog with negation
  - Relational calculus
- Database Design
- Functional Dependencies and BCNF

2. Datalog

- Very friendly notation for queries
- Designed in the 80's for *recursive* queries
- Confined to academia, until the Big Data explosion. Commercial systems today: LogicBlox, Yedalog (google)
- This lecture: *recursion-free* datalog with negation. Later lecture: recursion

2. Datalog

How to try out datalog quickly:

- Download DLV from http://www.dbai.tuwien.ac.at/proj/dlv/
- Run DLV on this file:

```
parent(william, john).
parent(john, james).
parent(james, bill).
parent(sue, bill).
parent(james, carol).
parent(sue, carol).

male(john).
male(james).
female(sue).
male(bill).
female(carol).

grandparent(X, Y) :- parent(X, Z), parent(Z, Y).
father(X, Y) :- parent(X, Y), male(X).
mother(X, Y) :- parent(X, Y), female(X).
brother(X, Y) :- parent(P, X), parent(P, Y), male(X), X != Y.
sister(X, Y)  :- parent(P, X), parent(P, Y), female(X), X != Y.
```

2. Datalog: Facts and Rules

Facts

Actor(344759,'Douglas', 'Fowley').
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, 'A Night in Armour', 1910).
Movie(29000, 'Arizona', 1940).
Movie(29445, 'Ave Maria', 1940).

Rules

Q1(y) :- Movie(x,y,z), z='1940'.

Q2(f, l) :- Actor(z,f,l), Casts(z,x),
Movie(x,y,'1940').

Q3(f,l) :- Actor(z,f,l), Casts(z,x1), Movie(x1,y1,1910),
Casts(z,x2), Movie(x2,y2,1940)

Facts = tuples in the database
Rules = queries

Extensional Database Predicates = EDB
Intensional Database Predicates = IDB

2. Datalog: Terminology

$$\overbrace{\text{Q2(f, l)}}^{\text{head}} \text{ :- } \overbrace{\overbrace{\text{Actor(z,f,l)}}^{\text{atom}}\text{, }\overbrace{\text{Casts(z,x)}}^{\text{atom}}\text{, }\overbrace{\text{Movie(x,y,'1940')}}^{\text{atom}}}^{\text{body}}\text{.}$$

f, l = head variables
x,y,z = existential variables

2. Datalog program

Find all actors with Bacon number ≤ 2

```
B0(x) :- Actor(x,'Kevin', 'Bacon')
B1(x) :- Actor(x,f,l), Casts(x,z), Casts(y,z), B0(y)
B2(x) :- Actor(x,f,l), Casts(x,z), Casts(y,z), B1(y)
Q4(x) :- B1(x)
Q4(x) :- B2(x)
```

Note: Q4 is the *union* of B1 and B2

2. Datalog with negation

Find all actors with Bacon number ≥ 2

```
B0(x) :- Actor(x,'Kevin', 'Bacon')
B1(x) :- Actor(x,f,l), Casts(x,z), Casts(y,z), B0(y)
Q6(x) :- Actor(x,f,l), not B1(x), not B0(x)
```

2. Safe Datalog Rules

Here are *unsafe* datalog rules. What's "unsafe" about them ?

U1(x,y) :- Movie(x,z,1994), y>1910

U2(x) :- Movie(x,z,1994), not Casts(u,x)

A datalog rule is *safe* if every variable appears in some positive relational atom

2. Datalog v.s. SQL

- Non-recursive datalog with negation is very close to SQL; with some practice, you should be able to translate between them back and forth without difficulty; see example in the paper

Outline

- Relational Query Languages
  - Relational algebra
  - Recursion-free datalog with negation
  - Relational calculus
- Database Design
- Functional Dependencies and BCNF

3. Relational Calculus

- Also known as *predicate calculus*, or *first order logic*
- The most expressive formalism for queries: easy to write complex queries

- TRC = Tuple RC = named perspective
- DRC = Domain RC = unnamed perspective

3. Relational Calculus

Predicate P:

$$P ::= \text{atom} \mid P \wedge P \mid P \vee P \mid P \Rightarrow P \mid \text{not}(P) \mid \forall x.P \mid \exists x.P$$

Query Q:

$$Q(x_1, \ldots, x_k) = P$$

Example: find the first/last names of actors who acted in 1940

$$Q(f,l) = \exists x.\ \exists y.\ \exists z.\ (\text{Actor}(z,f,l) \wedge \text{Casts}(z,x) \wedge \text{Movie}(x,y,1940))$$

What does this query return ?

$$Q(f,l) = \exists z.\ (\text{Actor}(z,f,l) \wedge \forall x.(\text{Casts}(z,x) \Rightarrow \exists y.\text{Movie}(x,y,1940)))$$

3. Relational Calculus: Example

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> beer they like.

$$Q(x) = \exists y.\ \exists z.\ \text{Frequents}(x, y) \wedge \text{Serves}(y,z) \wedge \text{Likes}(x,z)$$

3. Relational Calculus: Example

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> beer they like.

$$Q(x) = \exists y.\ \exists z.\ \text{Frequents}(x, y) \wedge \text{Serves}(y,z) \wedge \text{Likes}(x,z)$$

Find drinkers that frequent <u>only</u> bars that serves <u>some</u> beer they like.

3. Relational Calculus: Example

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> beer they like.

$$Q(x) = \exists y.\ \exists z.\ \text{Frequents}(x, y) \wedge \text{Serves}(y,z) \wedge \text{Likes}(x,z)$$

Find drinkers that frequent <u>only</u> bars that serves <u>some</u> beer they like.

$$Q(x) = \forall y.\ \text{Frequents}(x, y) \Rightarrow (\exists z.\ \text{Serves}(y,z) \wedge \text{Likes}(x,z))$$

Find drinkers that frequent <u>some</u> bar that serves <u>only</u> beers they like.

3. Relational Calculus: Example

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> beer they like.

$$Q(x) = \exists y.\ \exists z.\ \text{Frequents}(x, y) \wedge \text{Serves}(y,z) \wedge \text{Likes}(x,z)$$

Find drinkers that frequent <u>only</u> bars that serves <u>some</u> beer they like.

$$Q(x) = \forall y.\ \text{Frequents}(x, y) \Rightarrow (\exists z.\ \text{Serves}(y,z) \wedge \text{Likes}(x,z))$$

Find drinkers that frequent <u>some</u> bar that serves <u>only</u> beers they like.

$$Q(x) = \exists y.\ \text{Frequents}(x, y) \wedge \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$$

Find drinkers that frequent <u>only</u> bars that serves <u>only</u> beer they like.

3. Relational Calculus: Example

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Find drinkers that frequent some bar that serves some beer they like.

$$Q(x) = \exists y.\ \exists z.\ \text{Frequents}(x, y) \land \text{Serves}(y,z) \land \text{Likes}(x,z)$$

Find drinkers that frequent only bars that serves some beer they like.

$$Q(x) = \forall y.\ \text{Frequents}(x, y) \Rightarrow (\exists z.\ \text{Serves}(y,z) \land \text{Likes}(x,z))$$

Find drinkers that frequent some bar that serves only beers they like.

$$Q(x) = \exists y.\ \text{Frequents}(x, y) \land \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$$

Find drinkers that frequent only bars that serves only beer they like.

$$Q(x) = \forall y.\ \text{Frequents}(x, y) \Rightarrow \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$$

3. Domain Independent Relational Calculus

- As in datalog, one can write “unsafe” RC queries; they are also called *domain dependent*
- See examples in the *Three Query Languages* paper
- Moral: make sure your RC queries are always domain independent

3. Relational Calculus

Take home message:

- Need to write a complex SQL query:
- First, write it in RC
- Next, translate it to datalog (see next)
- Finally, write it in SQL

As you gain experience, take shortcuts

3. From RC to Non-recursive Datalog w/ negation

Query: Find drinkers that like some beer so much that they frequent all bars that serve it

$$Q(x) = \exists y.\ \text{Likes}(x, y) \wedge \forall z.(\text{Serves}(z,y) \Rightarrow \text{Frequents}(x,z))$$

3. From RC to Non-recursive Datalog w/ negation

Query: Find drinkers that like some beer so much that they frequent all bars that serve it

$$Q(x) = \exists y.\ Likes(x, y) \wedge \forall z.(Serves(z,y) \Rightarrow Frequents(x,z))$$

Step 1: Replace ∀ with ∃ using de Morgan's Laws

$$Q(x) = \exists y.\ Likes(x, y) \wedge \neg\exists z.(Serves(z,y) \wedge \neg Frequents(x,z))$$

3. From RC to Non-recursive Datalog w/ negation

Query: Find drinkers that like some beer so much that they frequent all bars that serve it

$$Q(x) = \exists y.\ \text{Likes}(x, y) \wedge \forall z.(\text{Serves}(z,y) \Rightarrow \text{Frequents}(x,z))$$

Step 1: Replace ∀ with ∃ using de Morgan's Laws

$$Q(x) = \exists y.\ \text{Likes}(x, y) \wedge \neg\exists z.(\text{Serves}(z,y) \wedge \neg\text{Frequents}(x,z))$$

Step 2: Make all subqueries domain independent

$$Q(x) = \exists y.\ \text{Likes}(x, y) \wedge \neg\exists z.(\text{Likes}(x,y) \wedge \text{Serves}(z,y) \wedge \neg\text{Frequents}(x,z))$$

3. From RC to Non-recursive Datalog w/ negation

$$Q(x) = \exists y.\ \text{Likes}(x, y) \land \underbrace{\neg \exists z.(\text{Likes}(x,y) \land \text{Serves}(z,y) \land \neg \text{Frequents}(x,z))}_{H(x,y)}$$

Step 3: Create a datalog rule for each subexpression;
(shortcut: only for subexpressions under ¬)

```
H(x,y)   :- Likes(x,y),Serves(y,z), not Frequents(x,z)
Q(x)     :- Likes(x,y), not H(x,y)
```

3. From RC to Non-recursive Datalog w/ negation

```
H(x,y)   :- Likes(x,y),Serves(y,z), not Frequents(x,z)
Q(x)     :- Likes(x,y), not H(x,y)
```

Step 4: Write it in SQL

```
SELECT DISTINCT L.drinker FROM Likes L
WHERE not exists
  (SELECT * FROM Likes L2, Serves S
   WHERE L2.drinker=L.drinker and L2.beer=L.beer
         and L2.beer=S.beer
         and not exists (SELECT * FROM Frequents F
                         WHERE F.drinker=L2.drinker
                           and F.bar=S.bar))
```

3. From RC to Non-recursive Datalog w/ negation

```
H(x,y)   :- ~~Likes(x,y),~~Serves(y,z), not Frequents(x,z)
Q(x)     :- Likes(x,y), not H(x,y)
```

Unsafe rule

Improve the SQL query by using an unsafe datalog rule

```
SELECT DISTINCT L.drinker FROM Likes L
WHERE not exists
  (SELECT * FROM Serves S
   WHERE L.beer=S.beer
        and not exists (SELECT * FROM Frequents F
                        WHERE F.drinker=L.drinker
                              and F.bar=S.bar))
```

Summary of Translation

- RC → recursion-free datalog w/ negation
  - Subtle: as we saw; more details in the paper
- Recursion-free datalog w/ negation → RA
  - Easy: see paper
- RA → RC
  - Easy: see paper

Summary

- All three have same expressive power:
  - RA
  - Non-recursive datalog w/ neg. (= “core” SQL)
  - RC

- Write complex queries in RC first, then translate to SQL

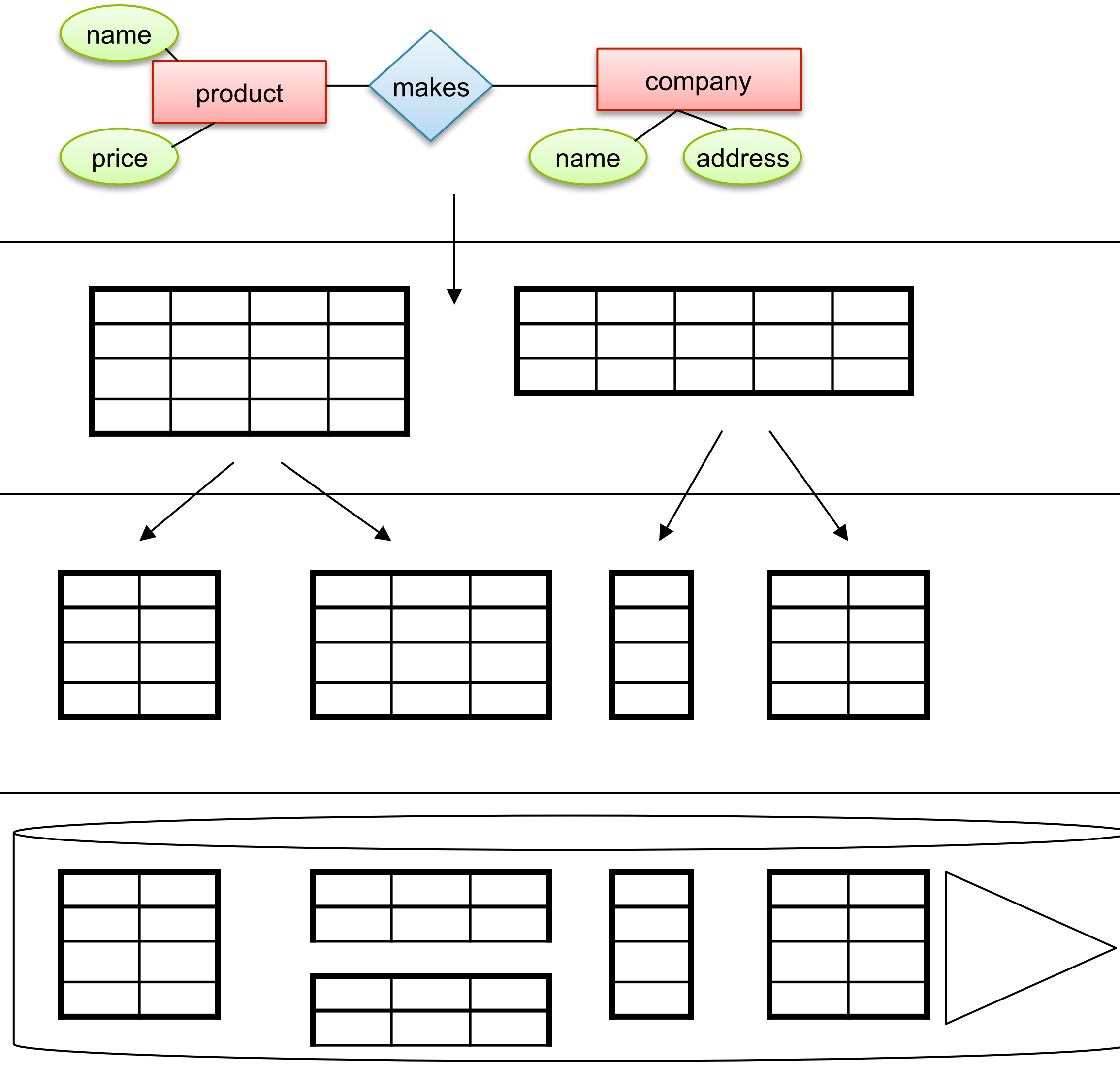
Outline

- Relational Query Languages
- Database Design:
  - On your own: slides and/or Chapters 2, 3
  - In class: *What goes around*
- Functional Dependencies and BCNF

Database Design

Database Design Process

Conceptual Model:



Relational Model:

Tables + constraints

And also functional dep.

Normalization:

Eliminates anomalies

Conceptual Schema

Physical storage details

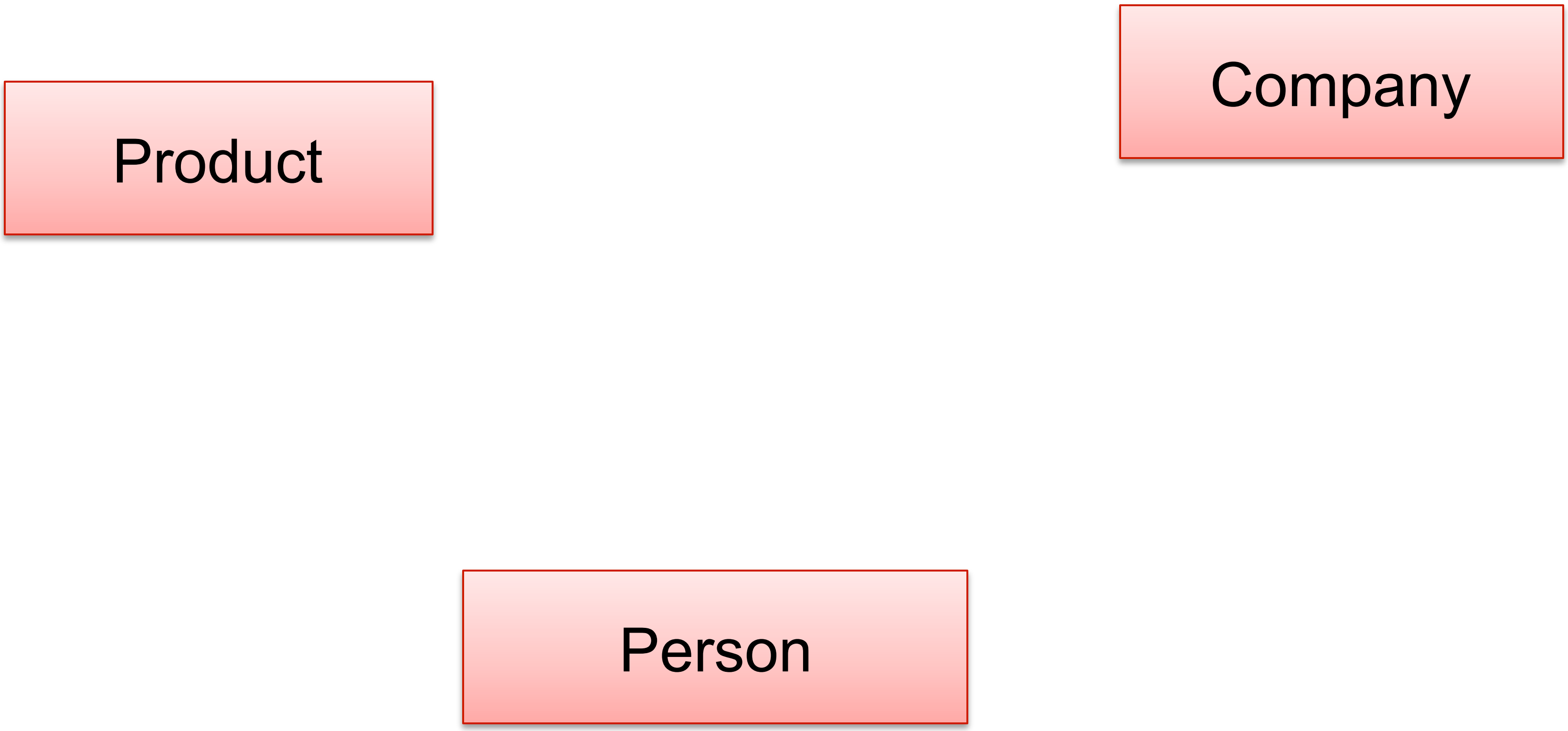
Physical Schema

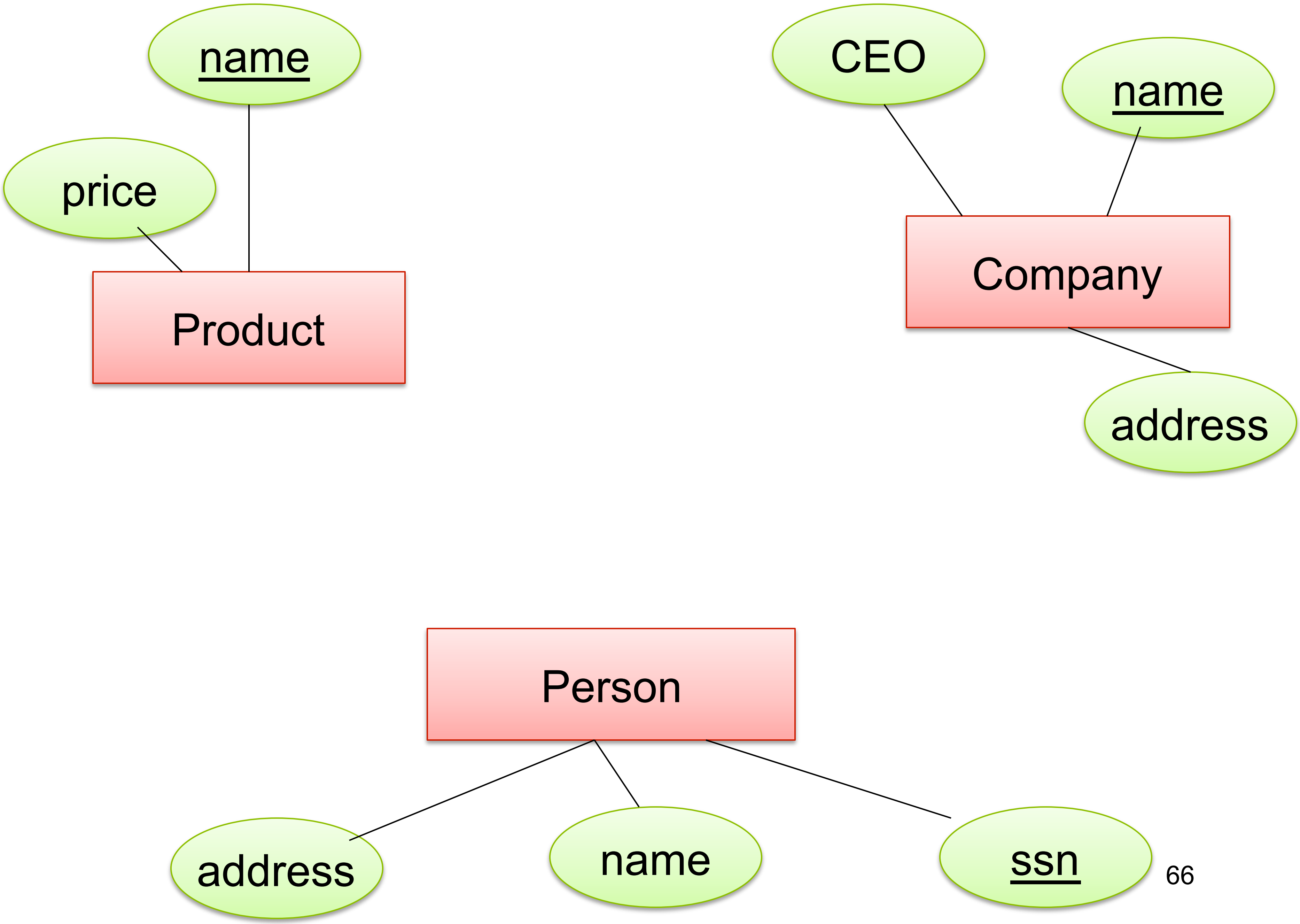
Entity / Relationship Diagrams

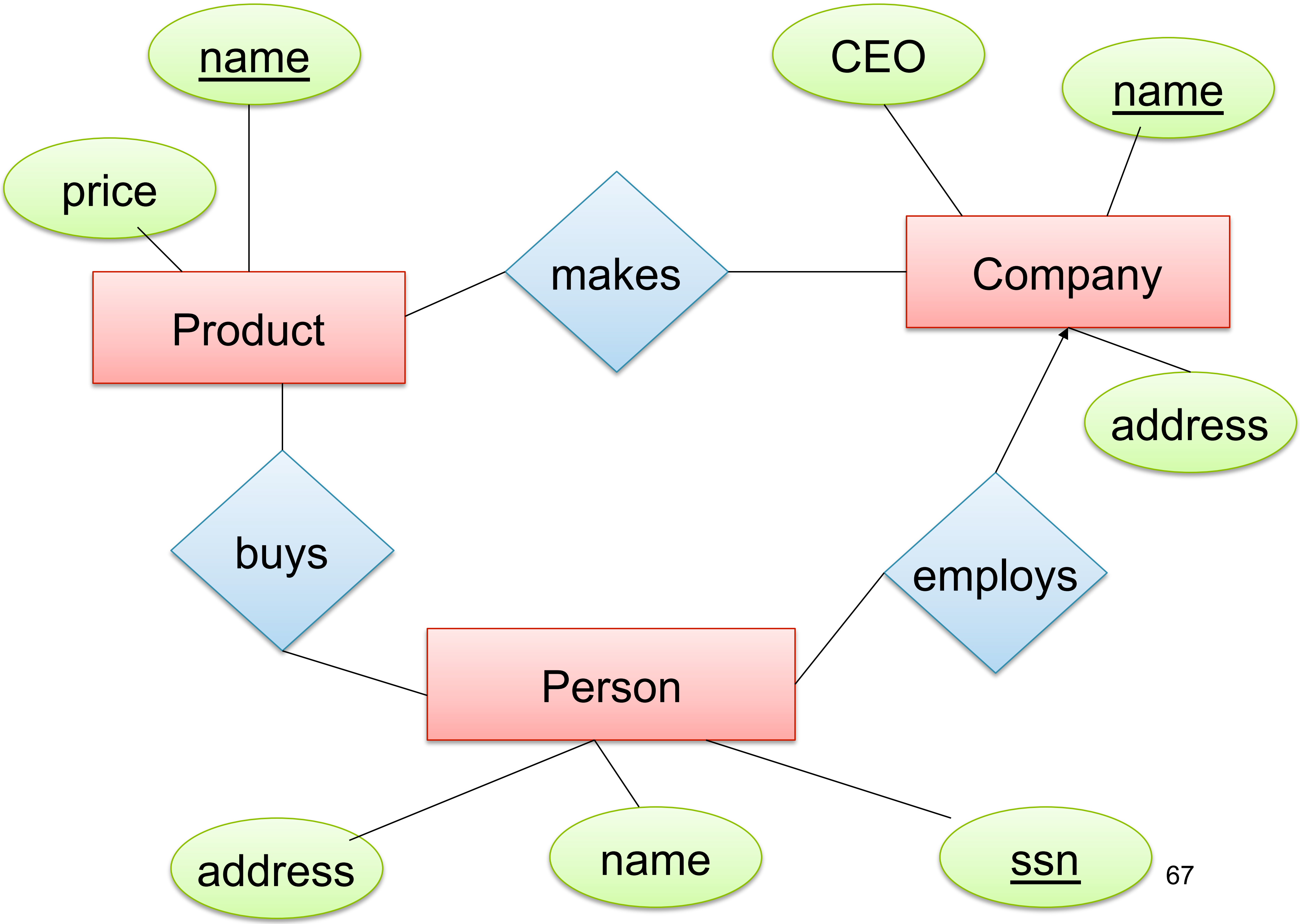
- Entity set = a class
  - An entity = an object

- Attribute

- Relationship

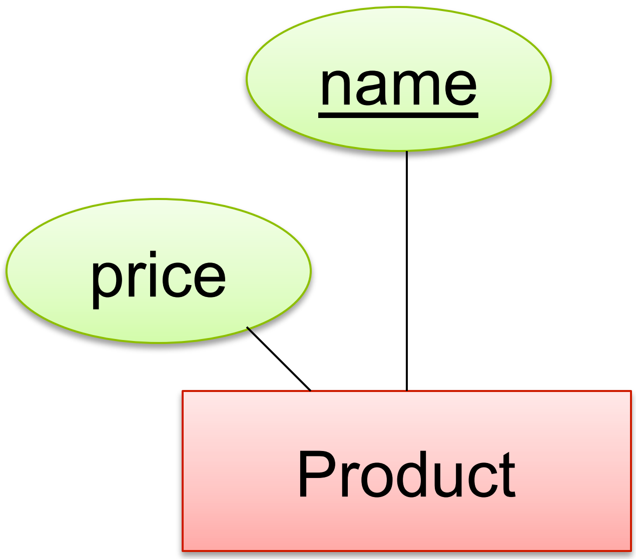

Product
Company
Person


name
price
Product
CEO
name
Company
address
Person
address
name
ssn


name
price
Product
makes
CEO
name
Company
address
buys
employs
Person
address
name
ssn

Keys in E/R Diagrams

- Every entity set must have a key

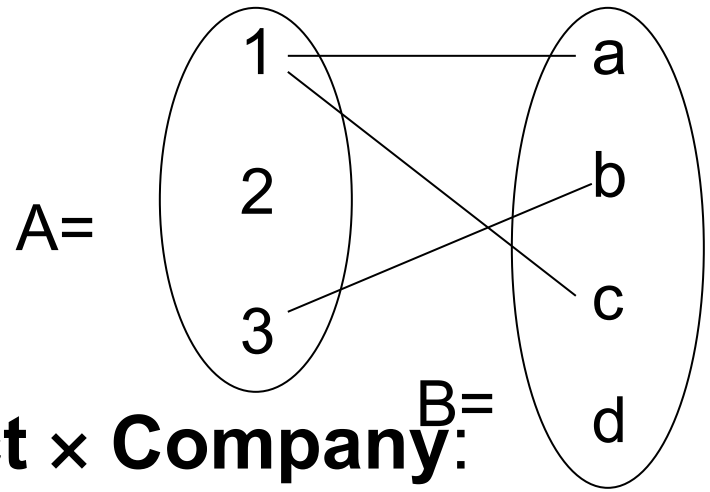


What is a Relation ?

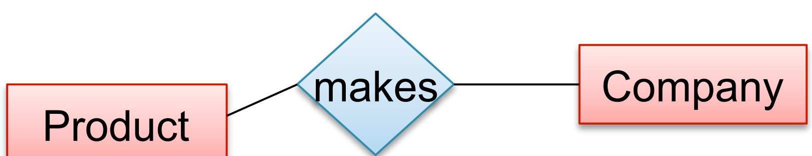
- A mathematical definition:
  - if A, B are sets, then a relation R is a subset of $A \times B$
- $A=\{1,2,3\}$, $B=\{a,b,c,d\}$,

  $A \times B = \{(1,a),(1,b), \ldots, (3,d)\}$

  $R = \{(1,a), (1,c), (3,b)\}$

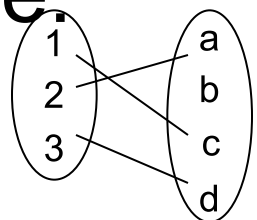


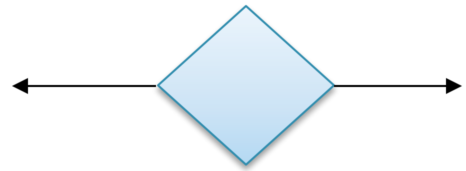
- **makes** is a subset of **Product** $\times$ **Company**:



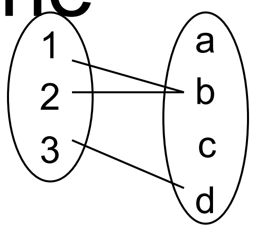
Multiplicity of E/R Relations

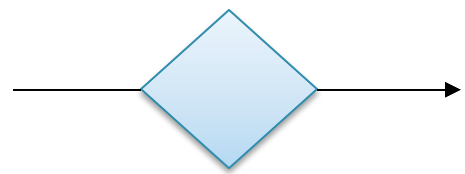
- one-one:



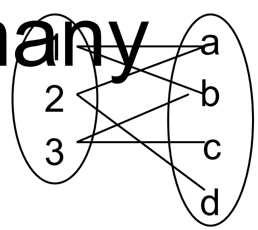


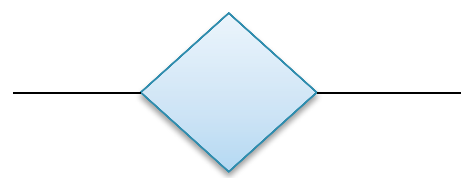
- many-one

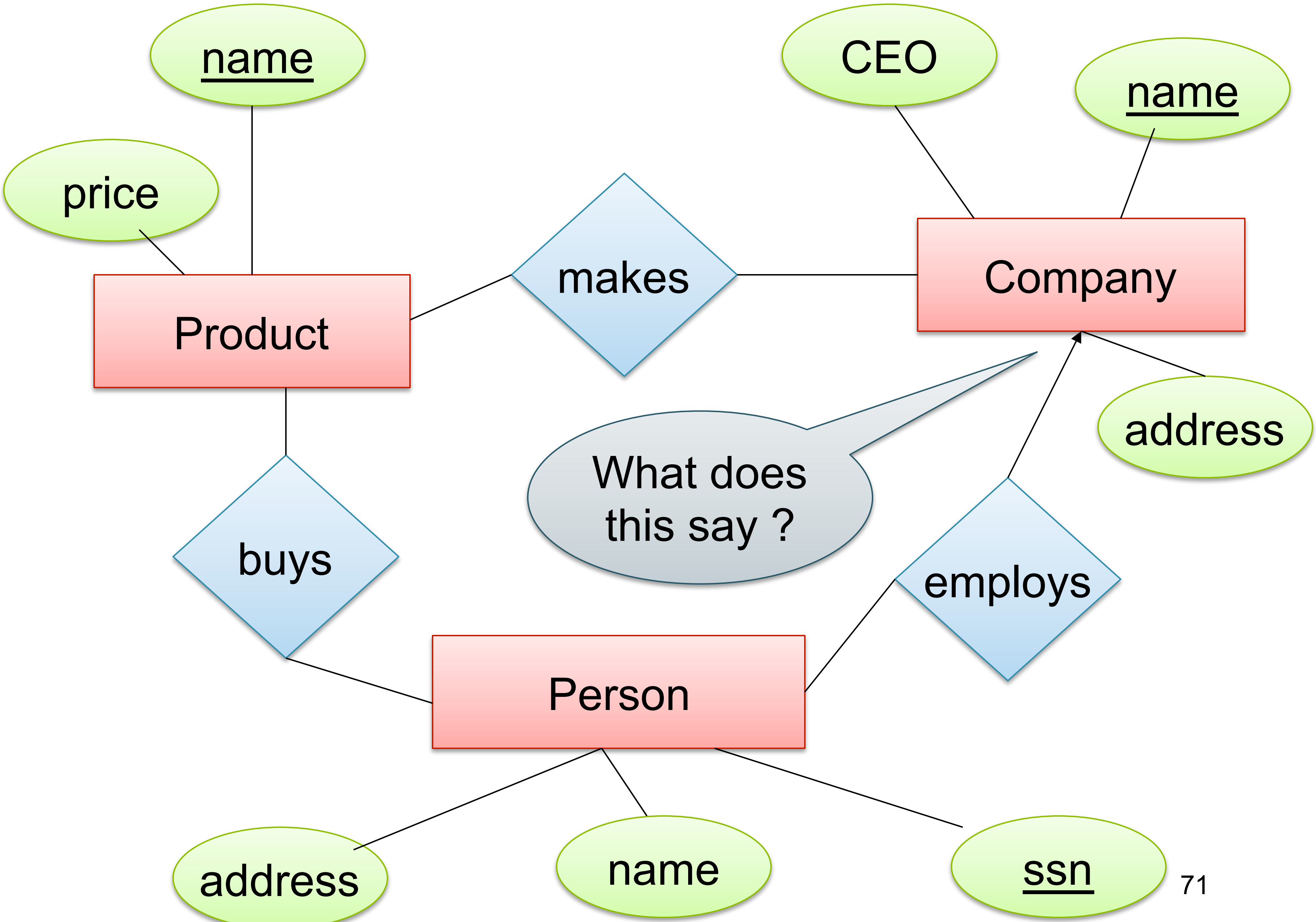




- many-many

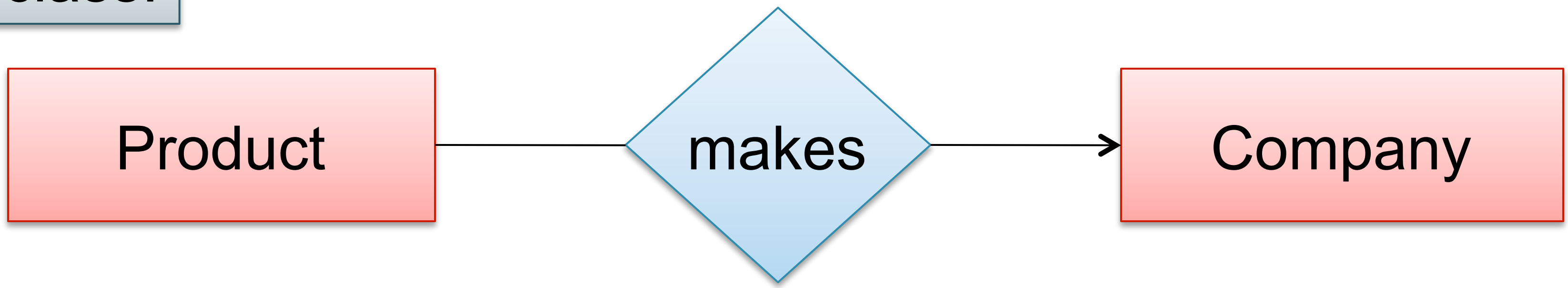




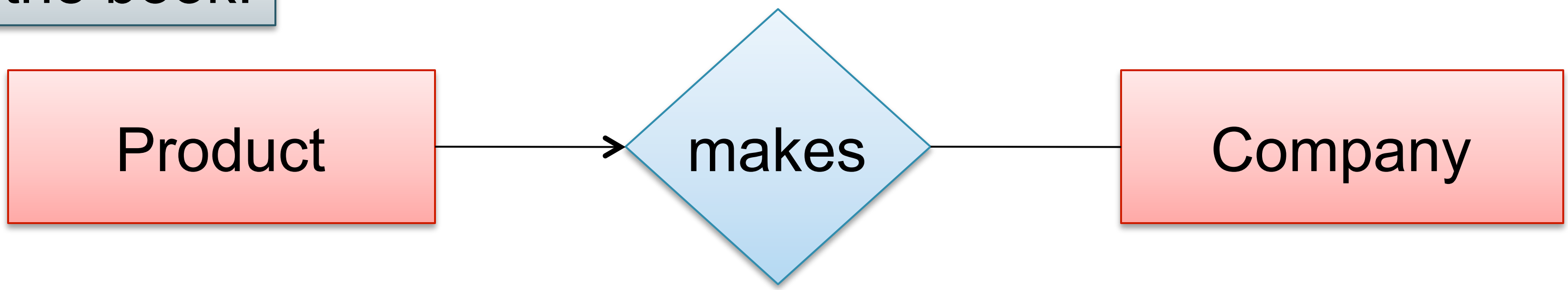

name
price
Product
makes
CEO
name
Company
address
What does this say ?
buys
employs
Person
address
name
ssn

Notation in Class v.s. the Book

In class:

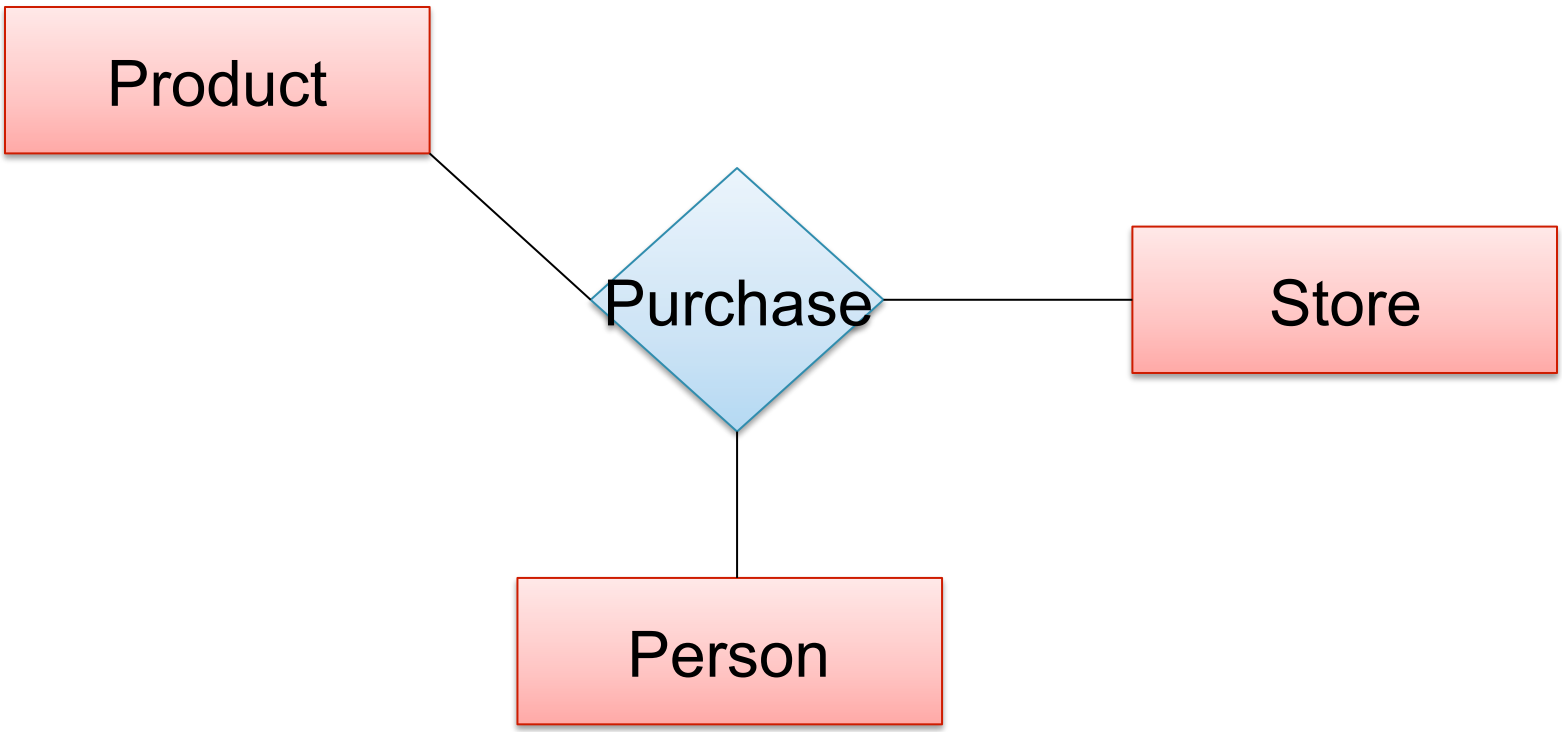


In the book:



Multi-way Relationships

How do we model a purchase relationship between buyers, products and stores?

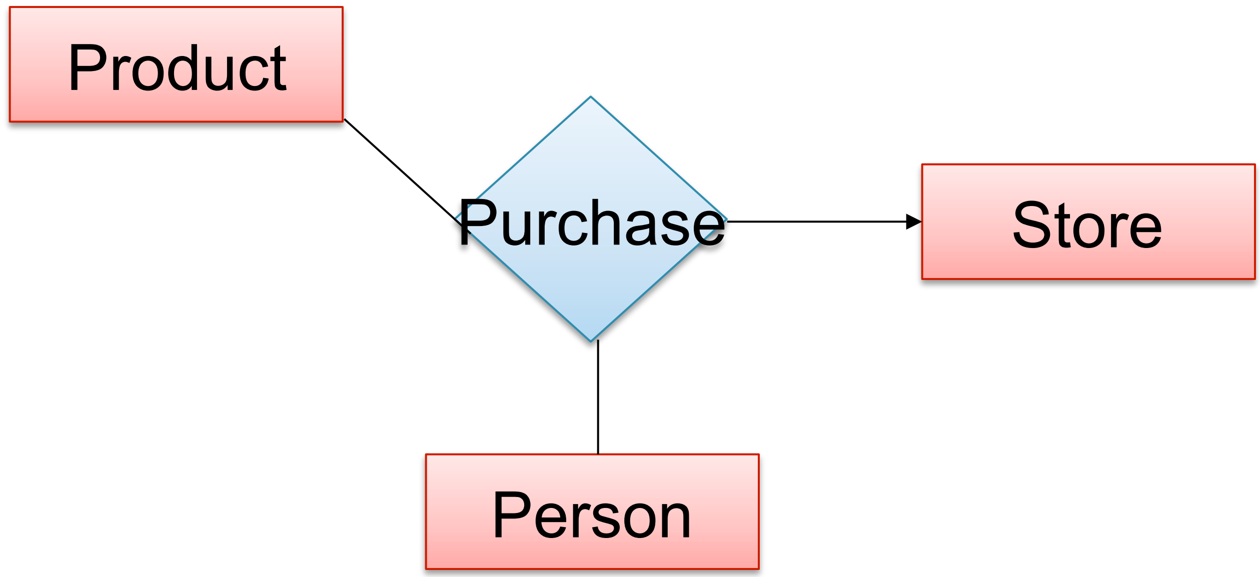


Can still model as a mathematical set (Q. how ?)

A. As a set of triples $\subseteq$ Person $\times$ Product $\times$ Store

Arrows in Multiway Relationships

Q: What does the arrow mean ?

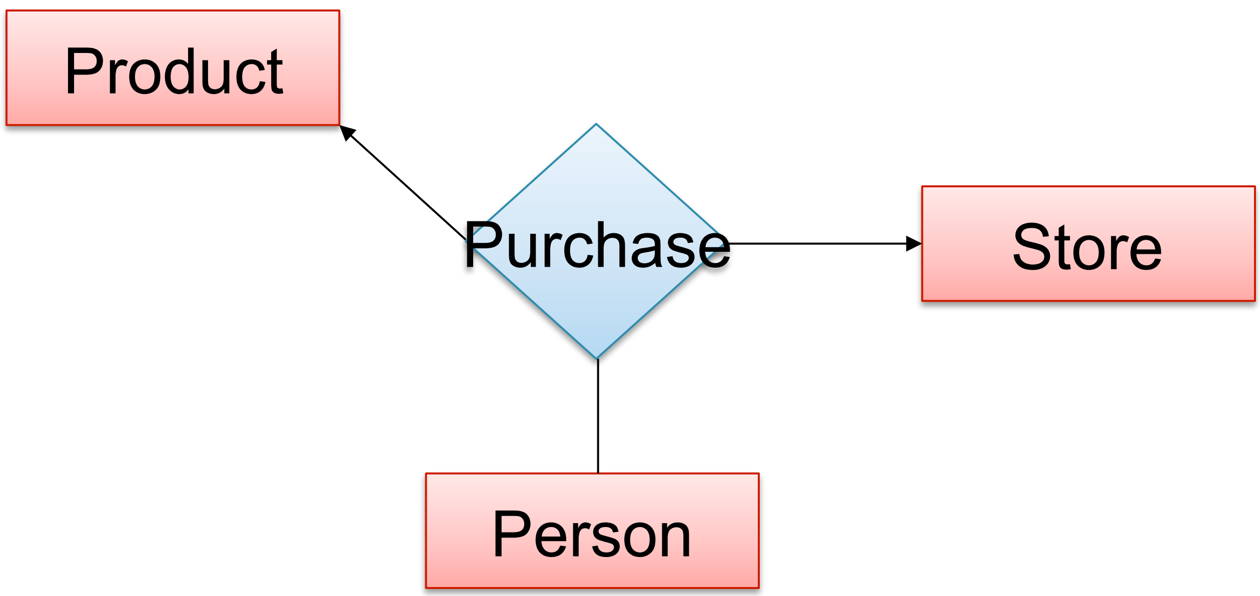


A: A given person buys a given product from at most one store

[Arrow pointing to E means that if we select one entity from each of the other entity sets in the relationship, those entities are related to at most one entity in E]

Arrows in Multiway Relationships

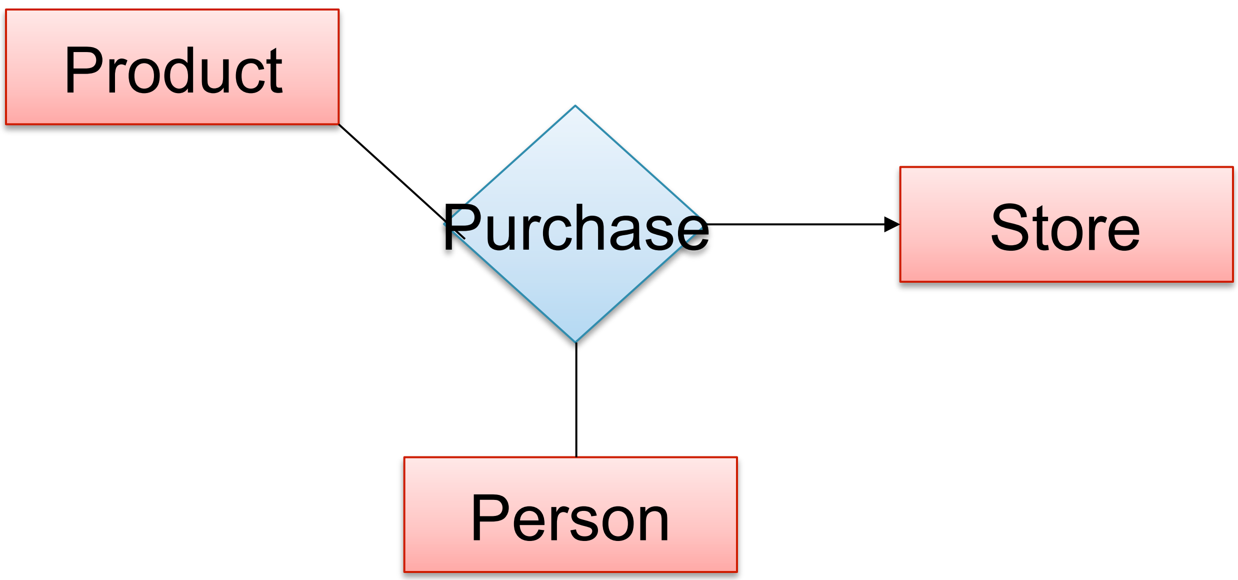
Q: What does the arrow mean ?



A: A given person buys a given product from at most one store AND every store sells to every person at most one product

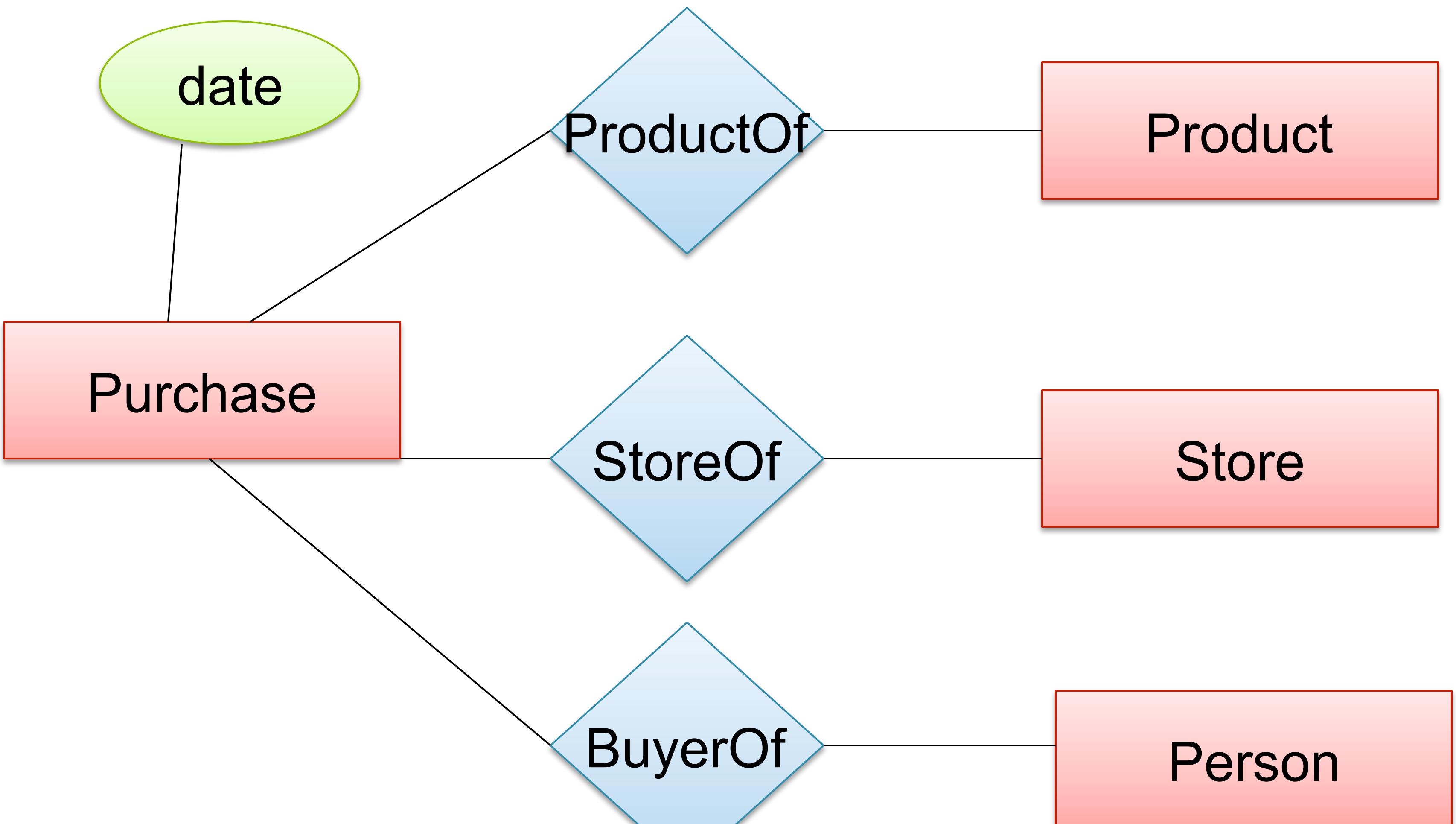
Arrows in Multiway Relationships

Q: How do we say that every person shops at at most one store ?



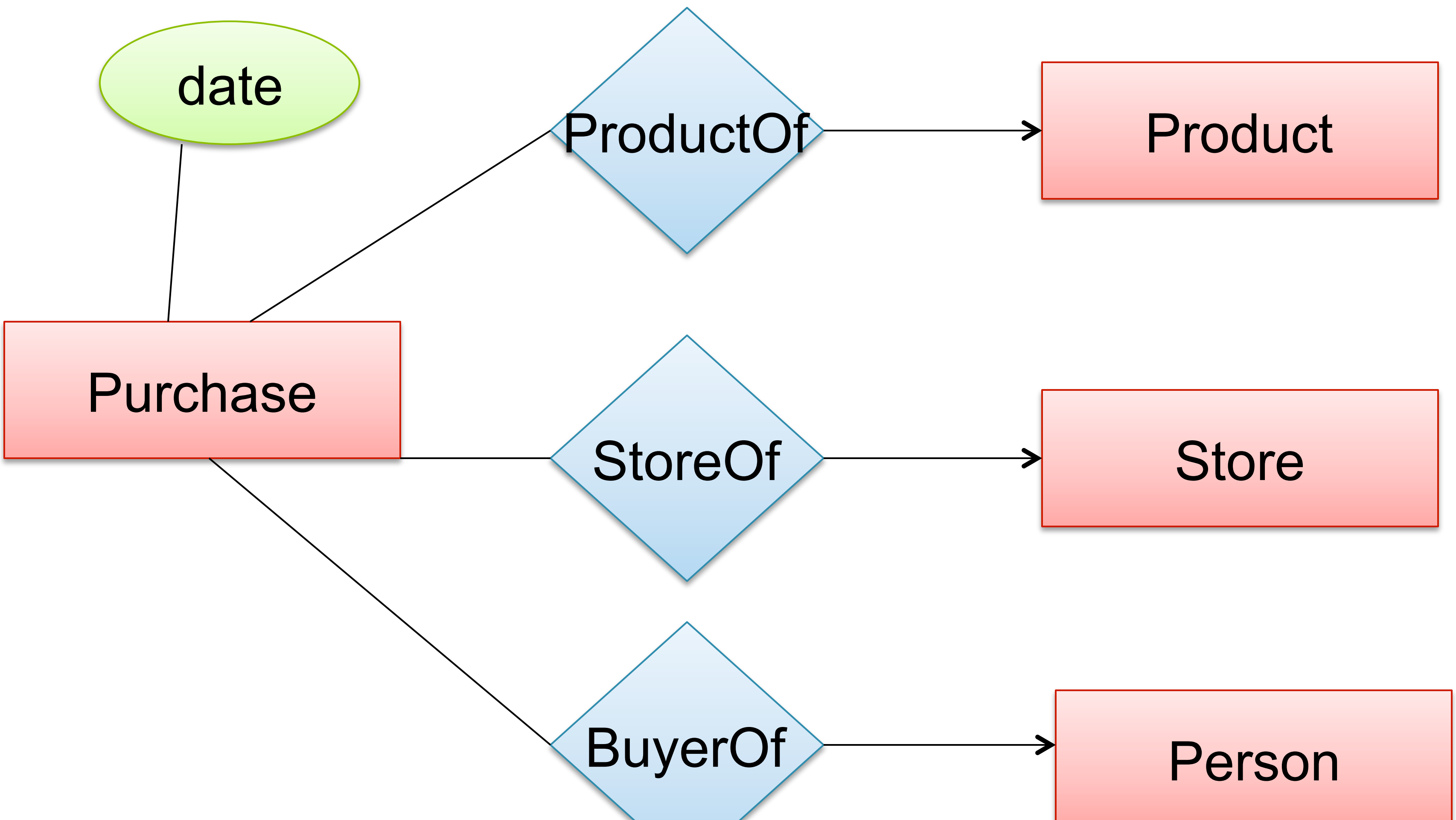
A: Cannot. This is the best approximation.
(Why only approximation ?)

Converting Multi-way Relationships to Binary



Arrows go in which direction?

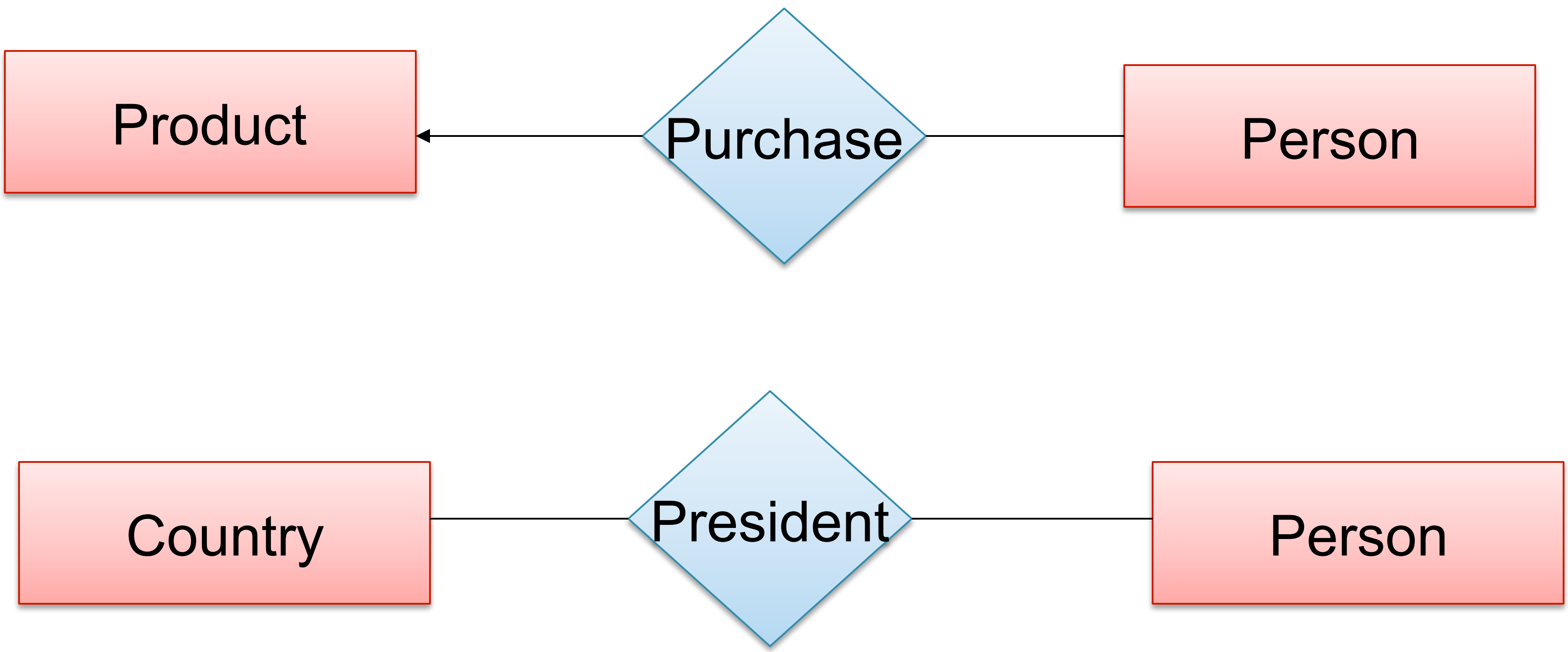
Converting Multi-way Relationships to Binary



Make sure you understand why!

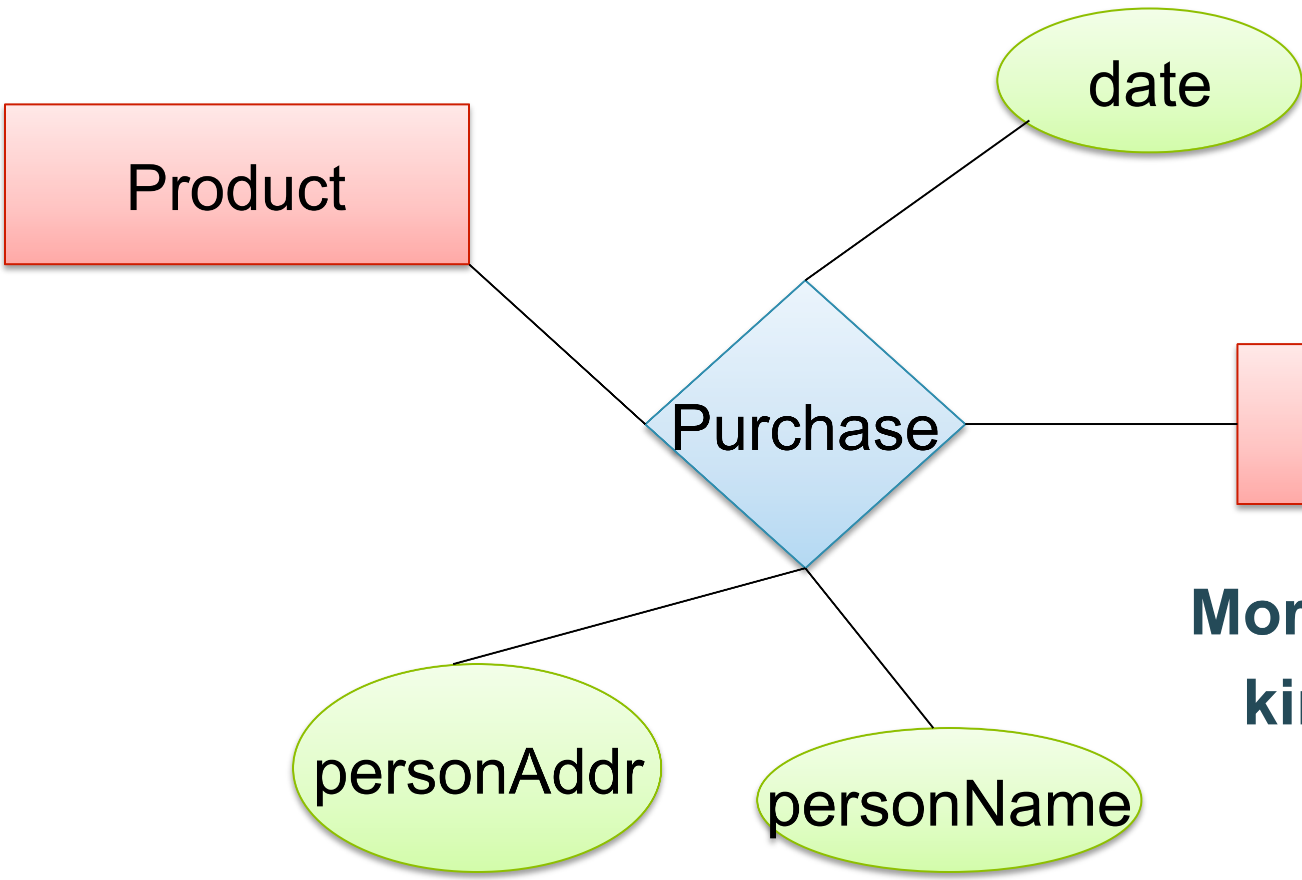
Design Principles

What's wrong?



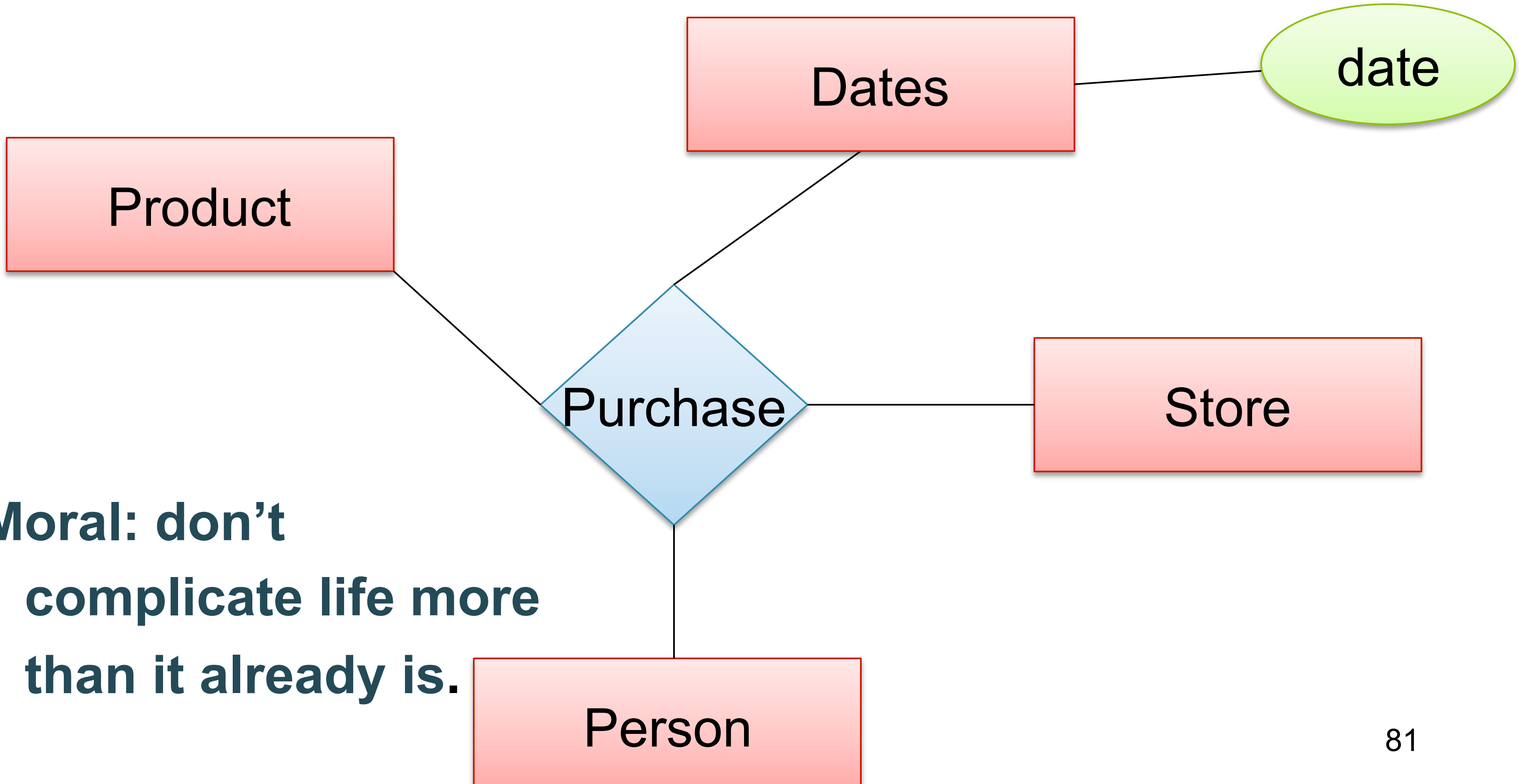
Moral: be faithful to the specifications of the app!

Design Principles: What's Wrong?



Moral: pick the right kind of entities.

Design Principles: What's Wrong?

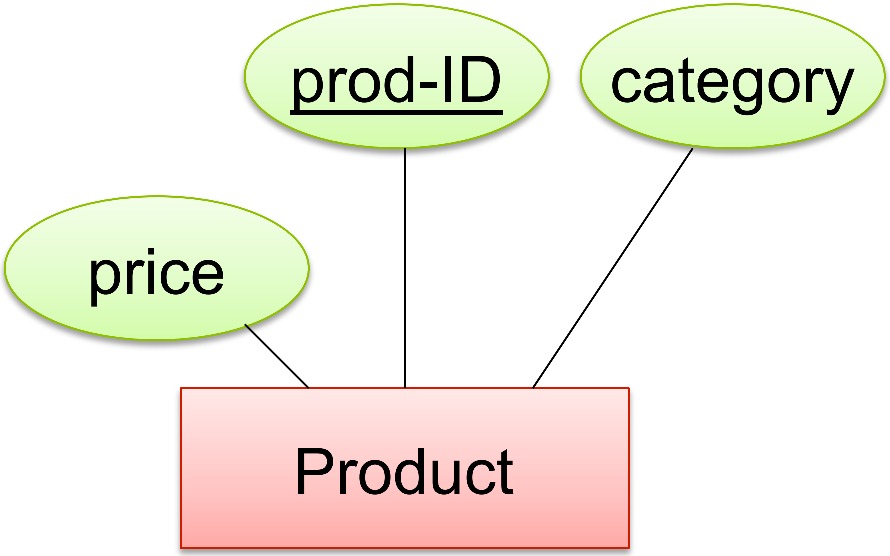


Moral: don't complicate life more than it already is.

From E/R Diagrams to Relational Schema

- Entity set → relation
- Relationship → relation

Entity Set to Relation



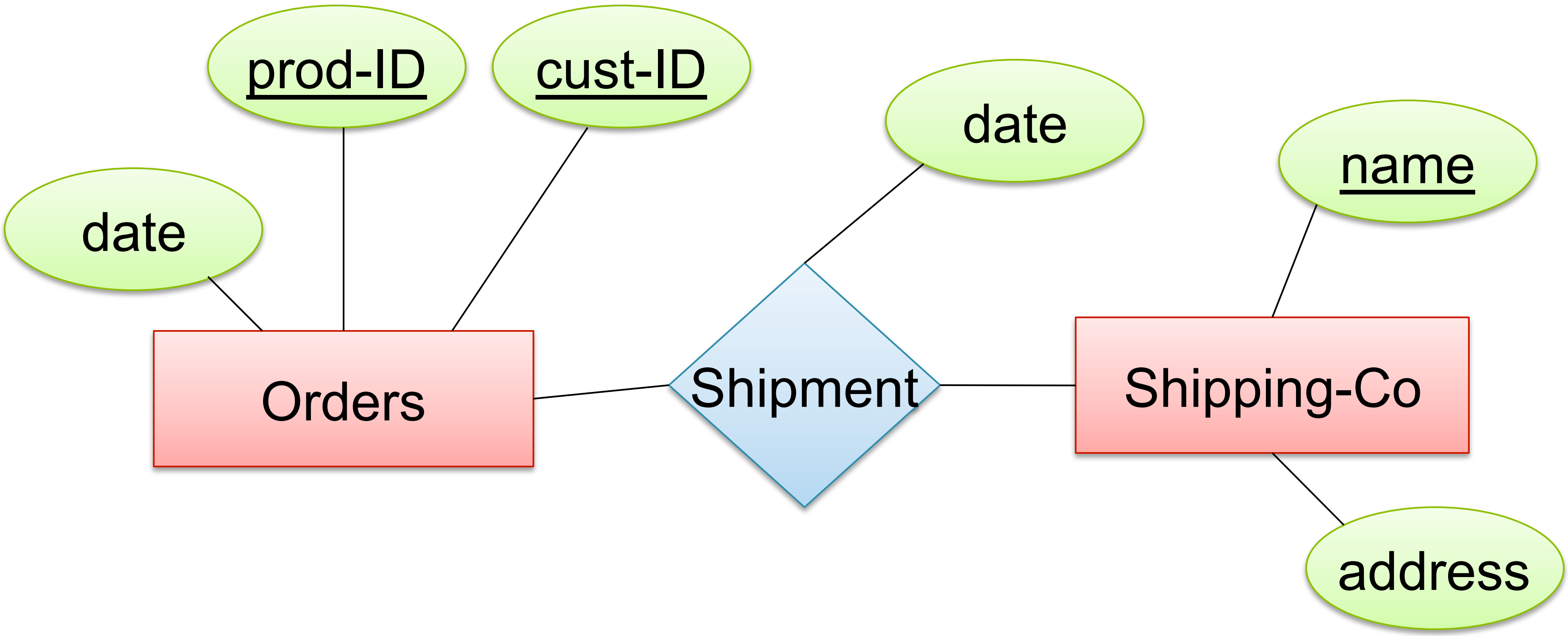
Product(<u>prod-ID</u>, category, price)

<u>prod-ID</u>	category	price
Gizmo55	Camera	99.99
Pokemn19	Toy	29.99

Create Table (SQL)

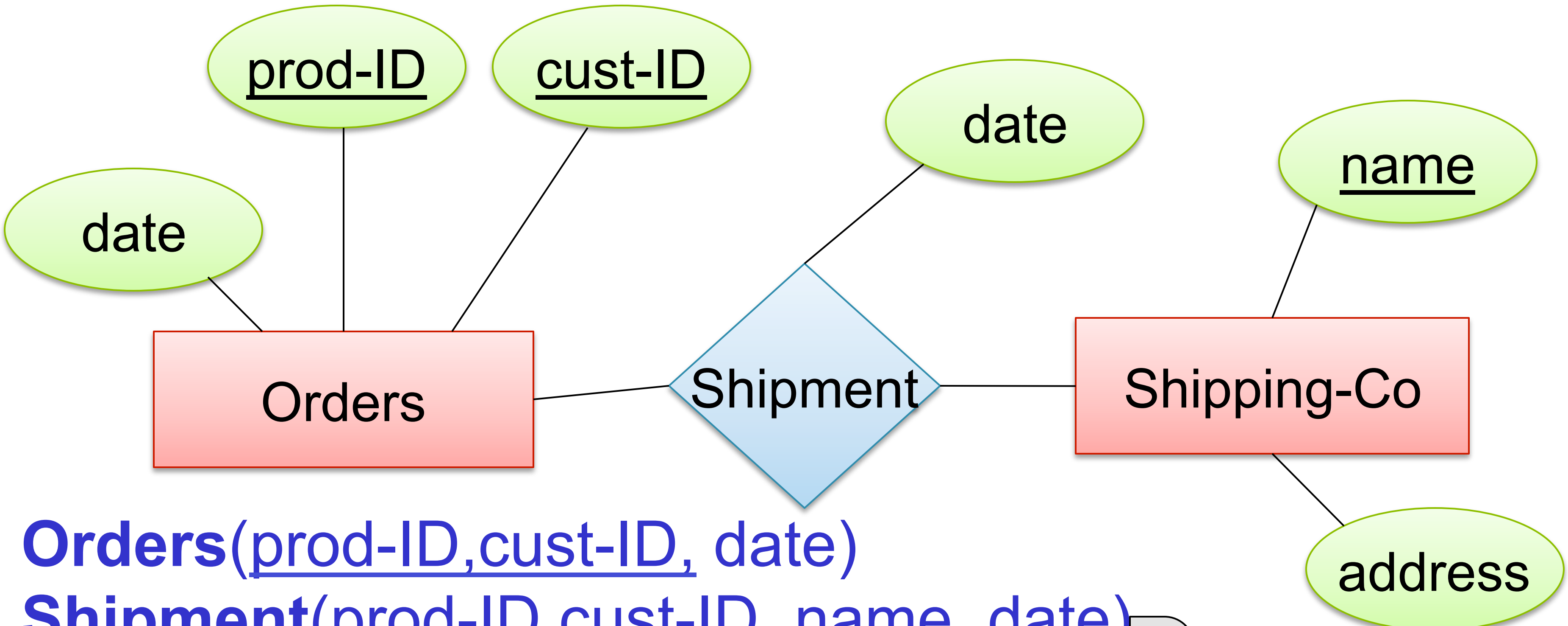
```
CREATE TABLE Product (
        prod-ID CHAR(30) PRIMARY KEY,
        category VARCHAR(20),
         price double)
```

N-N Relationships to Relations



Represent that in relations!

N-N Relationships to Relations



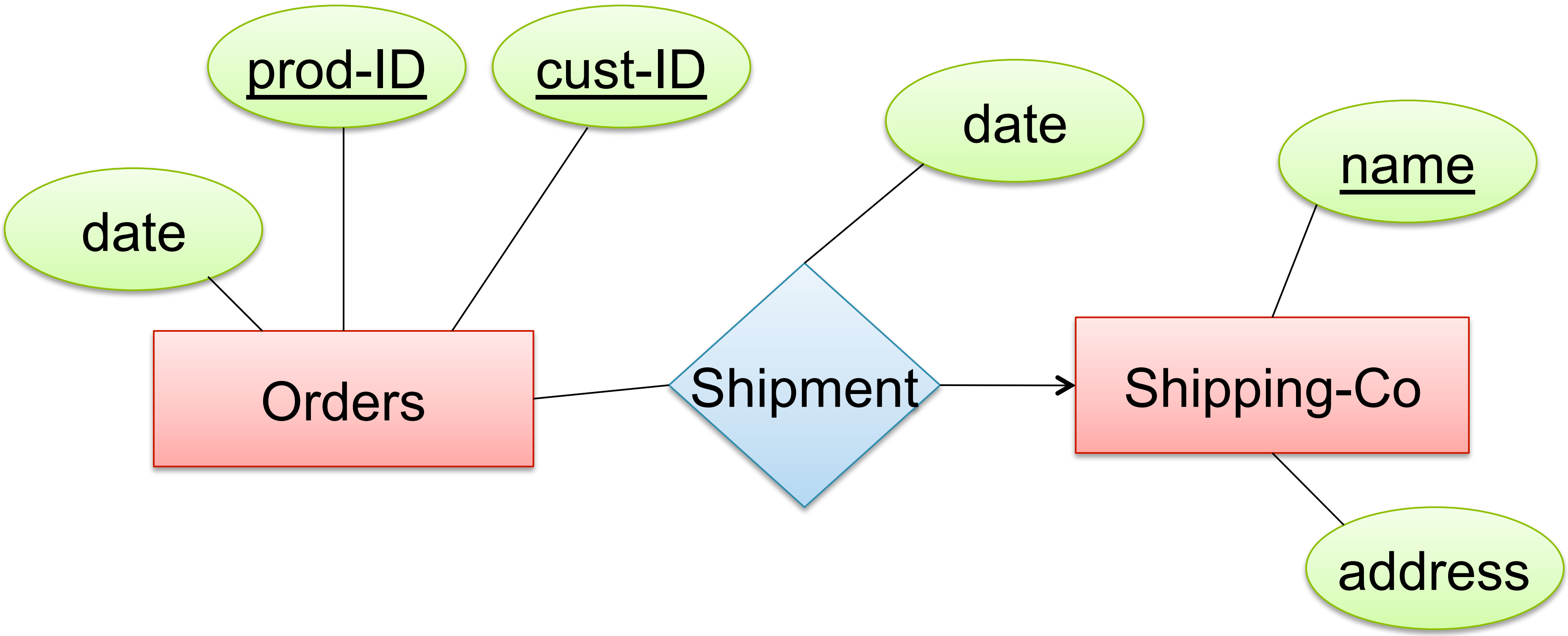
Orders(<u>prod-ID,cust-ID,</u> date)
Shipment(<u>prod-ID,cust-ID, name</u>, date)
Shipping-Co(<u>name</u>, address)

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

Create Table (SQL)

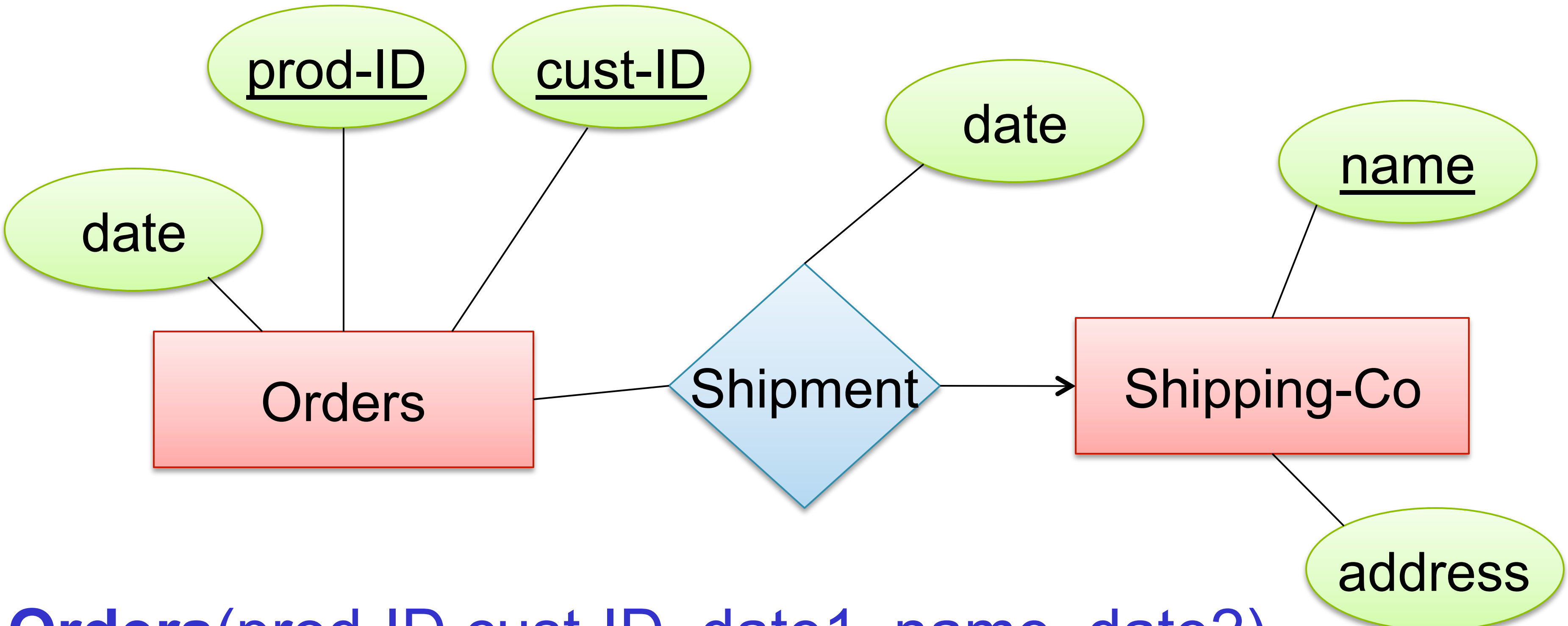
```
CREATE TABLE Shipment(
        name CHAR(30)
                REFERENCES Shipping-Co,
        prod-ID CHAR(30),
        cust-ID VARCHAR(20),
        date DATETIME,
PRIMARY KEY (name, prod-ID, cust-ID),
FOREIGN KEY (prod-ID, cust-ID)
            REFERENCES  Orders
)
```

N-1 Relationships to Relations



Represent this in relations!

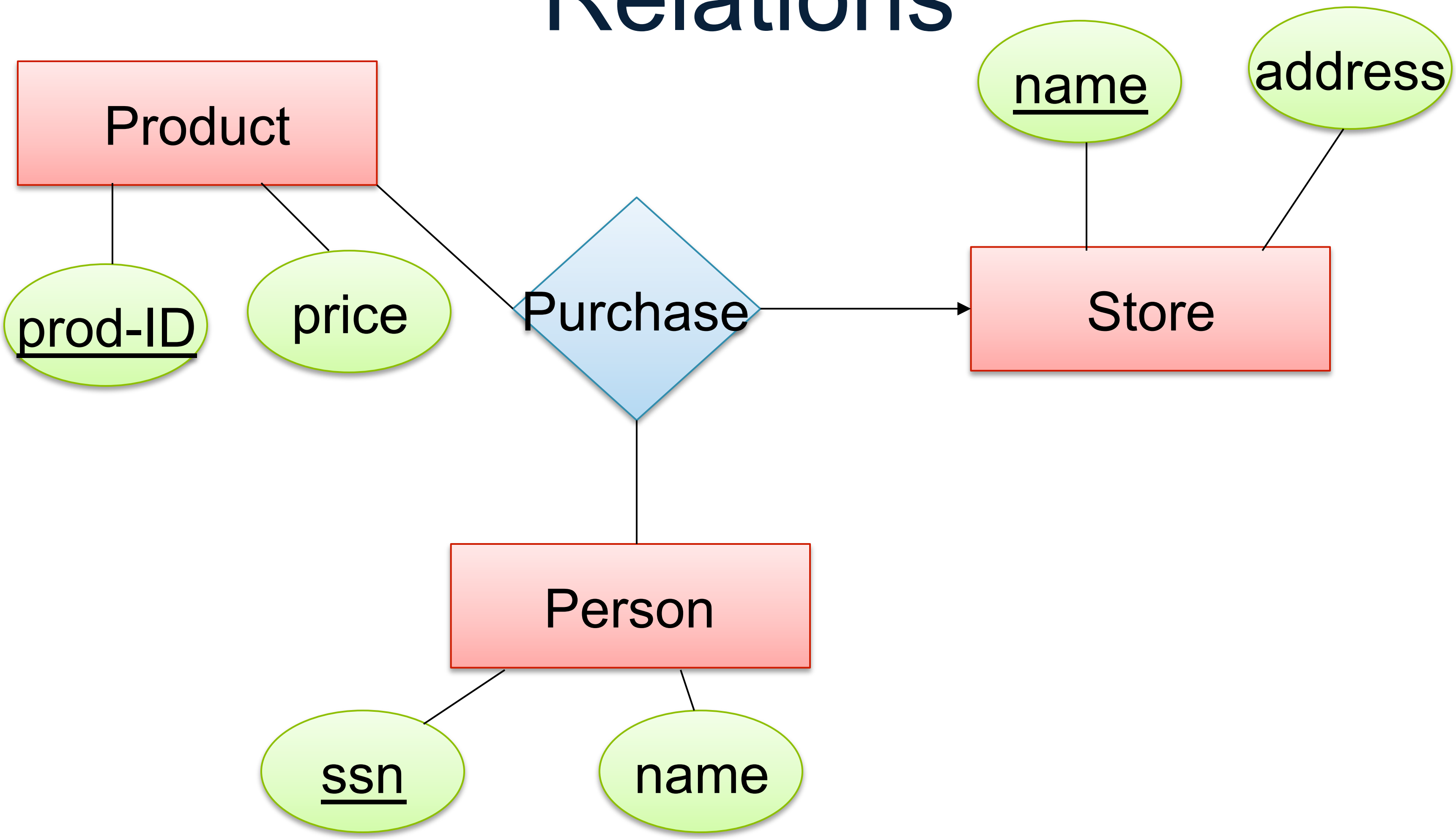
N-1 Relationships to Relations



Orders(<u>prod-ID,cust-ID,</u> date1, name, date2)
Shipping-Co(<u>name</u>, address)

Remember: no separate relations for many-one relationship

Multi-way Relationships to Relations



Purchase(prod-ID, cust-ssn, store-name)

Modeling Subclasses

Some objects in a class may be special

- define a new class
- better: define a *subclass*

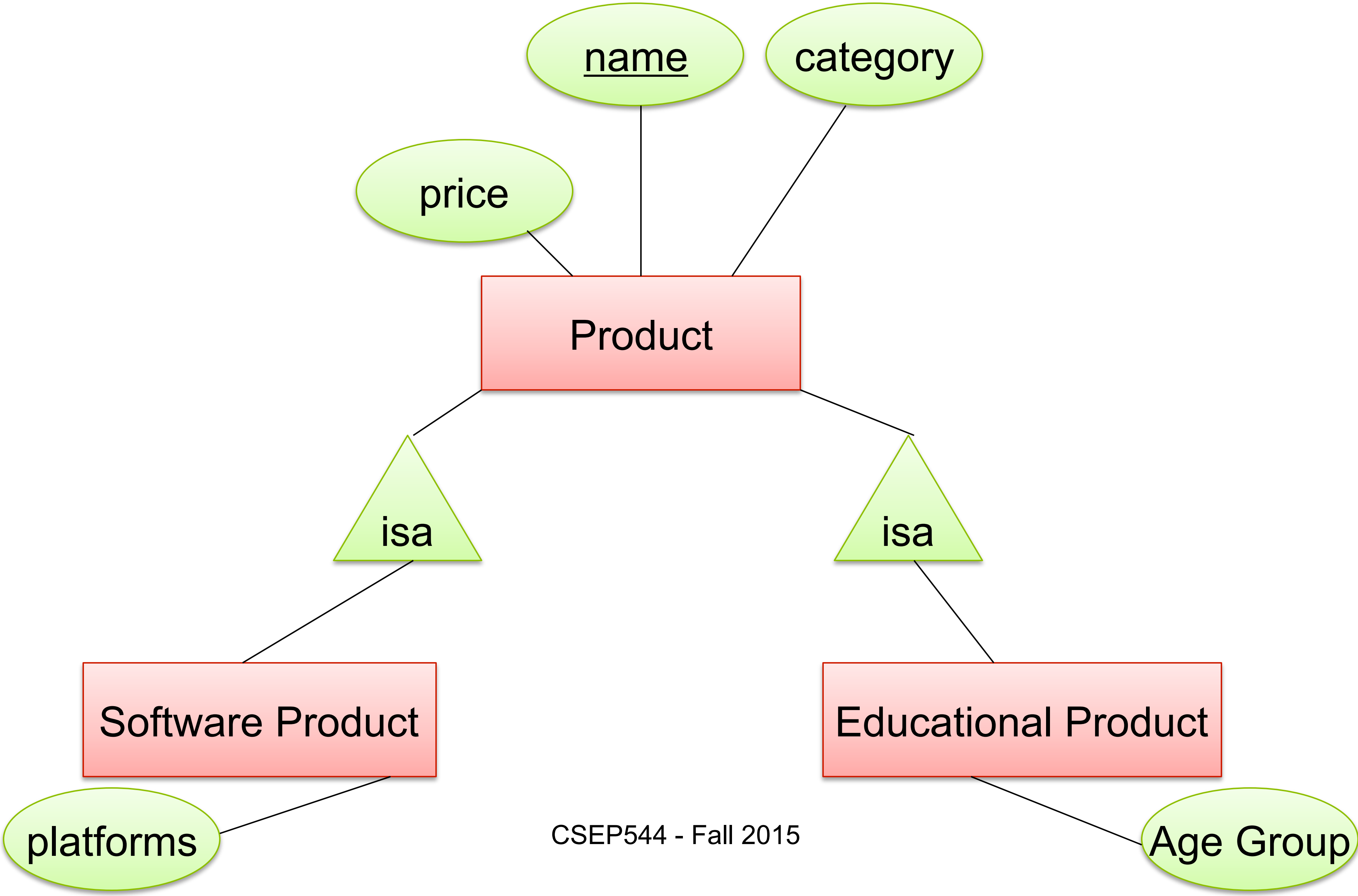
Products

Software products

Educational products

So --- we define subclasses in E/R

Subclasses


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

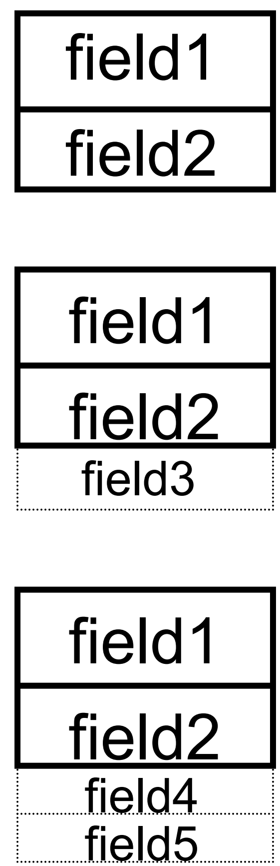
Understanding Subclasses

Think in terms of records:

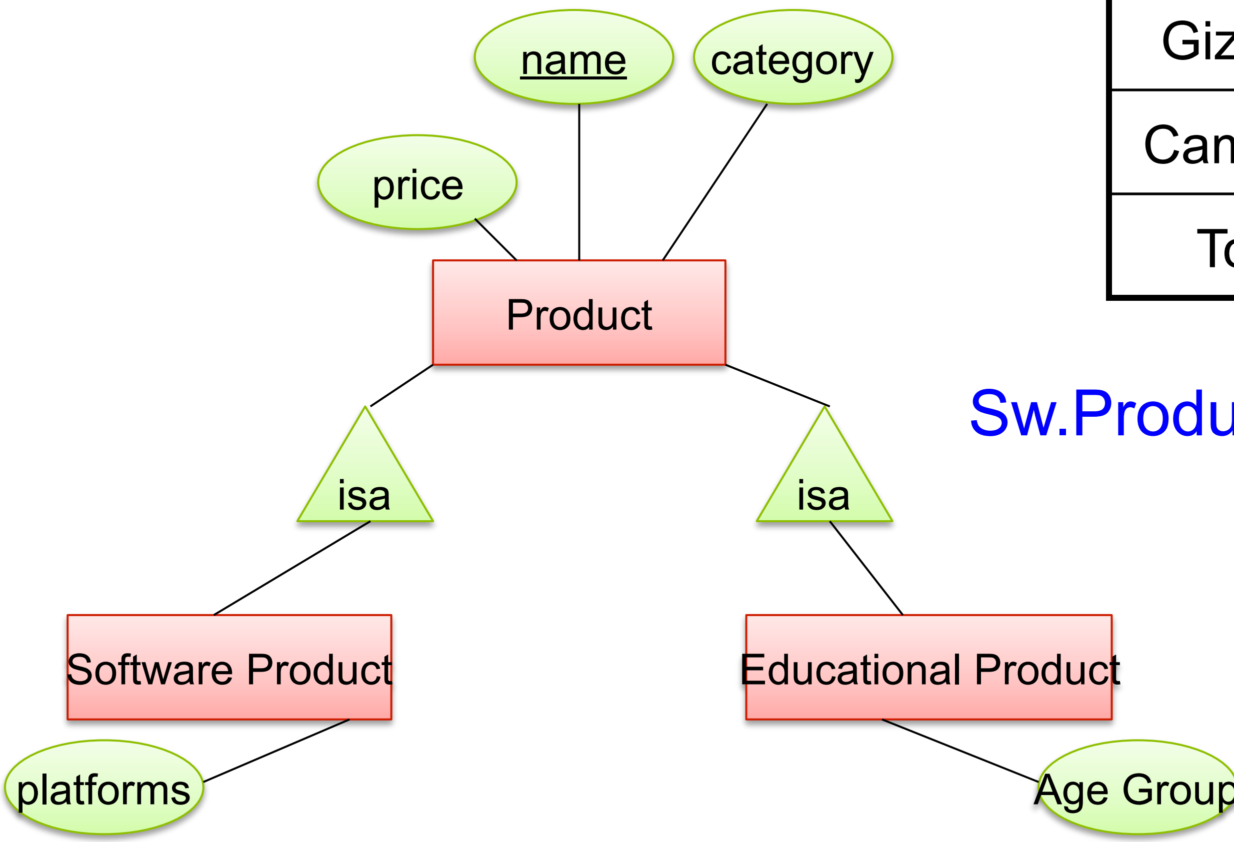
Product

SoftwareProduct

EducationalProduct



Subclasses to Relations



Product

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

Sw.Product

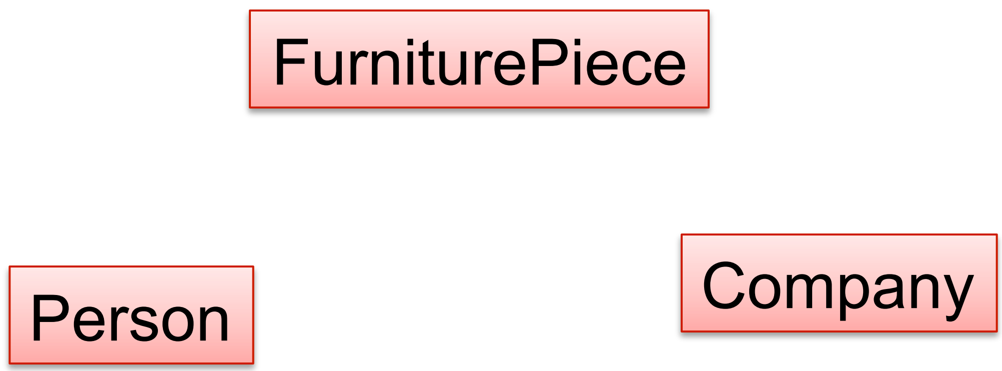
Name	platforms
Gizmo	unix

Ed.Product

Name	Age Group
Gizmo	toddler
Toy	retired

Other ways to convert are possible

Modeling Union Types With Subclasses

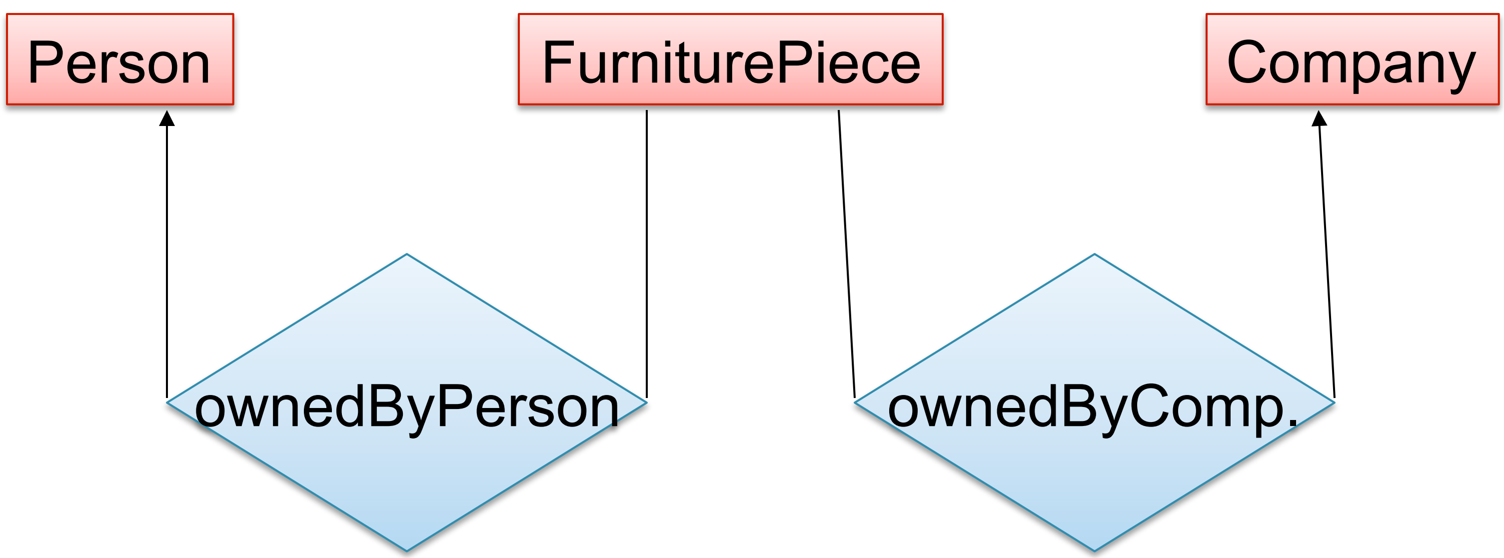


Say: each piece of furniture is owned either by a person or by a company

Modeling Union Types With Subclasses

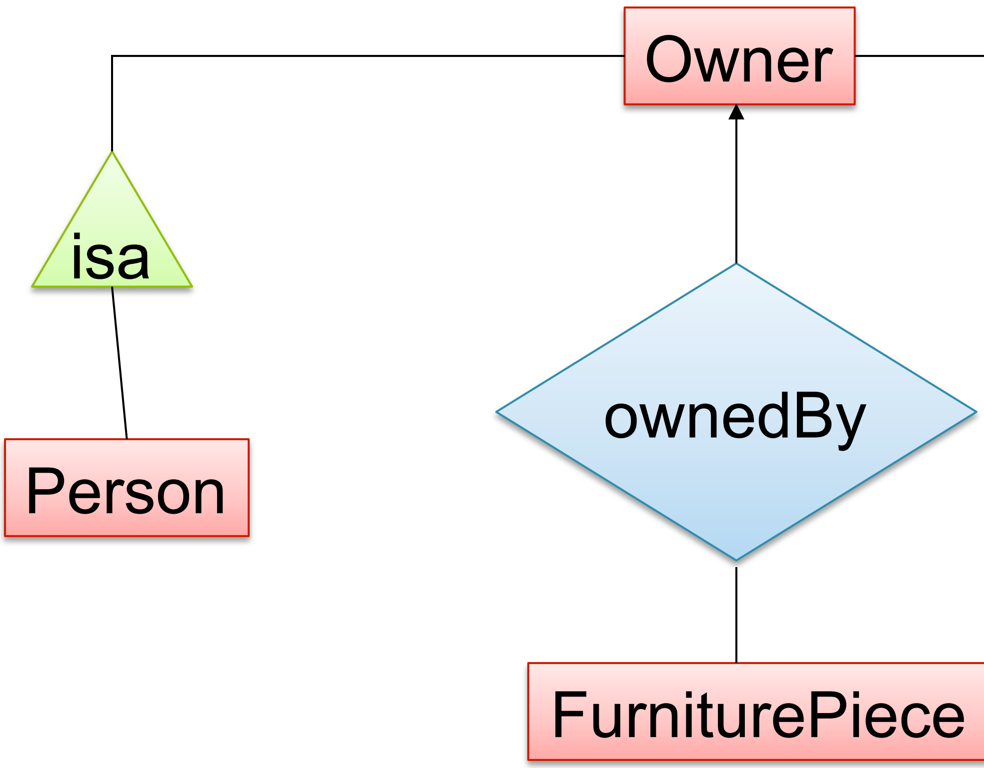
Say: each piece of furniture is owned either by a person or by a company

Solution 1. Acceptable but imperfect (What's wrong ?)



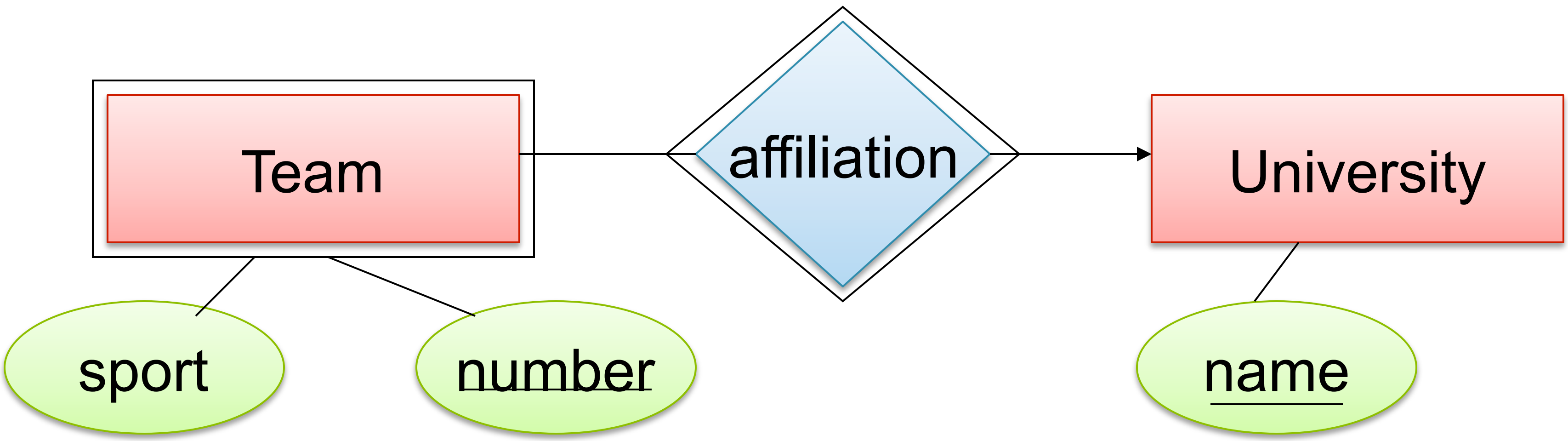
Modeling Union Types With Subclasses

Solution 2: better, more laborious



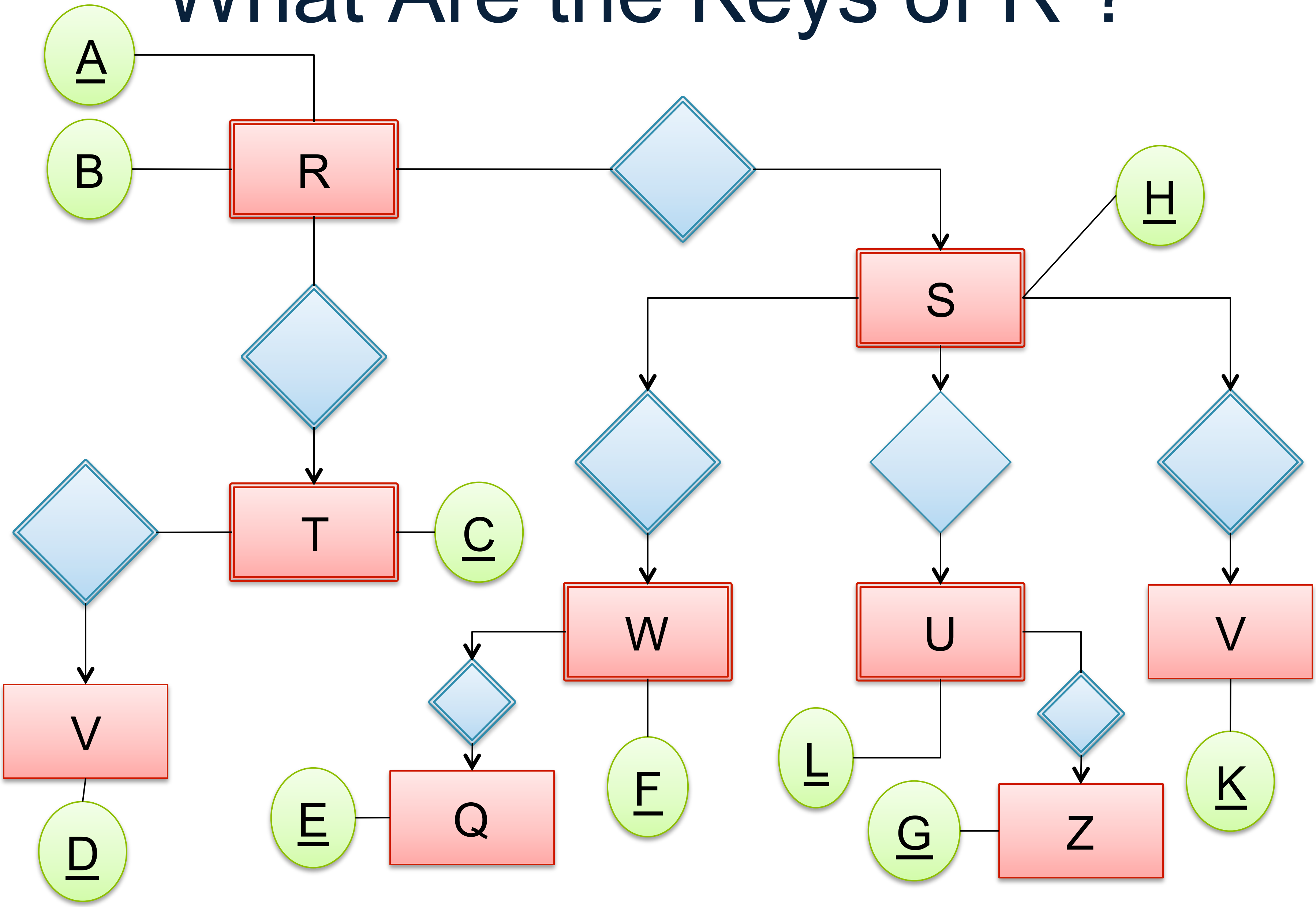
Weak Entity Sets

Entity sets are weak when their key comes from other classes to which they are related.



Team(sport, number, universityName)
University(name)

What Are the Keys of R ?



Constraints in E/R Diagrams

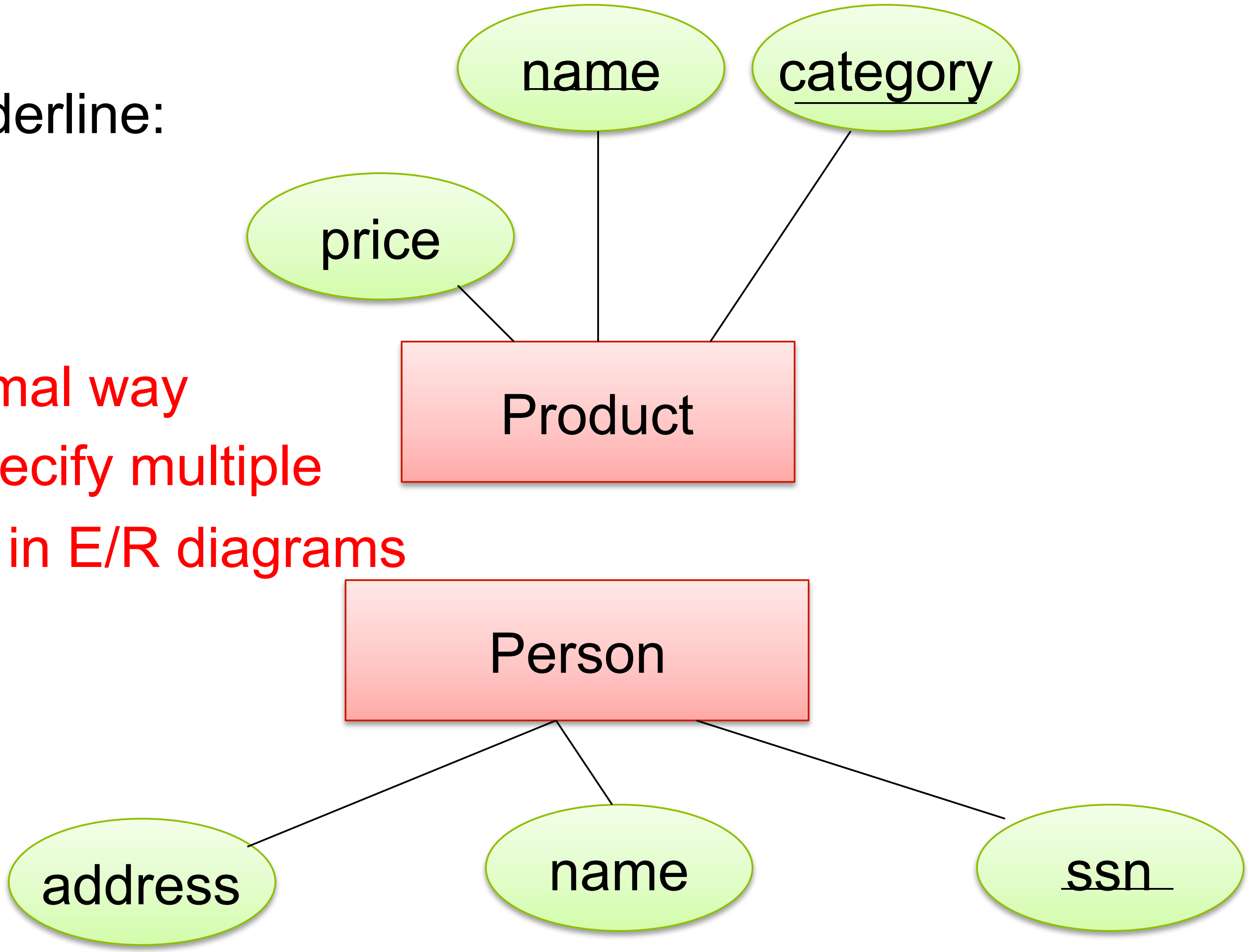
•Finding constraints is part of the modeling process.
•Commonly used constraints:

- Keys: social security number uniquely identifies a person.
- Single-value constraints: a person can have only one father.
- Referential integrity constraints: if you work for a company, it
- must exist in the database.
- Other constraints: peoples' ages are between 0 and 150.

Keys in E/R Diagrams

Underline:

No formal way
to specify multiple
keys in E/R diagrams

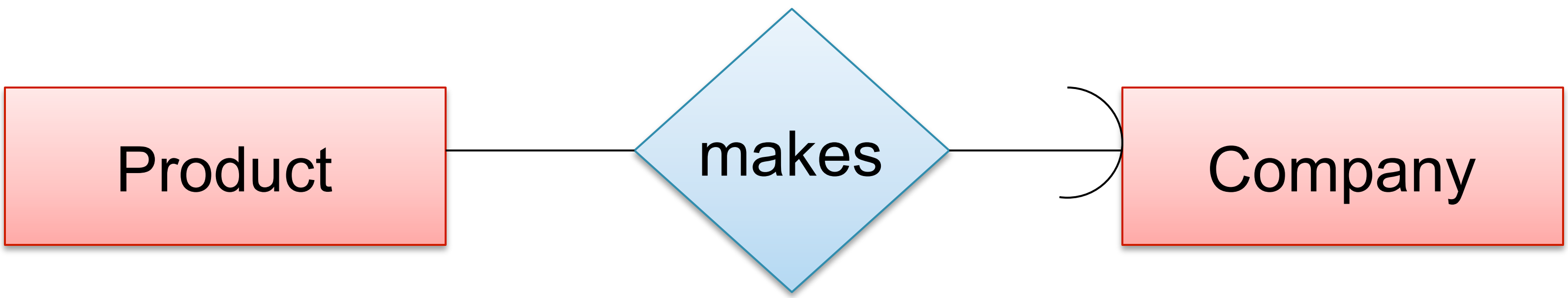


Single Value Constraints

v. s.

Referential Integrity Constraints

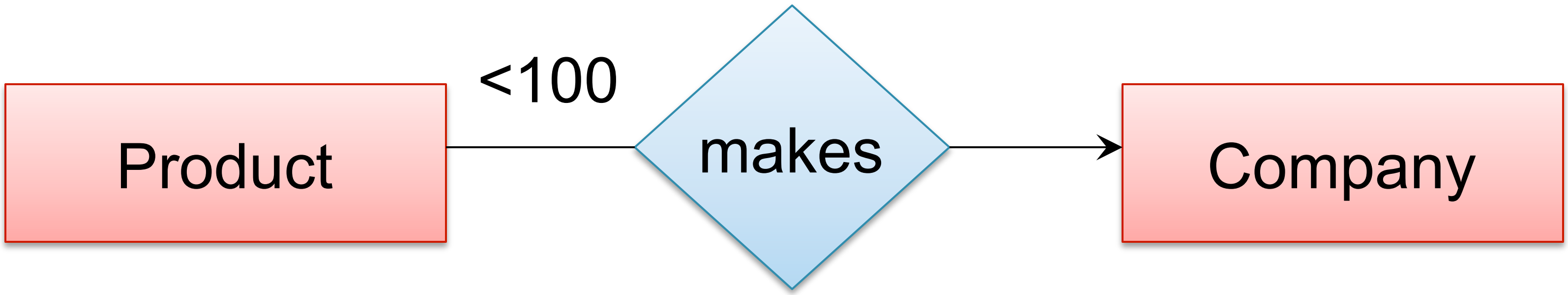
Each product made by at most one company.
Some products made by no company



Each product made by *exactly* one company.

Note: For weak entity sets → should be replaced by
(sec 4.4.2)

Other Constraints



Q: What does this mean ?

A: A Company entity cannot be connected

by relationship to more than 99 Product entities

Note: For "at least one", you can use "≥ 1" in a many-many relationship

Database Design Summary

- Conceptual modeling = design the database schema
  - Usually done with Entity-Relationship diagrams
  - It is a form of documentation the database schema; it is not executable code
  - Straightforward conversion to SQL tables
  - Big problem in the real world: the SQL tables are updated, the E/R documentation is not maintained
- Schema refinement using normal forms
  - Functional dependencies, normalization

Outline

- Relational Query Languages
- Database Design:
  - On your own: slides and/or Chapters 2, 3
  - In class: *What goes around*
- Functional Dependencies and BCNF

Data Models

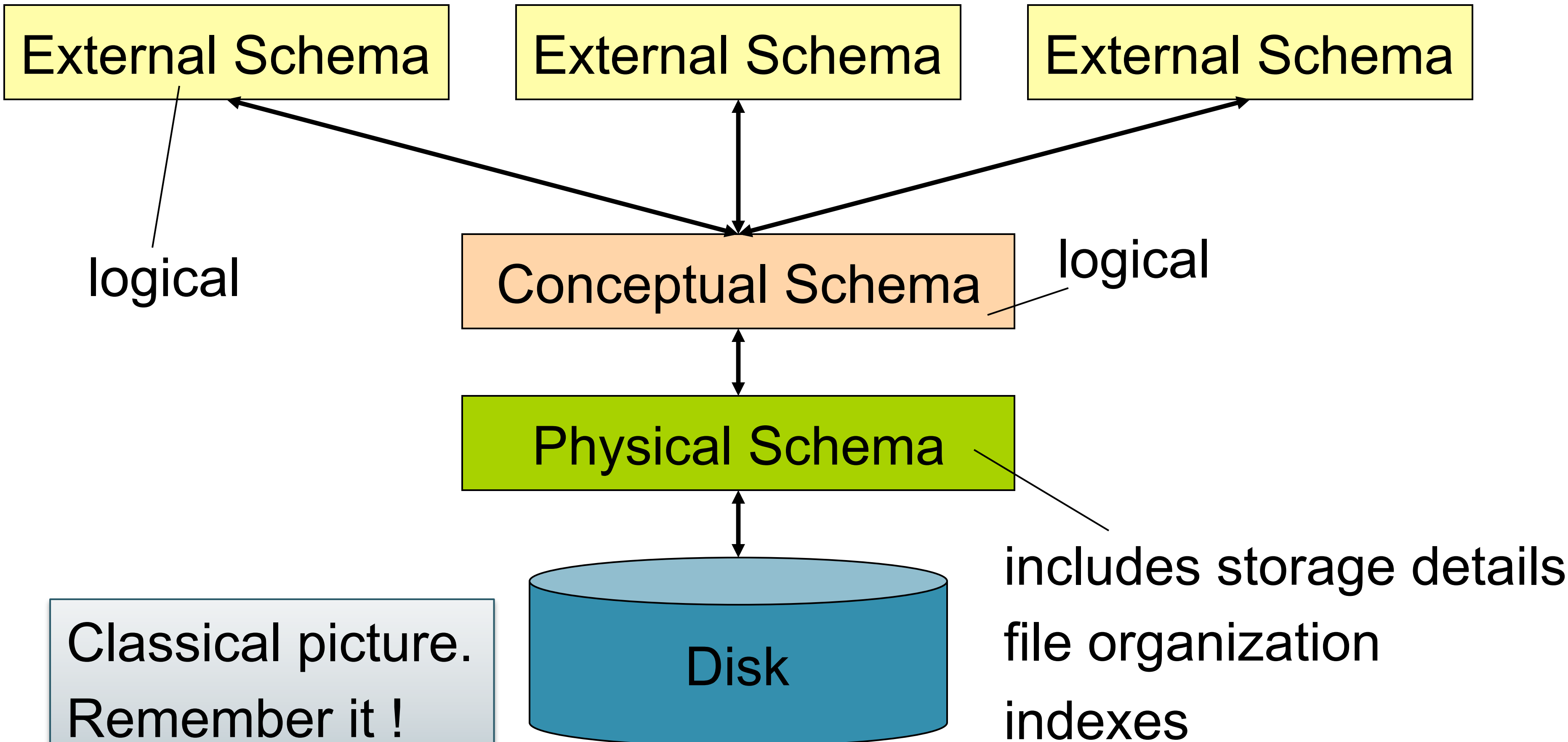
- M. Stonebraker and J. Hellerstein. What Goes Around Comes Around. In "Readings in Database Systems" (aka the Red Book). 4th ed.

"Data Model"

- Apps need to model real-world data
  - Typically includes entities and relationships between them
  - Entities: e.g. students, courses, products, clients
  - Relationships: e.g. course registrations, product purchases

- Data model enables a user to define the data using high-level constructs without worrying about many low-level details of how data will be stored on disk

Levels of Abstraction



Classical picture.
Remember it !

What goes around…

- **Structured data**
  - What is this ? Examples ?
- **Semistructured data**
  - What is this ?
  - Examples ?
- **Unstructured data**
  - What is this ? Examples ?

What goes around…

- **Structured data**
  - All data conforms to a schema. Ex: business data
- **Semistructured data**
  - Some structure in the data but implicit and irregular
  - Ex: resume, ads
- **Unstructured data**
  - No structure in data. Ex: text, sound, video, images

- In our class: structured data & relational DBMSs

Early Proposal 1: IMS

- What is it ?

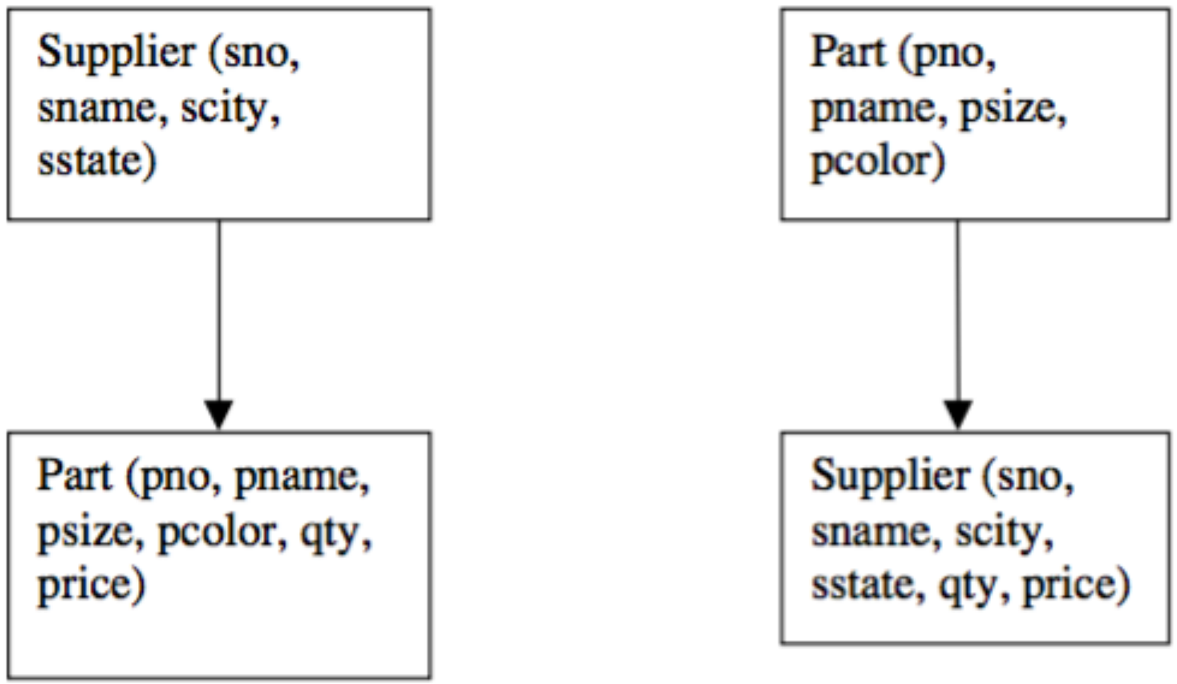
Early Proposal 1: IMS

- **Hierarchical data model**

- **Record**
  - **Type**: collection of named fields with data types (+)
  - **Instance**: must match type definition (+)
  - Each instance must have a **key** (+)
  - Record types must be arranged in a **tree** (-)

- **IMS database** is collection of instances of record types organized in a tree

IMS Example



Two Hierarchical Organizations
Figure 2

DL/1

- How does a programmer retrieve data in IMS ?

DL/1

- Each record has a hierarchical sequence key (HSK)
  - Records are totally ordered: depth-first and left-to-right
- HSK defines semantics of commands:
  - get_next
  - get_next_within_parent
- **DL/1 is a record-at-a-time language**
  - Programmer constructs an algorithm for solving the query
  - Programmer must worry about query optimization

Data storage

- How is the data physically stored in IMS ?

Data storage

- Root records
  - Stored sequentially (sorted on key)
  - Indexed in a B-tree using the key of the record
  - Hashed using the key of the record

- Dependent records
  - Physically sequential
  - Various forms of pointers

- Selected organizations restrict DL/1 commands
  - No updates allowed with sequential organization
  - No “get-next” for hashed organization

Data Independence

- What is it ?

Data Independence

- **Physical data independence**: Applications are insulated from changes in **physical storage details**

- **Logical data independence**: Applications are insulated from changes to **logical structure of the data**

IMS Limitations

- Tree-structured data model
  - Redundant data
  - Existence depends on parent

- Record-at-a-time user interface

- Very limited physical independence
  - Phys. organization limits possible operations
  - Application programs break if organization changes

- Provides some logical independence
  - DL/1 program runs on logical database
  - Difficult to achieve good logical data independence with a tree model

Early Proposal 2: CODASYL

- What is it ?

Early Proposal 2: CODASYL

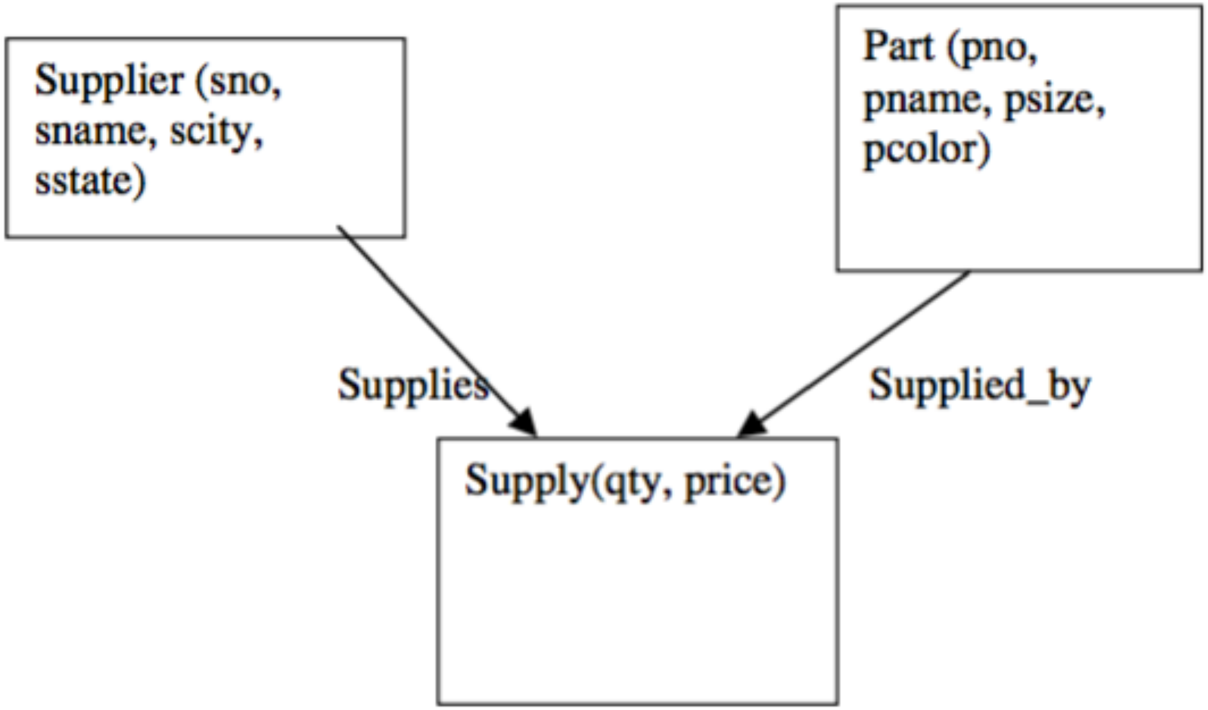
- **Networked data model**

- **Record types** with **keys** (+)

- Organized in a **network**
  - More flexible than hierarchy (+)
  - A record can have multiple parents (-)
  - Arcs between records are named
  - At least one entry point to the network

- **Record-at-a-time** DML (-)

CODASYL Example



A CODASYL Network
Figure 5

CODASYL Limitations

- No physical data independence
- No logical data independence
- Very complex:
  - Programs must “navigate the hyperspace”
  - Load and recover as one gigantic object

Relational Model Overview

- Proposed by Ted Codd in 1970
- Motivation: better logical and physical data independence

Relational Model Overview

- Defines logical data model
- No physical data model
- Set-at-a-time query language

Great Debate

- Pro relational
  - What where the arguments ?
- Against relational
  - What where the arguments ?
- How was it settled ?

Great Debate

- Pro relational
  - CODASYL is too complex
  - CODASYL does not provide sufficient data independence
  - Record-at-a-time languages are too hard to optimize
  - Trees/networks not flexible enough for common cases
- Against relational
  - COBOL programmers cannot understand relational languages
  - Impossible to represent the relational model efficiently
  - CODASYL can represent tables
- Ultimately settled by the market place

Other Data Models

- Entity-Relationship: 1970's
  - Successful in logical database design (this lecture + hw2)
- Extended Relational: 1980's
- Semantic: late 1970's and 1980's

- Object-oriented: late 1980's and early 1990's
  - Impedance mismatch: relational dbs ←→ OO languages
  - Interesting but ultimately failed (several reasons, see paper)
- Object-relational: late 1980's and early 1990's
  - User-defined types, ops, functions, and access methods
- Semi-structured: late 1990's to the present
  - XML, JSon, Protobuf

Outline

- Relational Query Languages
- Database Design:
  - On your own: slides and/or Chapters 2, 3
  - In class: *What goes around*
- Functional Dependencies and BCNF

Relational Schema Design

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield

One person may have multiple phones, but lives in only one city

Primary key is thus (SSN,PhoneNumber)

What is the problem with this schema?

Relational Schema Design

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield

Anomalies:

- Redundancy = repeat data
- Update anomalies = what if Fred moves to “Bellevue”?
- Deletion anomalies = what if Joe deletes his phone number?

Relation Decomposition

Break the relation into two:

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield

Name	SSN	City
Fred	123-45-6789	Seattle
Joe	987-65-4321	Westfield

SSN	PhoneNumber
123-45-6789	206-555-1234
123-45-6789	206-555-6543
987-65-4321	908-555-2121

Anomalies have gone:

No more repeated data

Easy to move Fred to “Bellevue” (how ?)

Easy to delete all Joe’s phone numbers (how ?)

Relational Schema Design (or Logical Design)

How do we do this systematically?

Start with some relational schema

Find out its ***<u>functional dependencies</u>*** (FDs)

Use FDs to ***<u>normalize</u>*** the relational schema

Functional Dependencies (FDs)

Definition

If two tuples agree on the attributes

$$\boxed{A_1, A_2, \ldots, A_n}$$

then they must also agree on the attributes

$$\boxed{B_1, B_2, \ldots, B_m}$$

Formally:

$A_1 \ldots A_n$ **determines** $B_1..B_m$

$$\boxed{A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m}$$

Functional Dependencies (FDs)

Definition $A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$ **holds** in R if:

$\forall t, t' \in R,$

$(t.A_1 = t'.A_1 \wedge \ldots \wedge t.A_m = t'.A_m \Rightarrow t.B_1 = t'.B_1 \wedge \ldots \wedge t.B_n = t'.B_n)$

R		A_1	...	A_m		B_1	...	B_n		
t										
t'										

if t, t' agree here then t, t' agree here

Example

An FD holds, or does not hold on an instance:

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

EmpID → Name, Phone, Position

Position → Phone

but not Phone → Position

Example

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876 ←	Salesrep
E1111	Smith	9876 ←	Salesrep
E9999	Mary	1234	Lawyer

Position → Phone

Example

EmpID	Name	Phone	Position
E0045	Smith	1234 →	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234 →	Lawyer

But not Phone → Position

Example

name → color
category → department
color, category → price

name	category	color	department	price
Gizmo	Gadget	Green	Toys	49
Tweaker	Gadget	Green	Toys	99

Do all the FDs hold on this instance?

Example

name → color
category → department
color, category → price

name	category	color	department	price
Gizmo	Gadget	Green	Toys	49
Tweaker	Gadget	Black	Toys	99
Gizmo	Stationary	Green	Office-supp.	59

What about this one ?

Terminology

FD **holds** or **does not hold** on an instance

If we can be sure that *every instance of R* will be one in which a given FD is true, then we say that **R satisfies the FD**

If we say that R satisfies an FD F, we are **stating a constraint on R**

An Interesting Observation

If all these FDs are true:

name → color
category → department
color, category → price

Then this FD also holds:

name, category → price

If we find out from application domain that a relation satisfies some FDs, it doesn't mean that we found all the FDs that it satisfies!
There could be more FDs implied by the ones we have.

Closure of a set of Attributes

Given a set of attributes $A_1, \ldots, A_n$

The **closure**, $\{A_1, \ldots, A_n\}^+$ = the set of attributes B s.t. $A_1, \ldots, A_n \rightarrow B$

Example:

1. name → color
2. category → department
3. color, category → price

Closures:

name$^+$ = {name, color}

{name, category}$^+$ = {name, category, color, department, price}

color$^+$ = {color}

Closure Algorithm

X={A1, …, An}.

Repeat until X doesn't change **do**:

if $B_1, \ldots, B_n \rightarrow C$ is a FD **and**

$B_1, \ldots, B_n$ are all in X

then add C to X.

Example:

1. name → color
2. category → department
3. color, category → price

$\{name, category\}^+ =$

{ }

Closure Algorithm

X={A1, …, An}.

Repeat until X doesn't change **do**:

if $B_1, \ldots, B_n \rightarrow C$ is a FD **and**

$B_1, \ldots, B_n$ are all in X

then add C to X.

Example:

1. name → color
2. category → department
3. color, category → price

$\{name, category\}^+ =$

$\{$ name, category, color, department, price $\}$

Closure Algorithm

X={A1, …, An}.

Repeat until X doesn't change **do**:

if $B_1, \ldots, B_n \rightarrow C$ is a FD **and**

$B_1, \ldots, B_n$ are all in X

then add C to X.

Example:

1. name → color
2. category → department
3. color, category → price

$\{$name, category$\}^+$ =

{ name, category, color, department, price }

Hence: name, category → color, department, price

Example

In class:

R(A,B,C,D,E,F)

$A, B \rightarrow C$
$A, D \rightarrow E$
$B \rightarrow D$
$A, F \rightarrow B$

Compute $\{A,B\}^+$ X = {A, B, }

Compute $\{A, F\}^+$ X = {A, F, }

Example

In class:

R(A,B,C,D,E,F)

A, B	$\rightarrow$	C
A, D	$\rightarrow$	E
B	$\rightarrow$	D
A, F	$\rightarrow$	B

Compute $\{A,B\}^+$ X = {A, B, C, D, E }

Compute $\{A, F\}^+$ X = {A, F, }

Example

In class:

R(A,B,C,D,E,F)

A, B	$\rightarrow$	C
A, D	$\rightarrow$	E
B	$\rightarrow$	D
A, F	$\rightarrow$	B

Compute $\{A,B\}^+$ X = {A, B, C, D, E }

Compute $\{A, F\}^+$ X = {A, F, B, C, D, E }

Example

In class:

R(A,B,C,D,E,F)

A, B	$\rightarrow$	C
A, D	$\rightarrow$	E
B	$\rightarrow$	D
A, F	$\rightarrow$	B

Compute $\{A,B\}^+$ X = {A, B, C, D, E }

Compute $\{A, F\}^+$ X = {A, F, B, C, D, E }

What is the key of R?

Practice at Home

Find all FD's implied by:

$A, B \rightarrow C$
$A, D \rightarrow B$
$B \rightarrow D$

Practice at Home

Find all FD's implied by:

$A, B \rightarrow C$
$A, D \rightarrow B$
$B \rightarrow D$

Step 1: Compute X^+, for every X:

A+ = A,   B+ = BD,   C+ = C,   D+ = D

AB+ =ABCD, AC+=AC, AD+=ABCD,
BC+=BCD, BD+=BD, CD+=CD

ABC+ = ABD+ = ACD^+ = ABCD (no need to compute– why ?)

BCD^+ = BCD,   ABCD+ = ABCD

Practice at Home

Find all FD's implied by:

$A, B \rightarrow C$
$A, D \rightarrow B$
$B \rightarrow D$

Step 1: Compute X^+, for every X:

$A+ = A$, $B+ = BD$, $C+ = C$, $D+ = D$

$AB+ = ABCD$, $AC+ = AC$, $AD+ = ABCD$,
$BC+ = BCD$, $BD+ = BD$, $CD+ = CD$

$ABC+ = ABD+ = ACD^+ = ABCD$ (no need to compute– why ?)

$BCD^+ = BCD$, $ABCD+ = ABCD$

Step 2: Enumerate all FD's $X \rightarrow Y$, s.t. $Y \subseteq X^+$ and $X \cap Y = \emptyset$:

$AB \rightarrow CD$, $AD \rightarrow BC$, $ABC \rightarrow D$, $ABD \rightarrow C$, $ACD \rightarrow B$

Keys

- A **superkey** is a set of attributes $A_1, ..., A_n$ s.t. for any other attribute B, we have $A_1, ..., A_n \rightarrow B$

- A **key** is a minimal superkey
  - A superkey and for which no subset is a superkey

Computing (Super)Keys

- For all sets X, compute X^+
- If X^+ = [all attributes], then X is a superkey
- Try only the minimal X's to get the keys

Example

Product(name, price, category, color)

name, category $\rightarrow$ price
category $\rightarrow$ color

What is the key ?

Example

Product(name, price, category, color)

name, category → price
category → color

What is the key ?

(name, category) + = { name, category, price, color }

Hence (name, category) is a key

Key or Keys ?

Can we have more than one key ?

Given R(A,B,C) define FD's s.t. there are two or more keys

Key or Keys ?

Can we have more than one key ?

Given R(A,B,C) define FD's s.t. there are two or more keys

$A \rightarrow B$ $B \rightarrow C$ $C \rightarrow A$	or	$AB \rightarrow C$ $BC \rightarrow A$	or	$A \rightarrow BC$ $B \rightarrow AC$

what are the keys here ?

Eliminating Anomalies

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield
Joe	987-65-4321	908-555-1234	Westfield

SSN → Name, City

What is the key?

Suggest a rule for decomposing the table to eliminate anomalies

Eliminating Anomalies

Main idea:

- X → A is OK if X is a (super)key

- X → A is not OK otherwise
  - Need to decompose the table, but how?

Boyce-Codd Normal Form

There are no "bad" FDs:

Definition. A relation R is in BCNF if:

Whenever $X \rightarrow B$ is a non-trivial dependency, then X is a superkey.

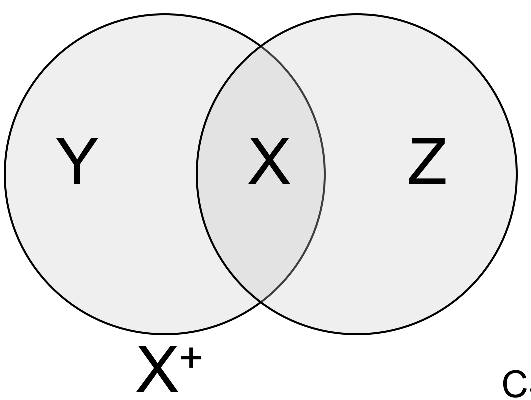
Equivalently:

Definition. A relation R is in BCNF if:

$\forall X$, either $X^+ = X$ or $X^+ =$ [all attributes]

BCNF Decomposition Algorithm

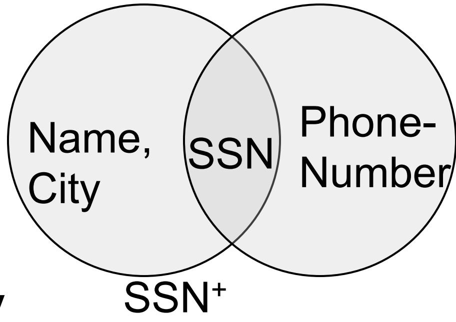
```
Normalize(R)
  find X s.t.: X ≠ X⁺ ≠ [all attributes]
  if (not found) then "R is in BCNF"
  let Y = X⁺ - X;    Z = [all attributes] - X⁺
  decompose R into R1(X ∪ Y) and R2(X ∪ Z)
  Normalize(R1); Normalize(R2);
```



Example

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield
Joe	987-65-4321	908-555-1234	Westfield

$\boxed{\text{SSN} \rightarrow \text{Name, City}}$



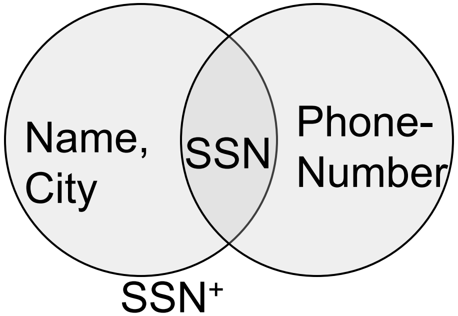
The only key is: {SSN, PhoneNumber}
Hence SSN → Name, City is a “bad” dependency

In other words:
SSN+ = Name, City and is neither SSN nor All Attributes

Example BCNF Decomposition

Name	**SSN**	**City**
Fred	123-45-6789	Seattle
Joe	987-65-4321	Westfield

SSN $\rightarrow$ Name, City



SSN	**PhoneNumber**
123-45-6789	206-555-1234
123-45-6789	206-555-6543
987-65-4321	908-555-2121
987-65-4321	908-555-1234

Let's check anomalies:

Redundancy ?

Update ?

Delete ?

Find X s.t.: $X \neq X^+ \neq$ [all attributes]

Example BCNF Decomposition

Person(name, SSN, age, hairColor, phoneNumber)

SSN → name, age

age → hairColor

Find X s.t.: $X \neq X^+ \neq$ [all attributes]

Example BCNF Decomposition

Person(name, SSN, age, hairColor, phoneNumber)

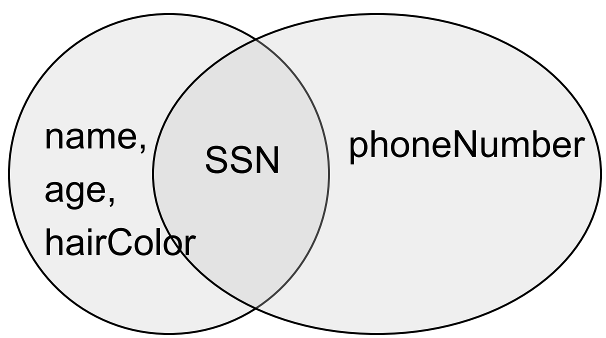
SSN → name, age

age → hairColor

Iteration 1: Person: SSN+ = SSN, name, age, hairColor

Decompose into: P(SSN, name, age, hairColor)

Phone(SSN, phoneNumber)



Find X s.t.: $X \neq X^+ \neq$ [all attributes]

Example BCNF Decomposition

Person(name, SSN, age, hairColor, phoneNumber)

SSN → name, age

age → hairColor

What are the keys ?

Iteration 1: Person: SSN+ = SSN, name, age, hairColor

Decompose into: P(SSN, name, age, hairColor)
Phone(SSN, phoneNumber)

Iteration 2: P: age+ = age, hairColor

Decompose: People(SSN, name, age)
Hair(age, hairColor)
Phone(SSN, phoneNumber)

Find X s.t.: X $\neq X^+ \neq$ [all attributes]

Example BCNF Decomposition

Person(name, SSN, age, hairColor, phoneNumber)

SSN → name, age

age → hairColor

Note the keys!

Iteration 1: Person: SSN+ = SSN, name, age, hairColor

Decompose into: P(<u>SSN</u>, name, age, hairColor)

Phone(<u>SSN, phoneNumber</u>)

Iteration 2: P: age+ = age, hairColor

Decompose: People(<u>SSN</u>, name, age)

Hair(<u>age</u>, hairColor)

Phone(<u>SSN, phoneNumber</u>)

R(A,B,C,D)

$A \rightarrow B$
$B \rightarrow C$

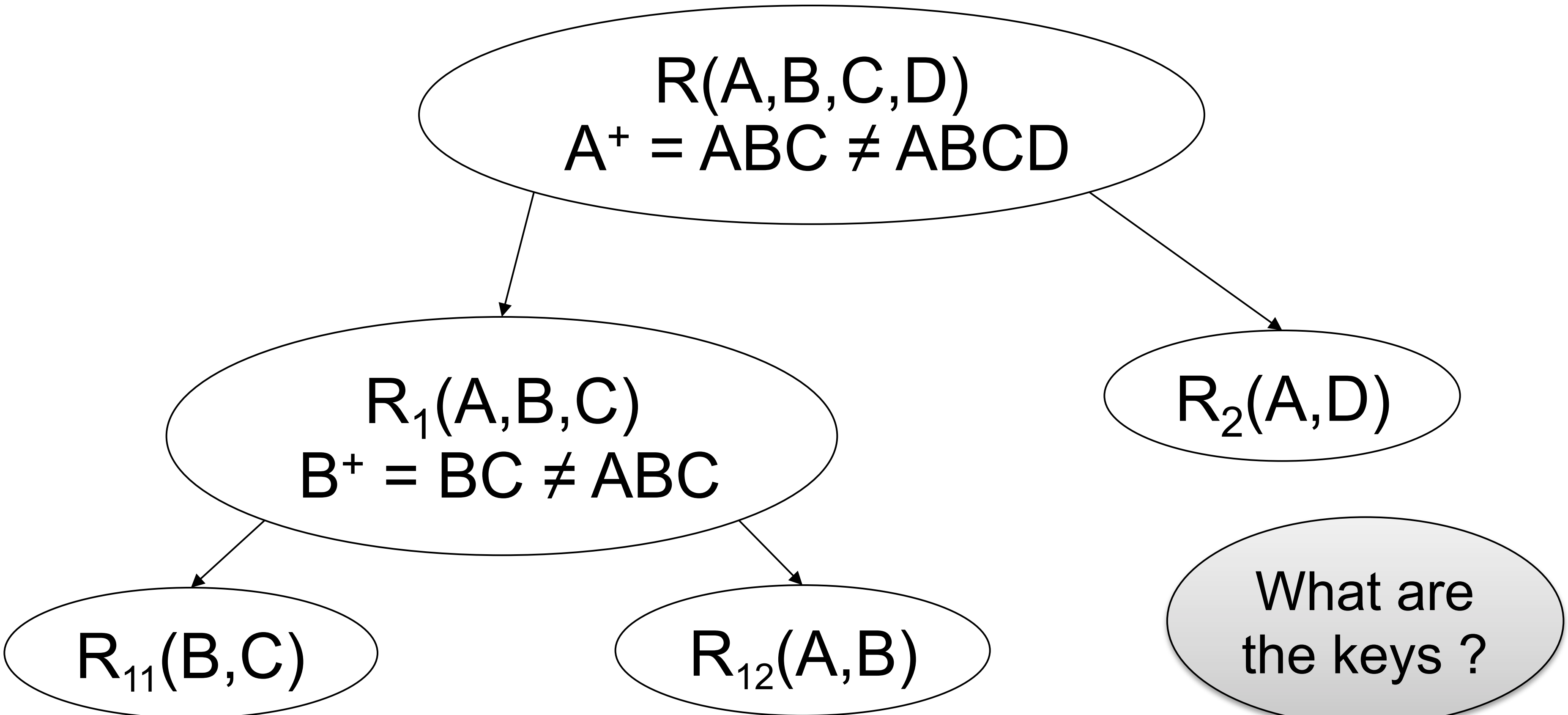
Practice at Home

R(A,B,C,D)
$A^+ = ABC \neq ABCD$

R(A,B,C,D)

$A \rightarrow B$
$B \rightarrow C$

Practice at Home



What happens if in R we first pick B^+ ? Or AB^+ ?

Schema Refinements = Normal Forms

- 1st Normal Form = all tables are flat
- 2nd Normal Form = obsolete
- Boyce Codd Normal Form = today
- 3rd Normal Form = see book