# CSE 344

## MAY 14TH – ENTITIES

# EXAMS

- **Scores**
- **Final grades**
- **Concerned?**
  - Email about meeting
- **Final Exam**
  - 35% of grade

# ADMINISTRIVIA

- **HW6 Due Wednesday**
- **OQ6 Out Wednesday**
- **HW7 Out Wednesday**
  - E/R + Normalization

# DATABASE DESIGN

**What it is:**

**Starting from scratch, design the database schema: relation, attributes, keys, foreign keys, constraints etc**

**Why it's hard**

**The database will be in operation for a very long time (years). Updating the schema while in production is very expensive (why?)**

# DATABASE DESIGN

**Consider issues such as:**

- What entities to model
- How entities are related
- What constraints exist in the domain

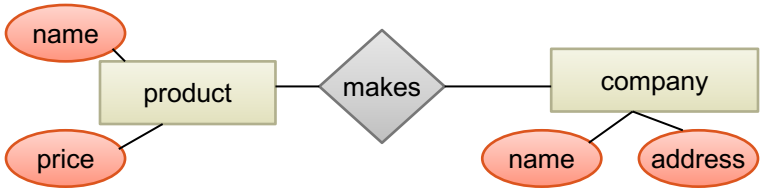
**Several formalisms exists**

- We discuss E/R diagrams
- UML, model-driven architecture

**Reading: Sec. 4.1-4.6**

# DATABASE DESIGN PROCESS

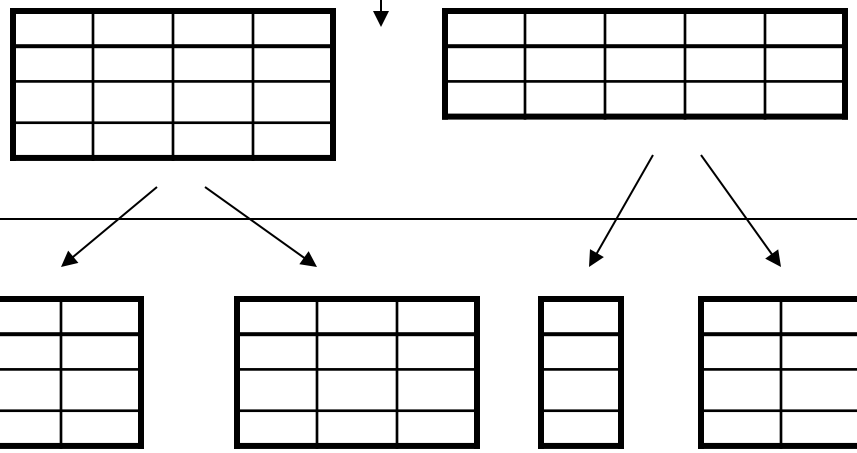
Conceptual Model:




Relational Model:

Tables + constraints

And also functional dep.



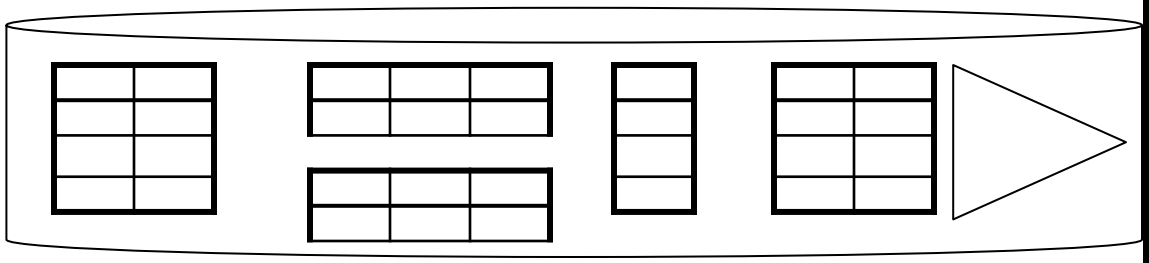
Normalization:

Eliminates anomalies

Conceptual Schema

Physical storage details

Physical Schema



# ENTITY / RELATIONSHIP DIAGRAMS

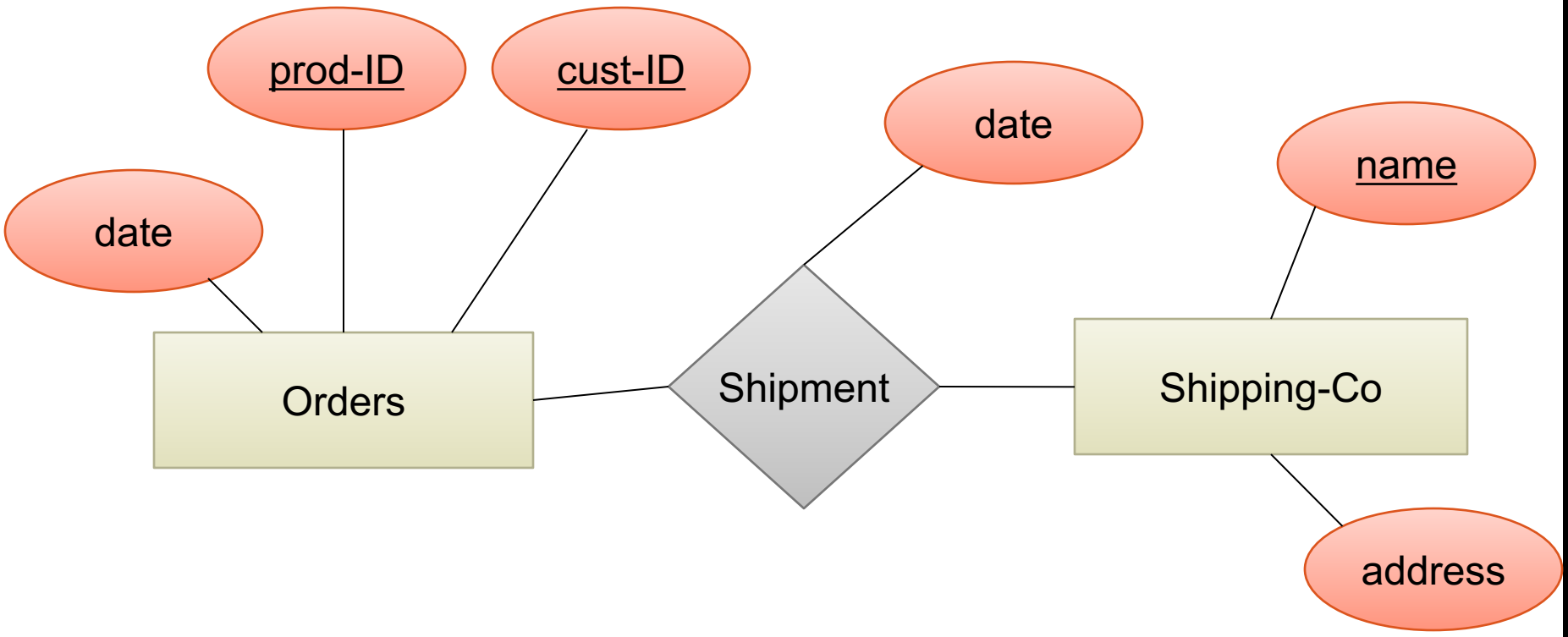
**Entity set = a class**

- An entity = an object


**Attribute**

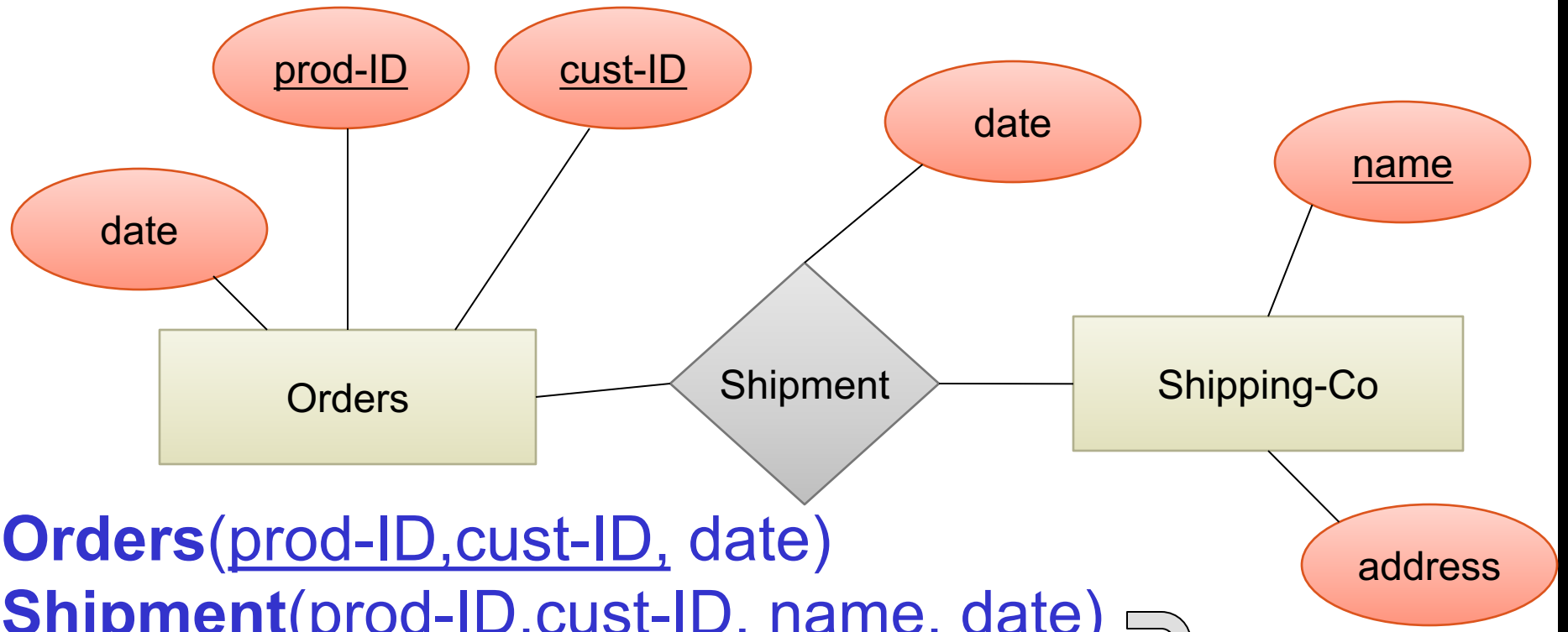

**Relationship**

# N-N RELATIONSHIPS TO RELATIONS




Represent this in relations

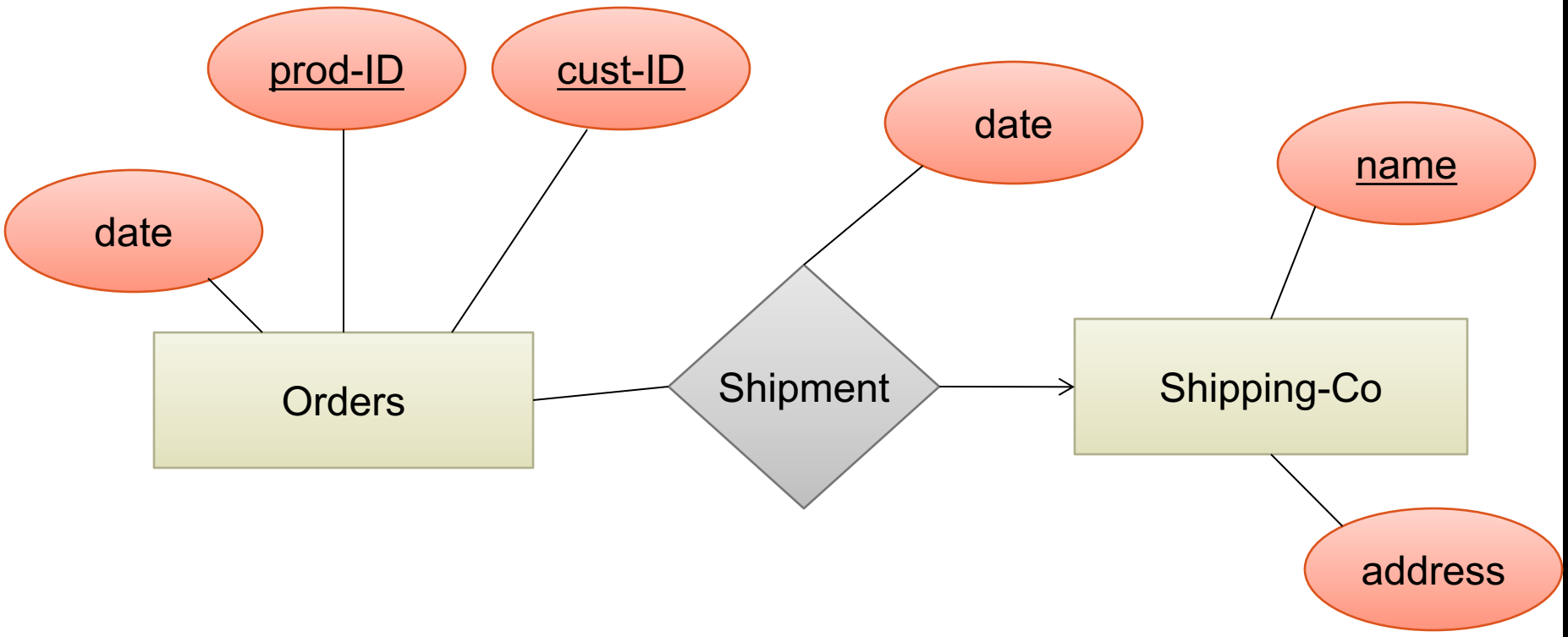
# N-N RELATIONSHIPS TO RELATIONS




**Orders**(prod-ID,cust-ID, date)
**Shipment**(prod-ID,cust-ID, name, date)
**Shipping-Co**(name, address)

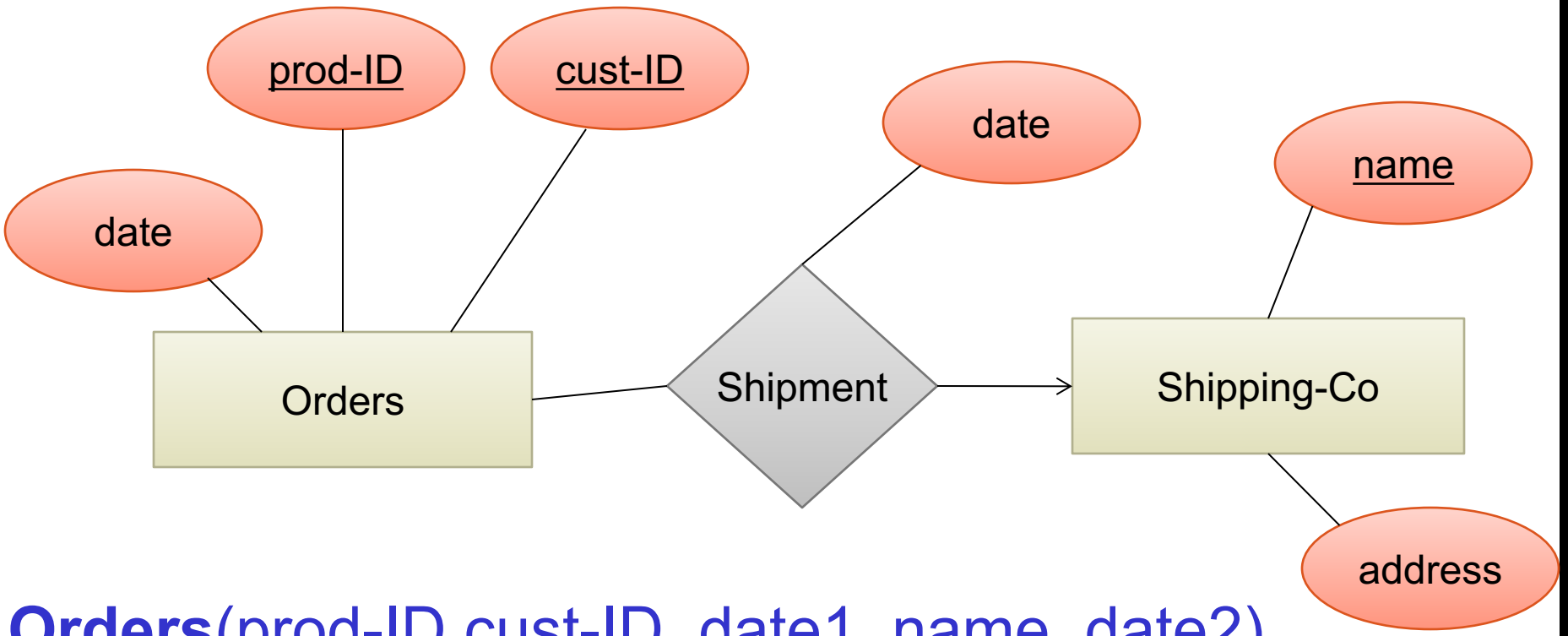
| prod-ID | cust-ID | name | date |
|---|---|---|---|
| Gizmo55 | Joe12 | UPS | 4/10/2011 |
| Gizmo55 | Joe12 | FEDEX | 4/9/2011 |

# N-1 RELATIONSHIPS TO RELATIONS




Represent this in relations

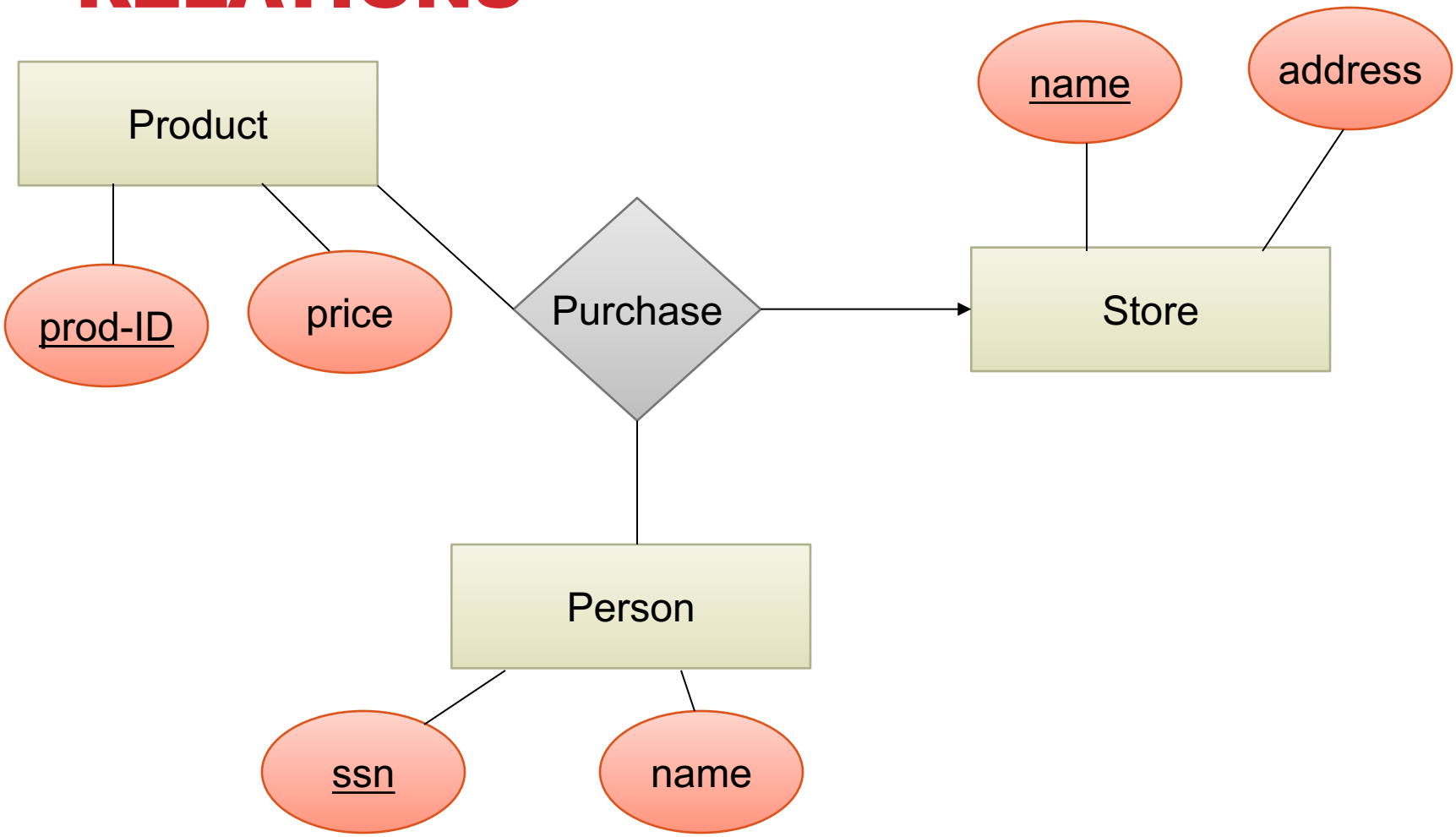
# N-1 RELATIONSHIPS TO RELATIONS




**Orders**(prod-ID,cust-ID, date1, name, date2)
**Shipping-Co**(name, address)

Remember: no separate relations for many-one relationship

# MULTI-WAY RELATIONSHIPS TO RELATIONS

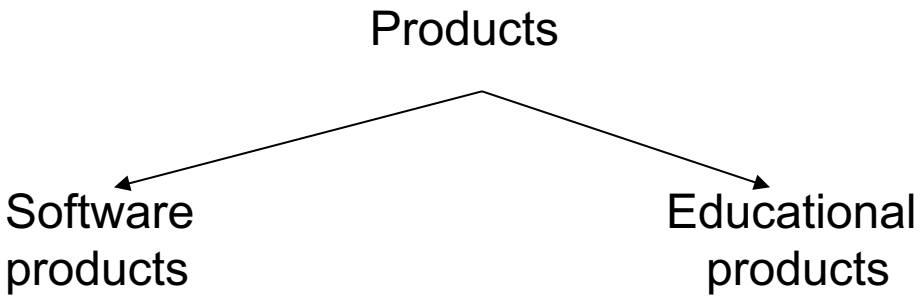



**Purchase**(prod-ID, ssn, name)

# MODELING SUBCLASSES

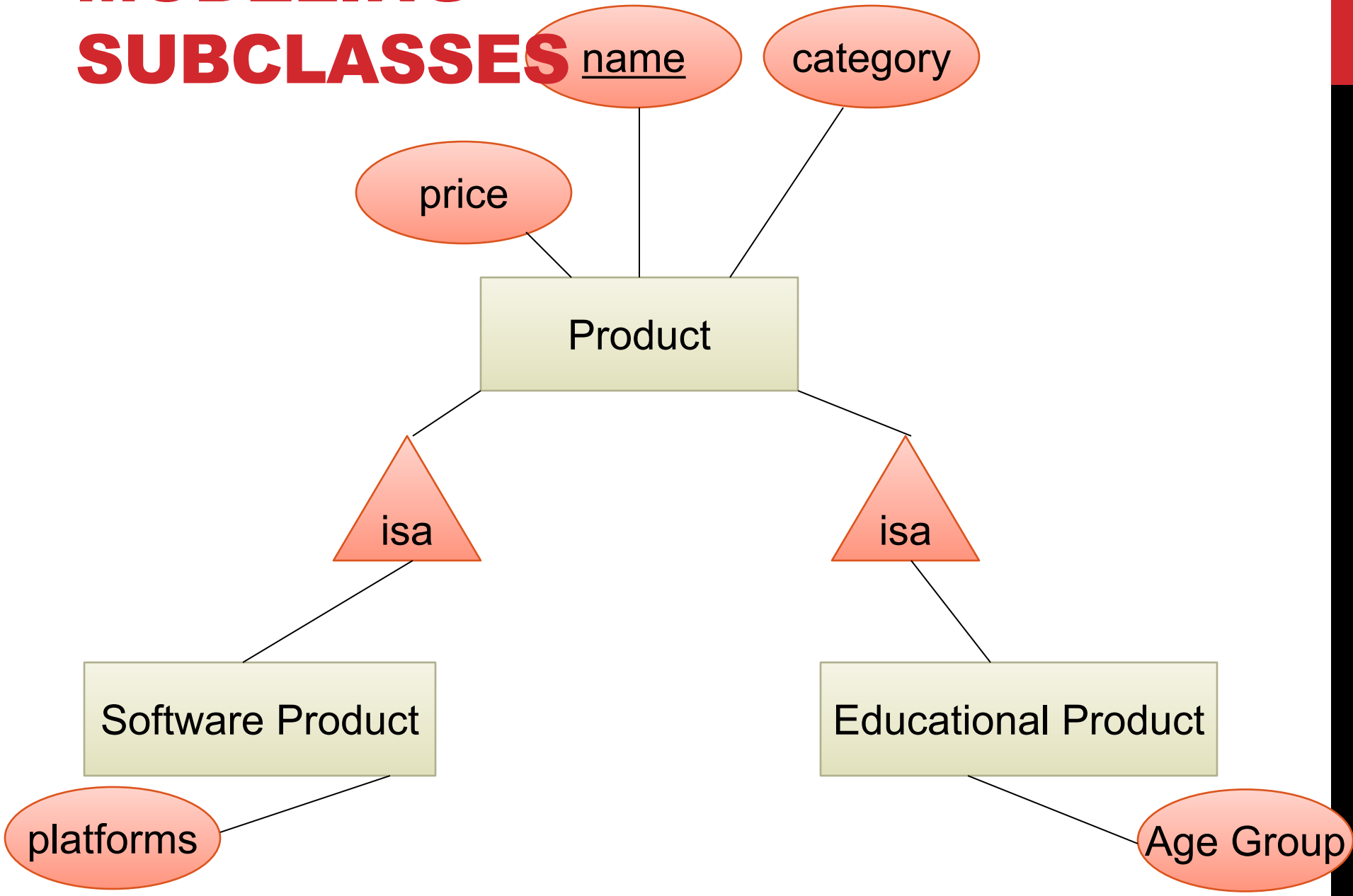
Some objects in a class may be special

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

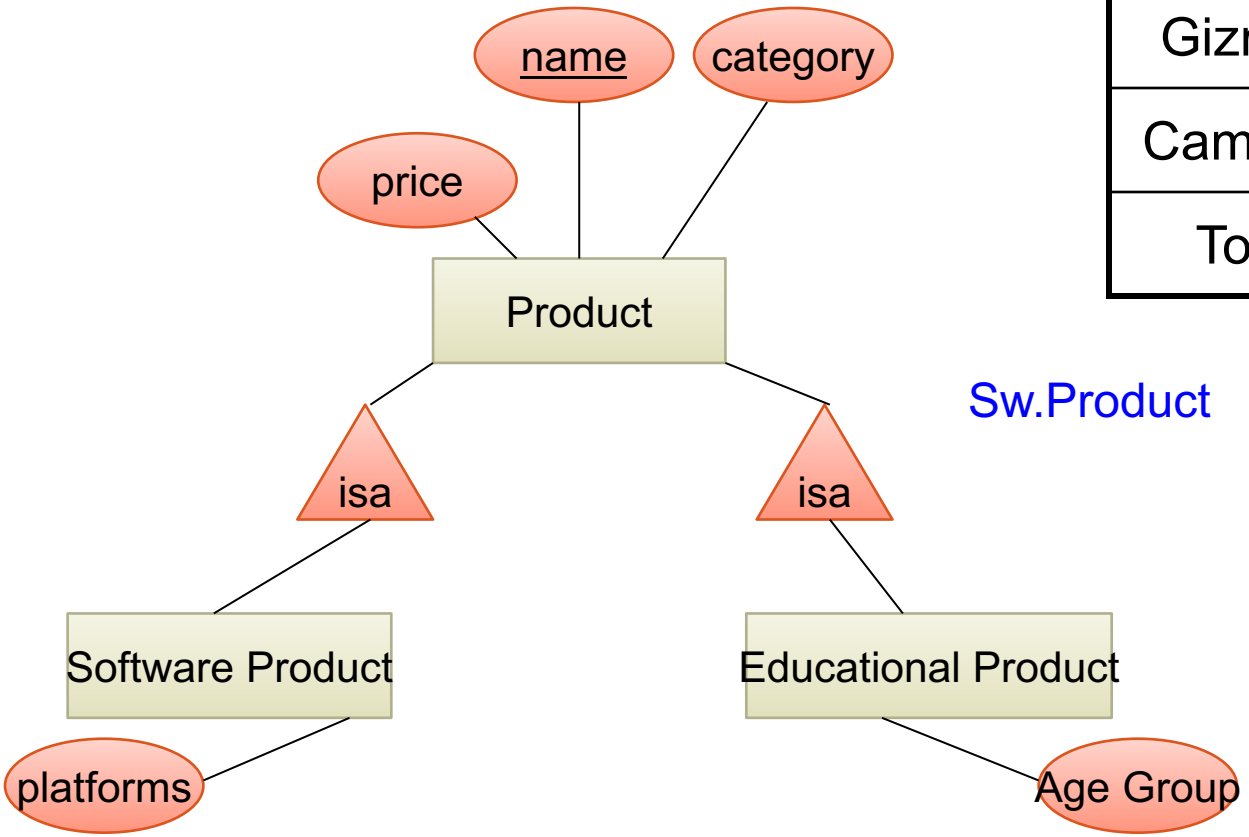



So --- we define subclasses in E/R

# MODELING SUBCLASSES



# MODELING SUBCLASSES




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 |

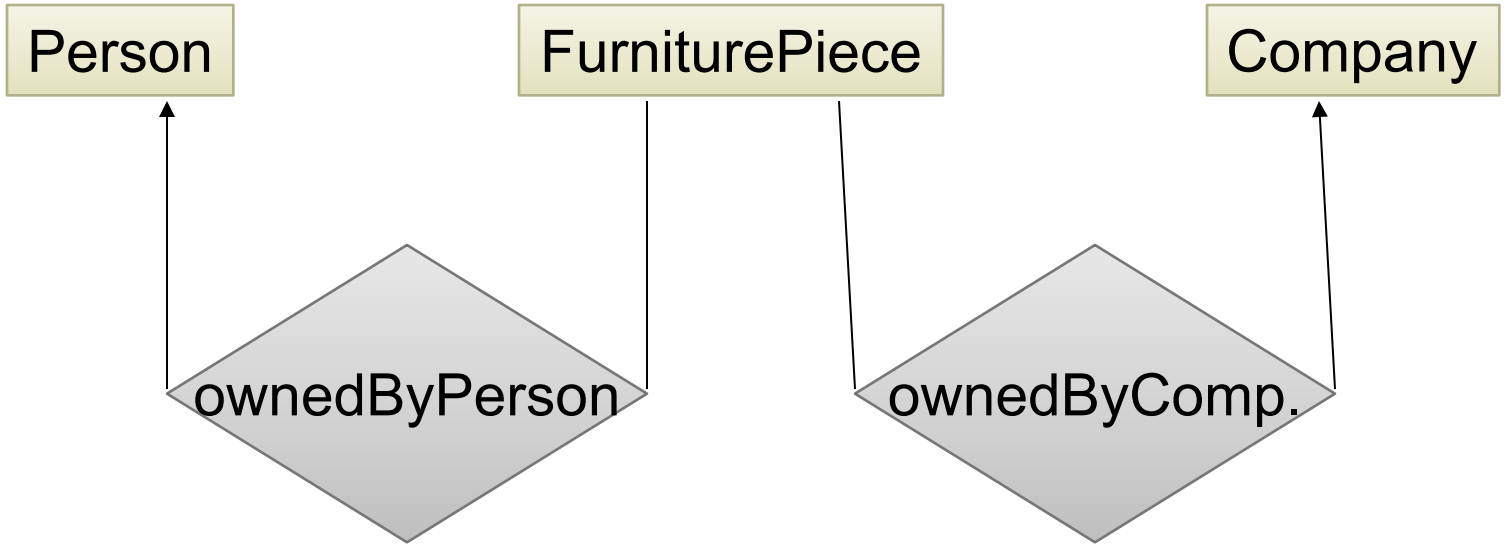
# MODELING UNION TYPES WITH SUBCLASSES


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

# MODELING UNION TYPES WITH SUBCLASSES

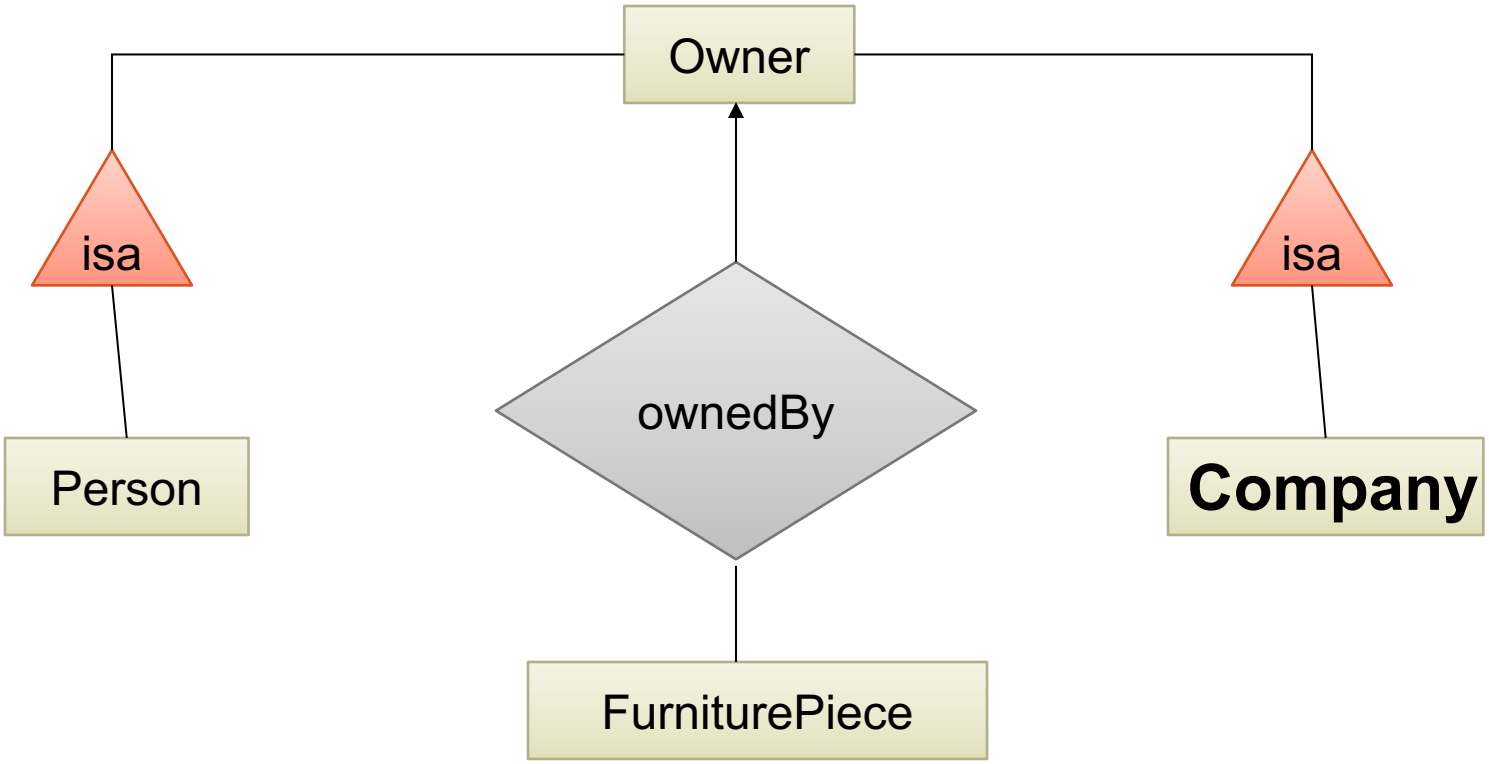
**Say: each piece of furniture is owned either by a person or by a company**

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



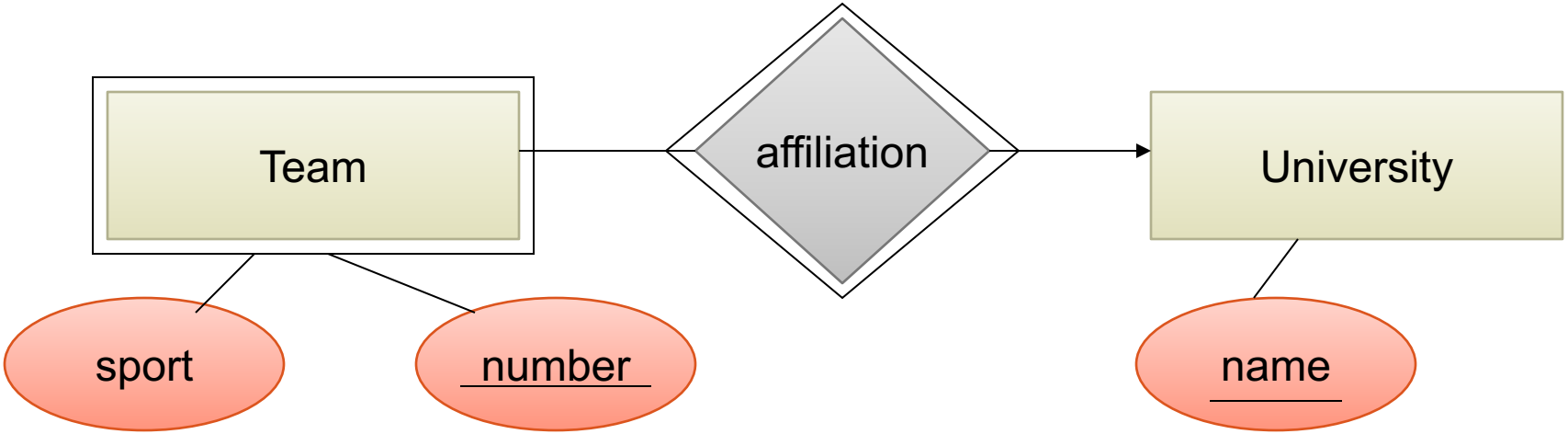
# MODELING UNION TYPES WITH SUBCLASSES

**Solution 2: better, more laborious**



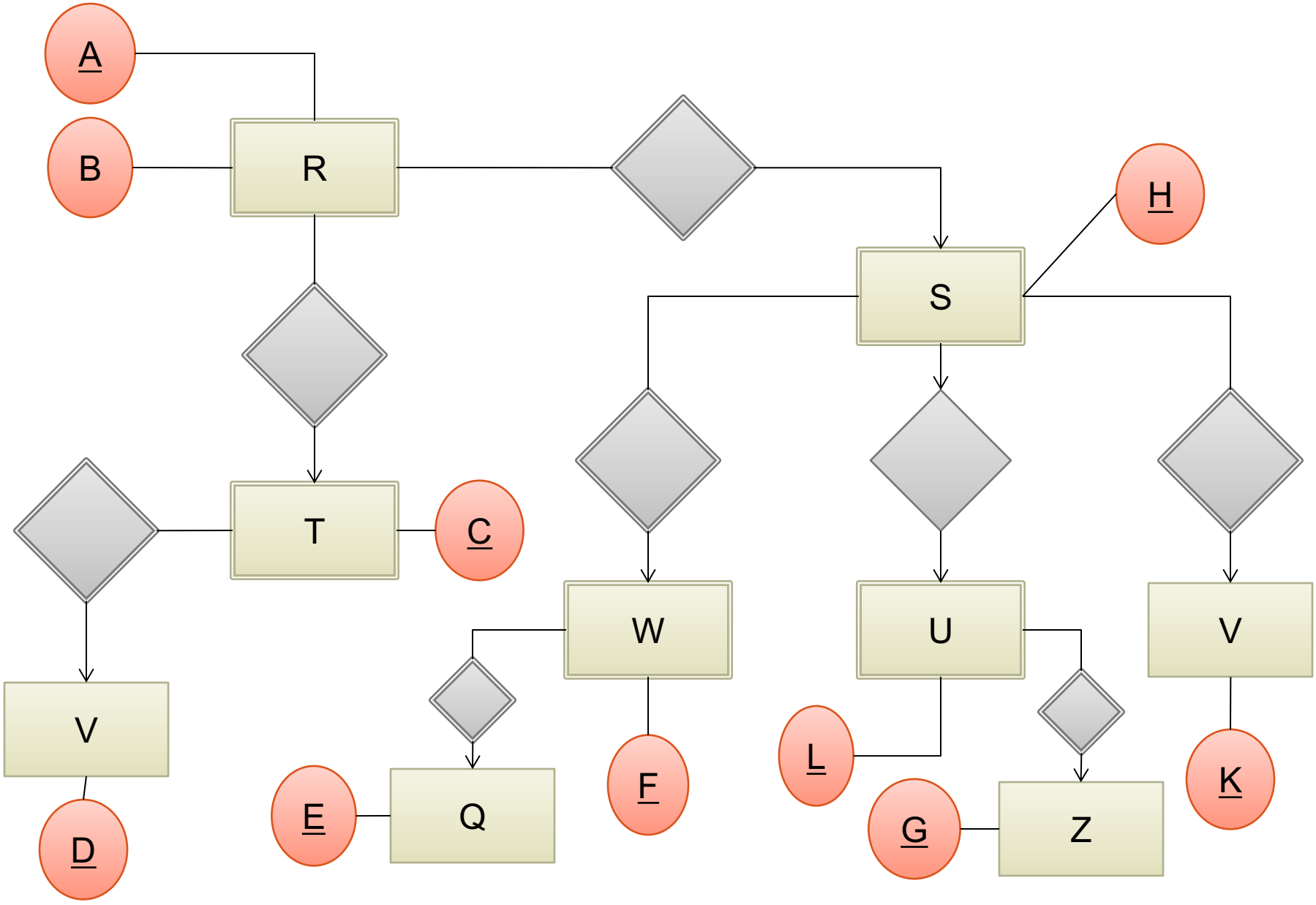
# WEAK ENTITY SETS

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 ?



# INTEGRITY CONSTRAINTS MOTIVATION

An integrity constraint is a condition specified on a database schema that restricts the data that can be stored in an instance of the database.

**ICs help prevent entry of incorrect information**

**How? DBMS enforces integrity constraints**

- Allows only legal database instances (i.e., those that satisfy all constraints) to exist
- Ensures that all necessary checks are always performed and avoids duplicating the verification logic in each application

# CONSTRAINTS IN E/R DIAGRAMS

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

Keys: social security number uniquely identifies a person.

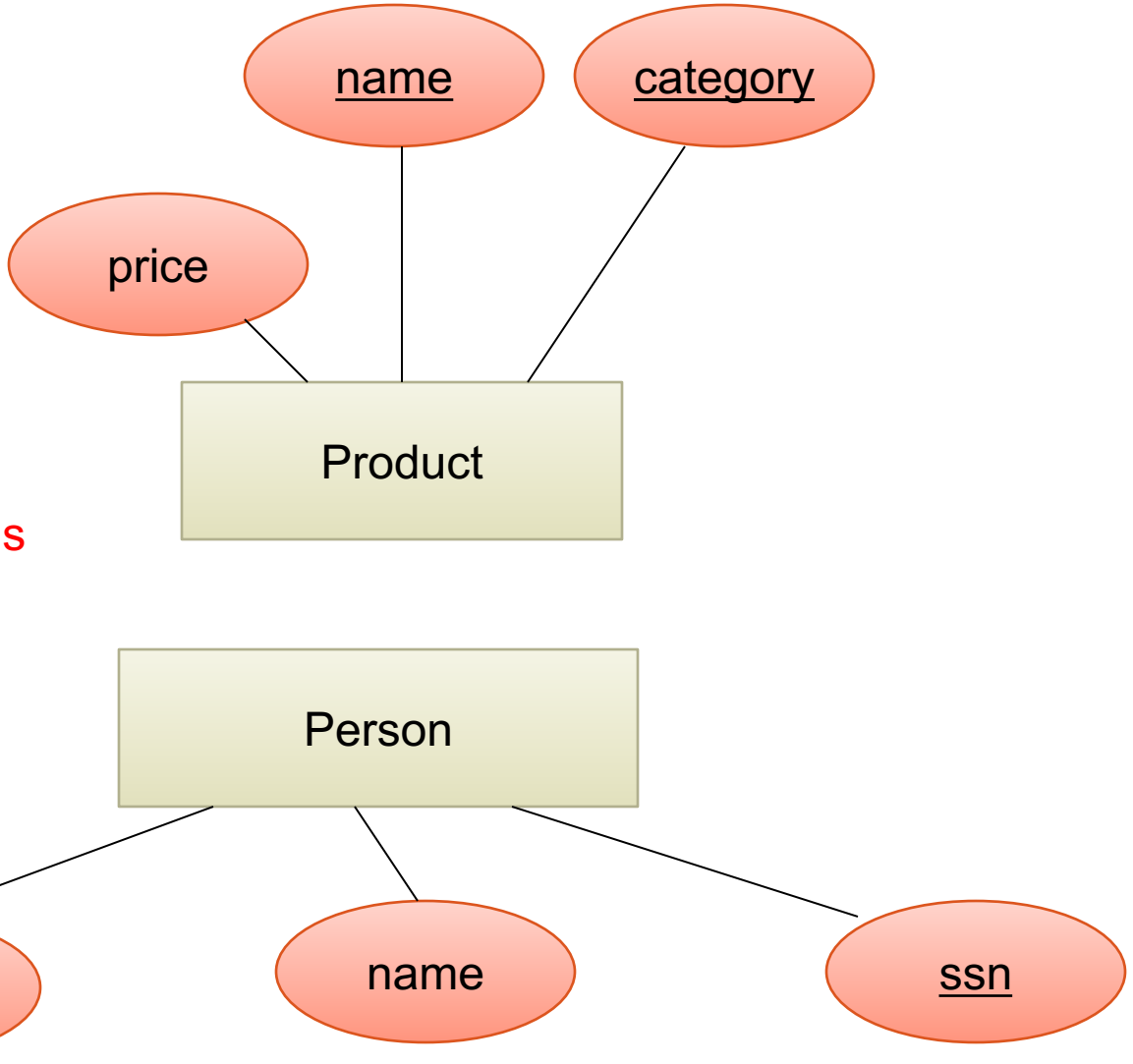
Single-value constraints: a person can have only one father.

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:

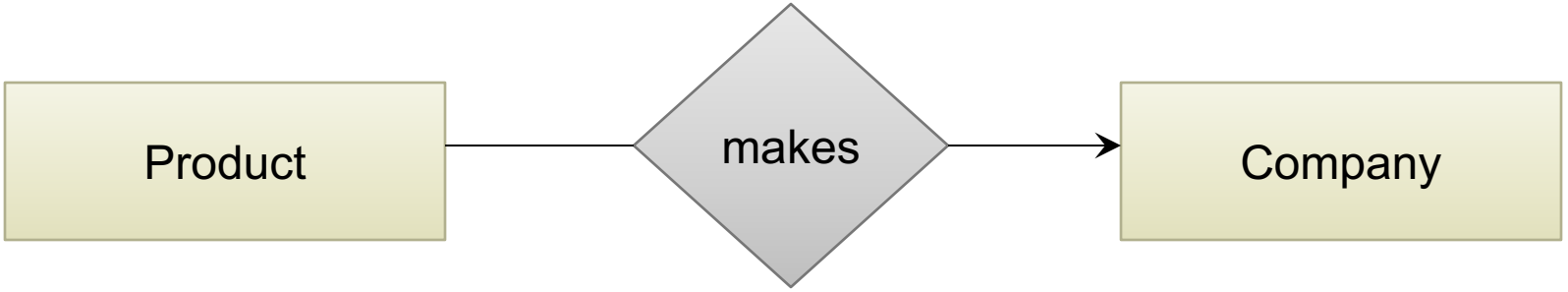



No formal way
to specify multiple
keys in E/R diagrams

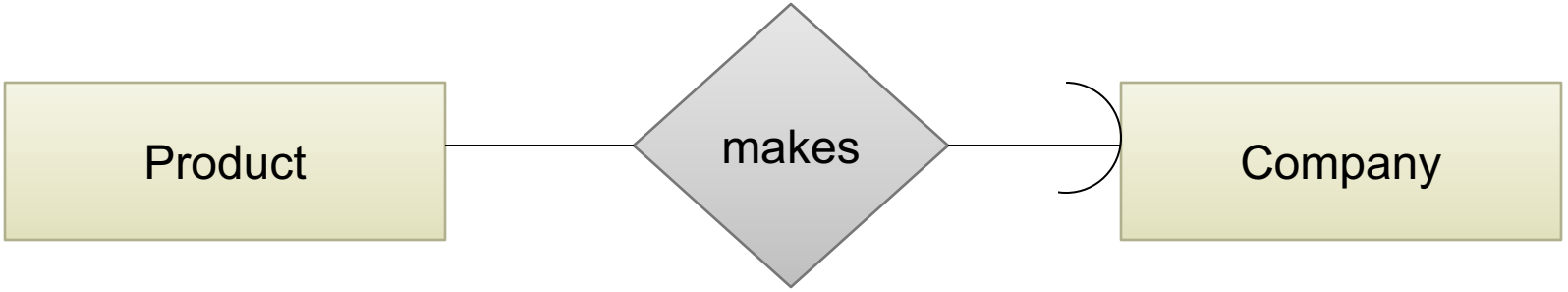
# SINGLE VALUE CONSTRAINTS


vs.

# REFERENTIAL INTEGRITY CONSTRAINTS

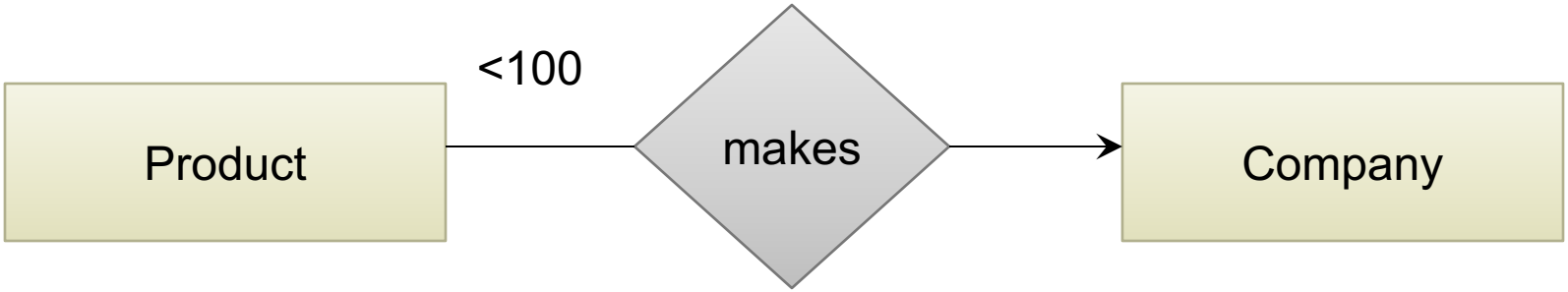



Each product made by at most one company.
Some products made by no company




Each product made by *exactly* one company.

# OTHER CONSTRAINTS




Q: What does this mean ?
A: A Company entity cannot be connected
by relationship to more than 99 Product entities

# CONSTRAINTS IN SQL

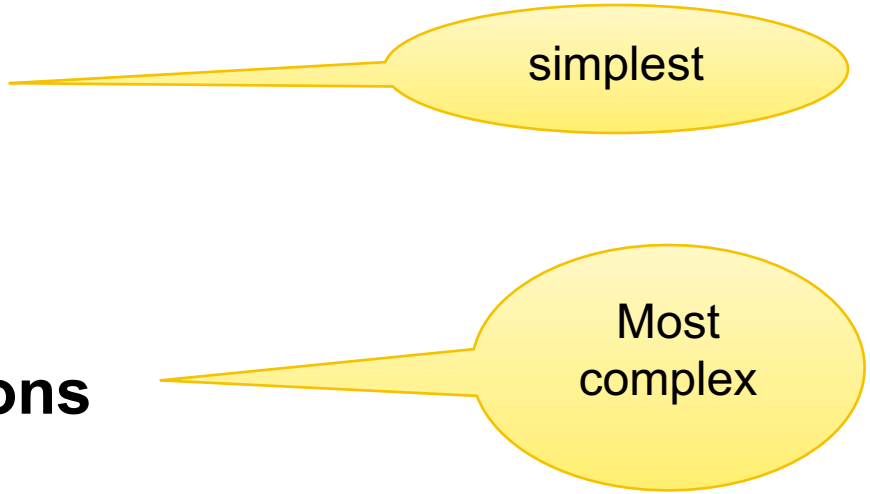
**Constraints in SQL:**

**Keys, foreign keys**

**Attribute-level constraints**

**Tuple-level constraints**

**Global constraints: assertions**




**The more complex the constraint, the harder it is to check and to enforce**

# KEY CONSTRAINTS

Product(name, category)

```
CREATE TABLE Product (
      name CHAR(30) PRIMARY KEY,
      category VARCHAR(20))
```

**OR:**

```
CREATE TABLE Product (
      name CHAR(30),
      category VARCHAR(20),
PRIMARY KEY (name))
```

# KEYS WITH MULTIPLE ATTRIBUTES

Product(name, category, price)

```
CREATE TABLE Product (
        name CHAR(30),
        category VARCHAR(20),
        price INT,
    PRIMARY KEY (name, category))
```

| Name | Category | Price |
|---|---|---|
| Gizmo | Gadget | 10 |
| Camera | Photo | 20 |
| Gizmo | Photo | 30 |
| ~~Gizmo~~ | ~~Gadget~~ | ~~40~~ |

# OTHER KEYS

```
CREATE TABLE Product (
        productID  CHAR(10),
        name CHAR(30),
        category VARCHAR(20),
        price INT,
        PRIMARY KEY (productID),
        UNIQUE (name, category))
```

There is at most one PRIMARY KEY;
there can be many UNIQUE

# FOREIGN KEY CONSTRAINTS

```
CREATE TABLE Purchase (
        prodName CHAR(30)
        REFERENCES Product(name),
        date DATETIME)
```

Referential integrity constraints

prodName is a **foreign key** to Product(name)
name must be a **key** in Product

May write just Product if name is PK

# FOREIGN KEY CONSTRAINTS

**Example with multi-attribute primary key**

```
CREATE TABLE Purchase (
      prodName CHAR(30),
      category VARCHAR(20),
      date DATETIME,
      FOREIGN KEY (prodName, category)
            REFERENCES  Product(name, category)
```

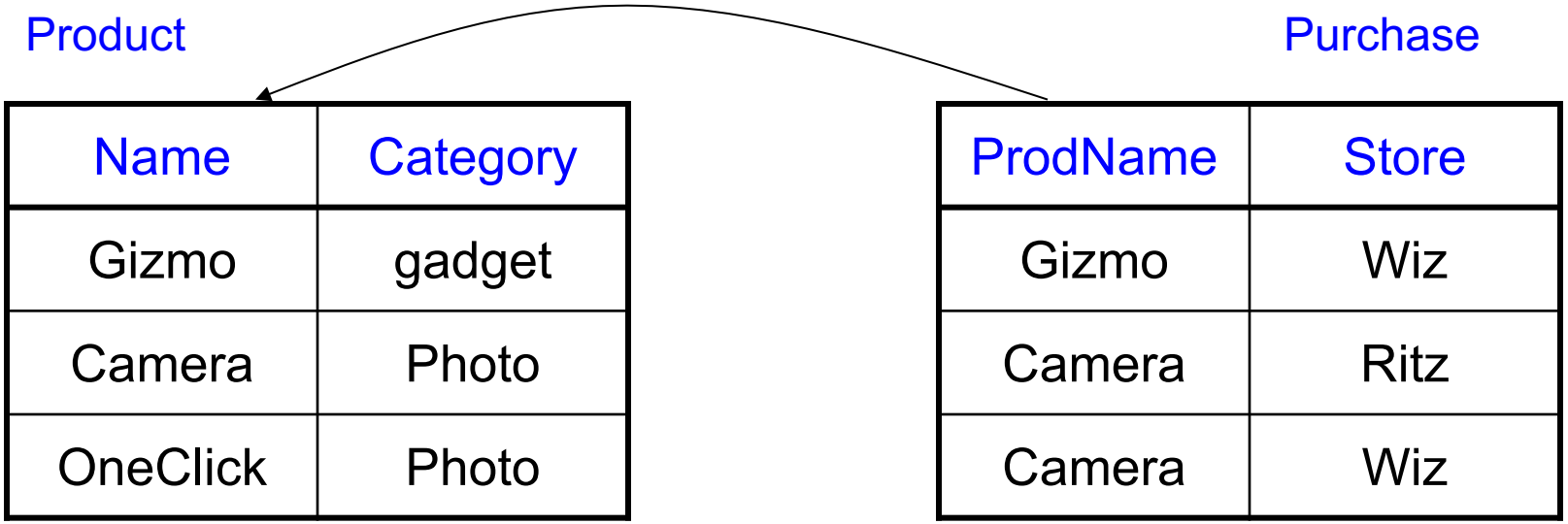
**(name, category) must be a KEY in Product**

# WHAT HAPPENS WHEN DATA CHANGES?

**Types of updates:**

**In Purchase: insert/update**

**In Product: delete/update**



# WHAT HAPPENS WHEN DATA CHANGES?

**SQL has three policies for maintaining referential integrity:**

**NO ACTION reject violating modifications (default)**

**CASCADE after delete/update do delete/update**

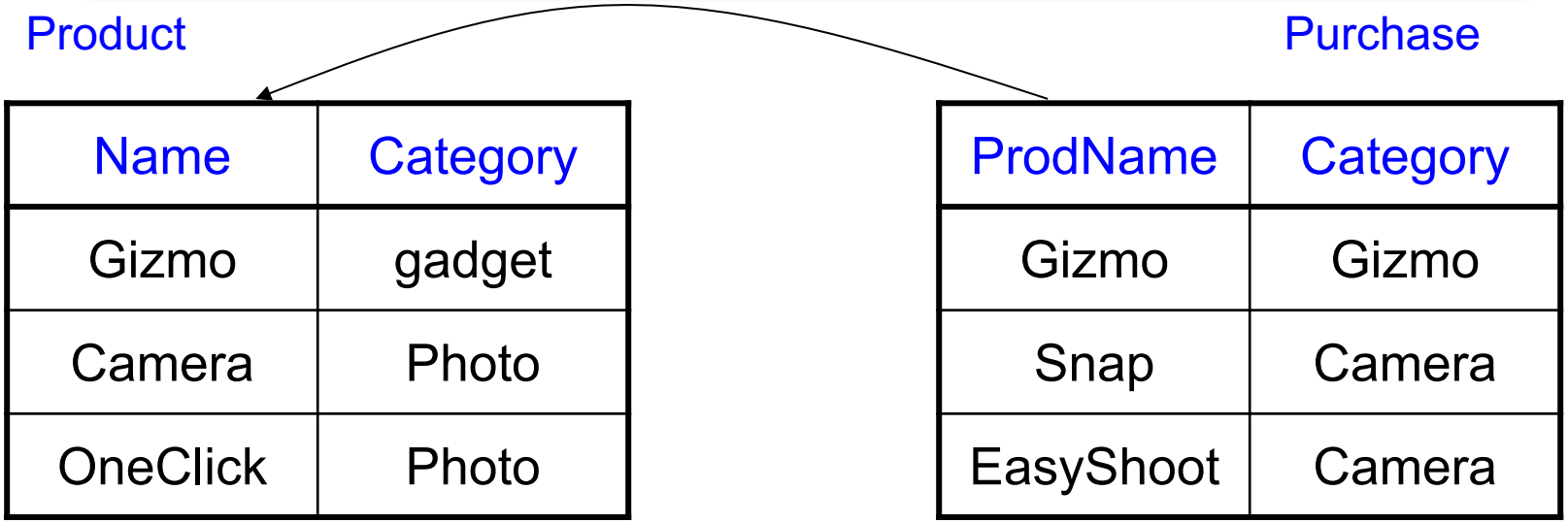
**SET NULL set foreign-key field to NULL**

**SET DEFAULT set foreign-key field to default value**

- need to be declared with column, e.g.,
  `CREATE TABLE Product (pid INT DEFAULT 42)`

# MAINTAINING REFERENTIAL INTEGRITY

```
CREATE TABLE Purchase (
        prodName CHAR(30),
        category VARCHAR(20),
        date DATETIME,
        FOREIGN KEY (prodName, category)
                REFERENCES  Product(name, category)
          ON UPDATE CASCADE
          ON DELETE SET NULL    )
```




| Name | Category |
|---|---|
| Gizmo | gadget |
| Camera | Photo |
| OneClick | Photo |

| ProdName | Category |
|---|---|
| Gizmo | Gizmo |
| Snap | Camera |
| EasyShoot | Camera |

# CONSTRAINTS ON ATTRIBUTES AND TUPLES

**Constraints on attributes:**

**NOT NULL** -- obvious meaning...

**CHECK condition** -- any condition !

**Constraints on tuples**

**CHECK condition**

# CONSTRAINTS ON ATTRIBUTES AND TUPLES

```
CREATE TABLE R (
        A int NOT NULL,
        B int CHECK (B > 50 and B < 100),
        C varchar(20),
    D int,
        CHECK (C >= 'd' or D > 0))
```

# CONSTRAINTS ON ATTRIBUTES AND TUPLES

```
CREATE TABLE Product (
        productID  CHAR(10),
        name CHAR(30),
        category VARCHAR(20),
        price INT CHECK (price > 0),
        PRIMARY KEY (productID),
        UNIQUE (name, category))
```

# Constraints on Attributes and Tuples

What does this constraint do?

```
CREATE TABLE Purchase (
      prodName CHAR(30)
            CHECK (prodName IN
                        (SELECT Product.name
                         FROM Product),
      date DATETIME NOT NULL)
```

What
is the difference from
Foreign-Key ?

# GENERAL ASSERTIONS

```
CREATE ASSERTION myAssert CHECK
  (NOT EXISTS(
       SELECT Product.name
       FROM Product, Purchase
       WHERE Product.name = Purchase.prodName
       GROUP BY Product.name
       HAVING count(*) > 200) )
```

But most DBMSs do not implement assertions
Because it is hard to support them efficiently
Instead, they provide triggers