Database Systems
CSE 414

Lecture 16: Design Theory
(Ch. 3.1, 3.3-4)

Announcements

- HW5 - Was on NoSQL (not doing)
- HW6 - Out tonight

- Midterm Will Use Gradescope
  - Will be out by tonight.
  - Check you UW Email address for Gradescope link
  - Have until Friday to file re-grades

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 years.
- Updating the schema in production is very hard:
  - schema change modifications are expensive (why?)
  - making the change without introducing any bugs is hard
    - this part is, by far, the most important consideration in practice

Database Design Process

Conceptual Model:

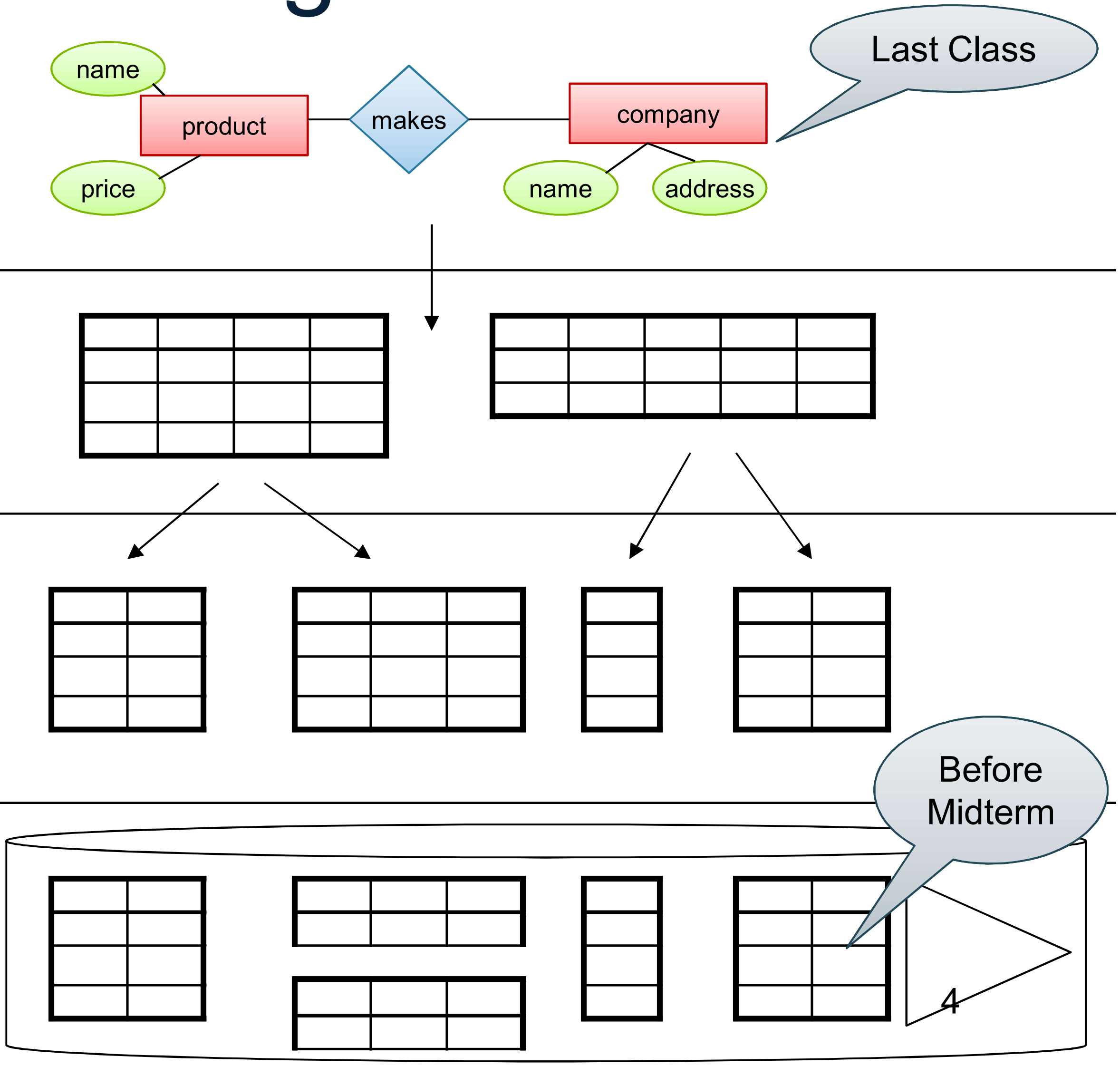
Relational Model:
Tables + constraints
And also functional dep.

Normalization:
Eliminates anomalies

Conceptual Schema

Physical storage details

Physical Schema

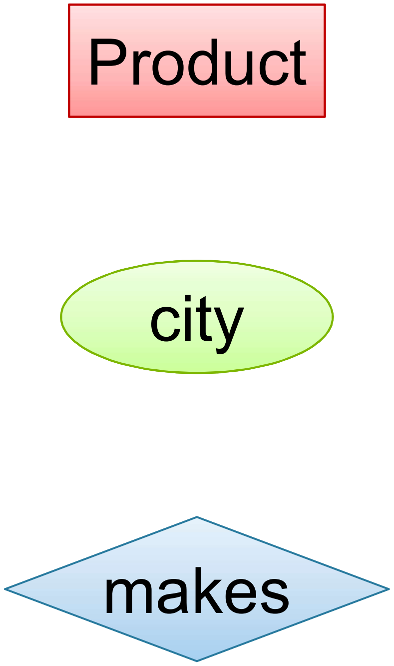


Last Class

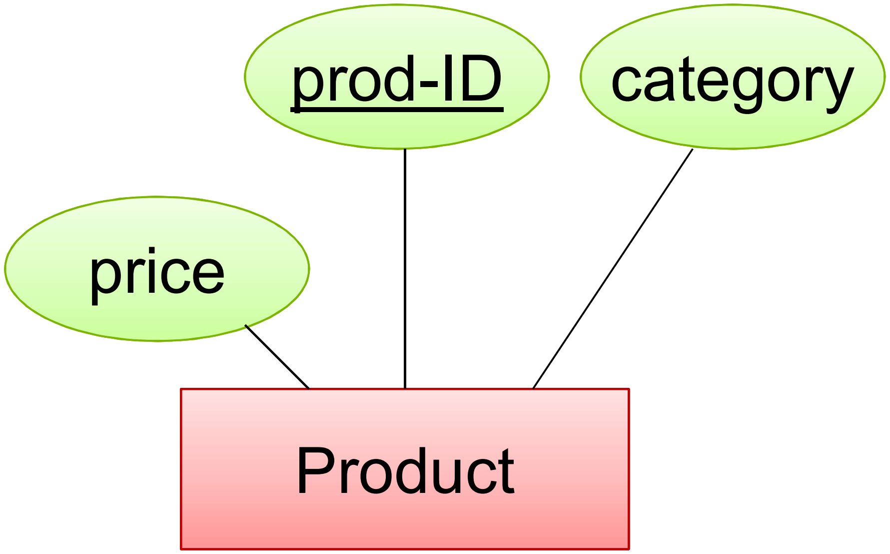
Before Midterm

Entity / Relationship Diagrams

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



Entity Set to Relation

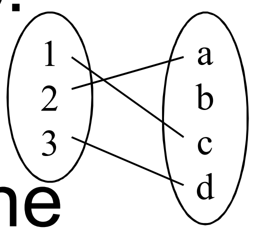


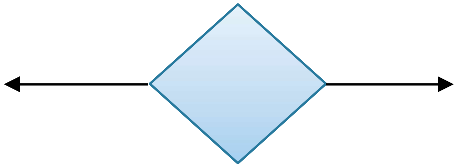
Product(<u>prod-ID</u>, category, price)

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

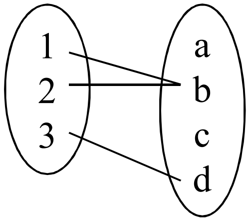
Multiplicity of E/R Relations

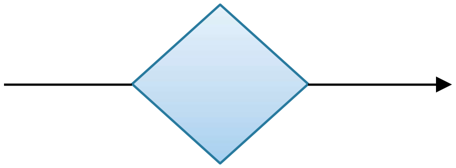
- one-one:



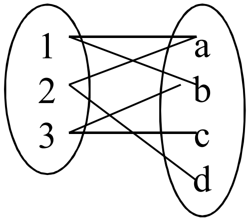


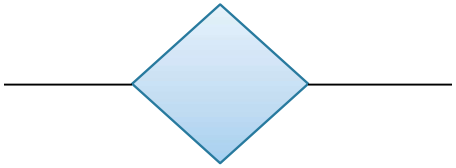
- many-one



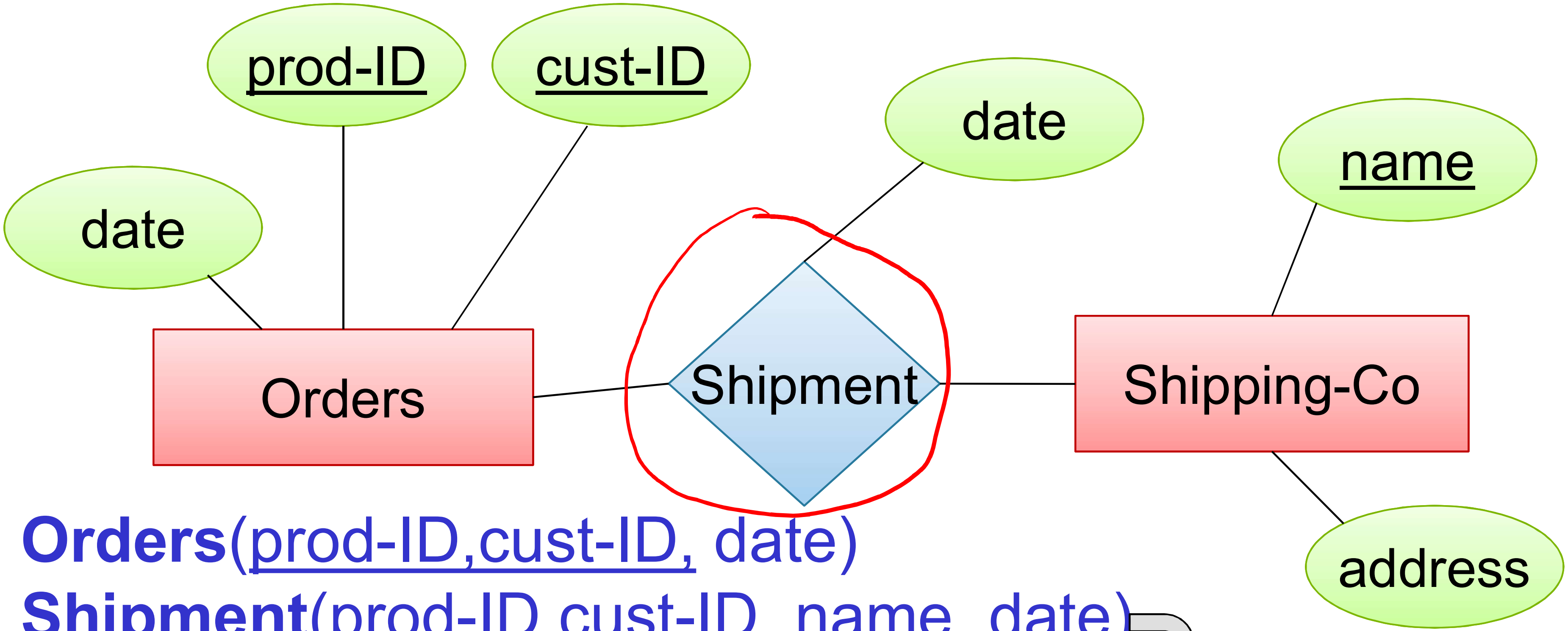


- many-many





N-N Relationships to Relations



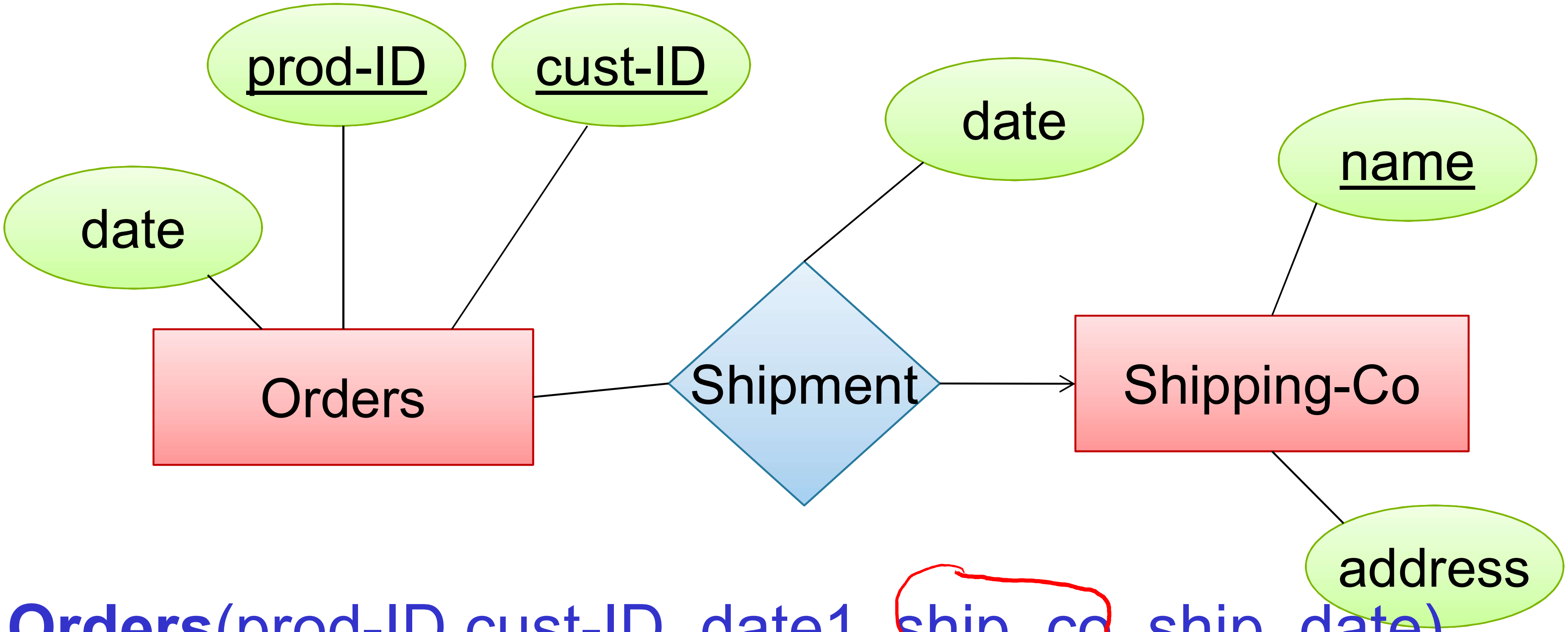
Orders(prod-ID,cust-ID, date)

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

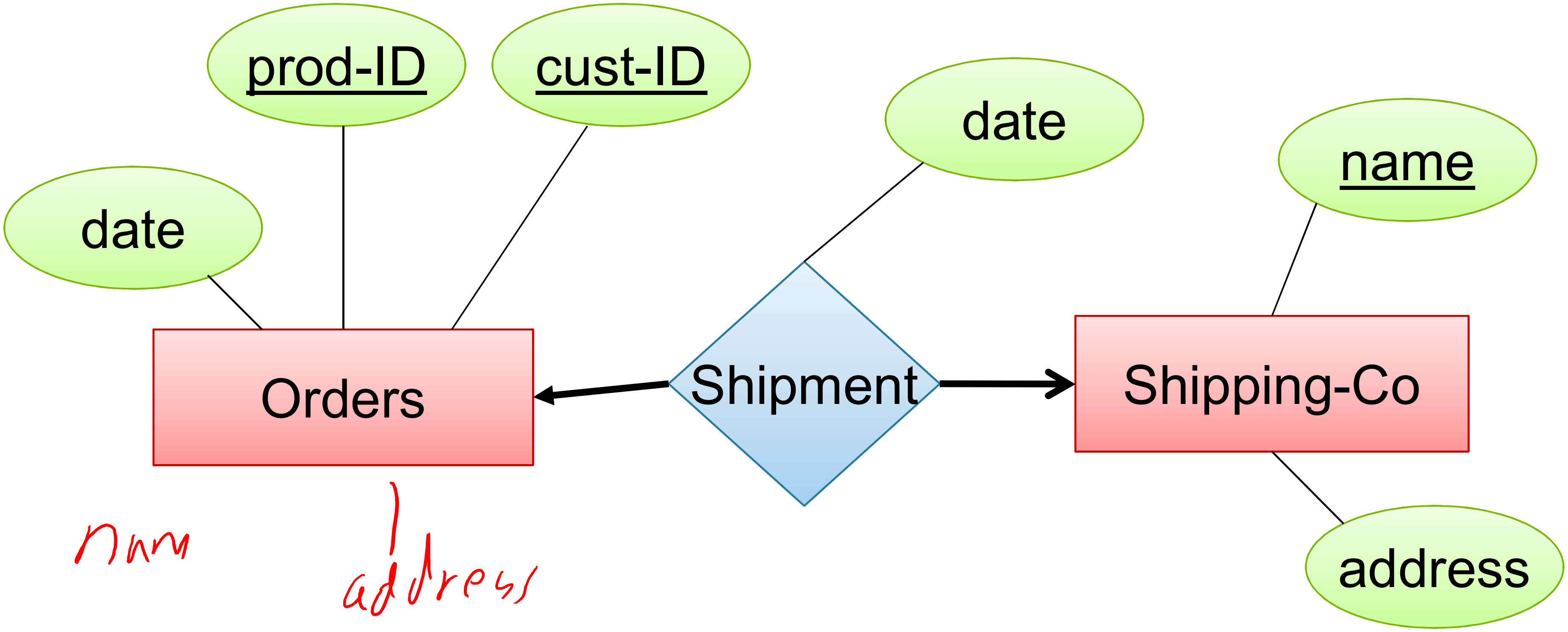
N-1 Relationships to Relations



Orders(prod-ID,cust-ID, date1, ship_co, ship_date)
Shipping-Co(name, address)

Note: many-one relationship becomes FK not relation

What about 1 - 1 relationship

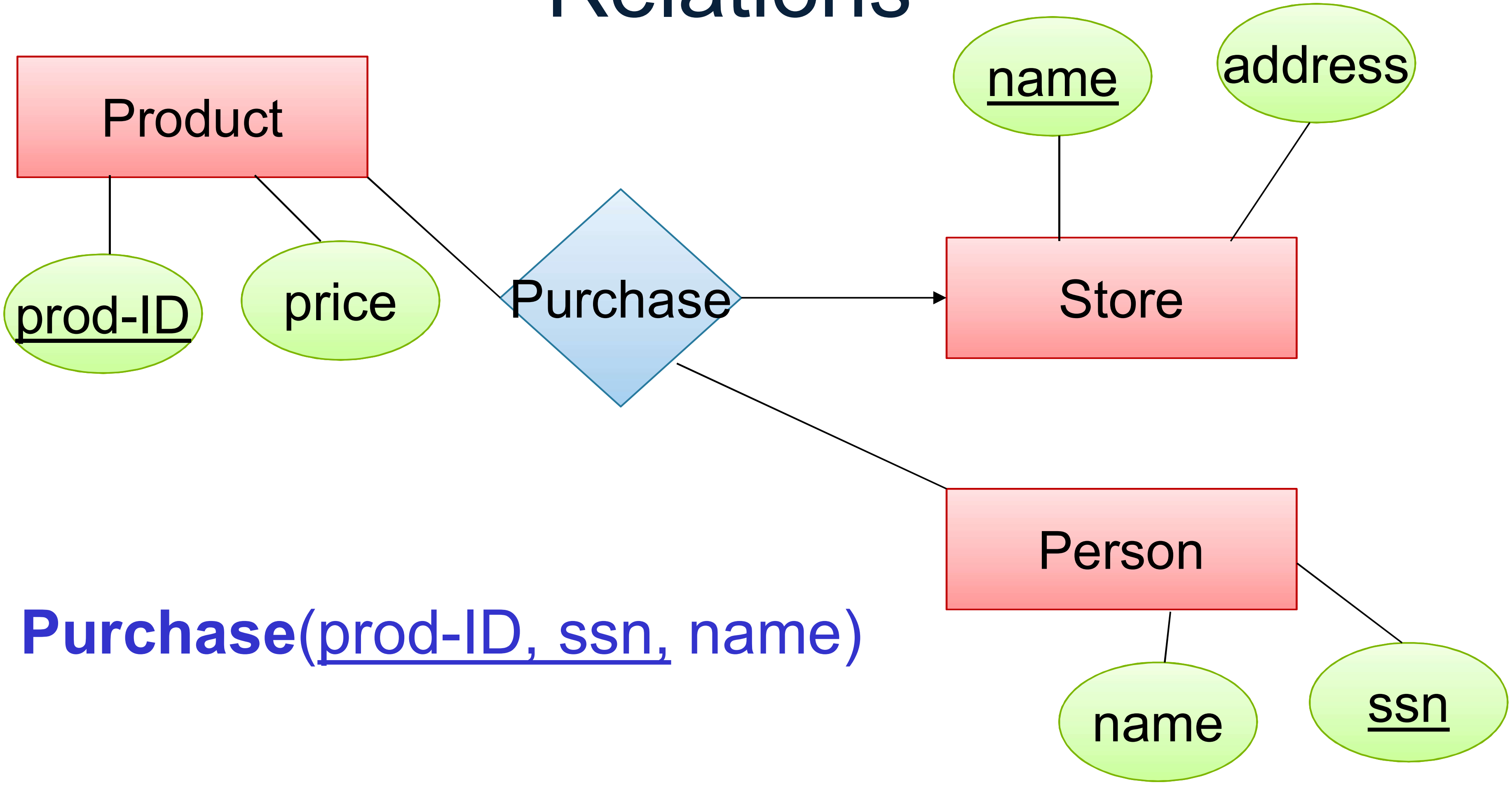


Orders(prod-ID,cust-ID, date1, ship_co, ship_date)
Shipping-Co(name, address)

~~**Note:** one-one relationship make FK part of child PK~~

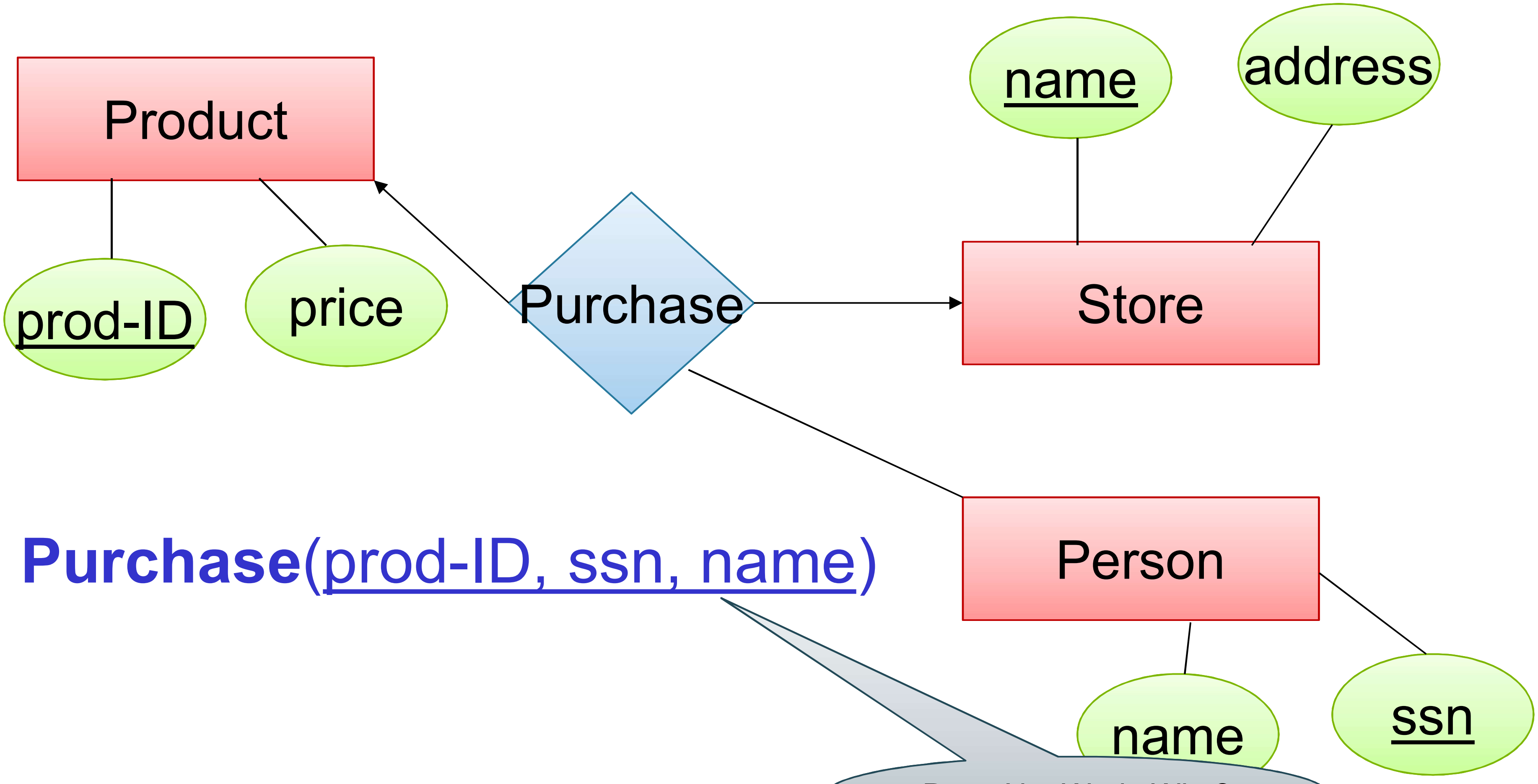
`one-one relationship need to have a UNIQUE constraint for each key.`

Multi-way Relationships to Relations



Purchase(prod-ID, ssn, name)

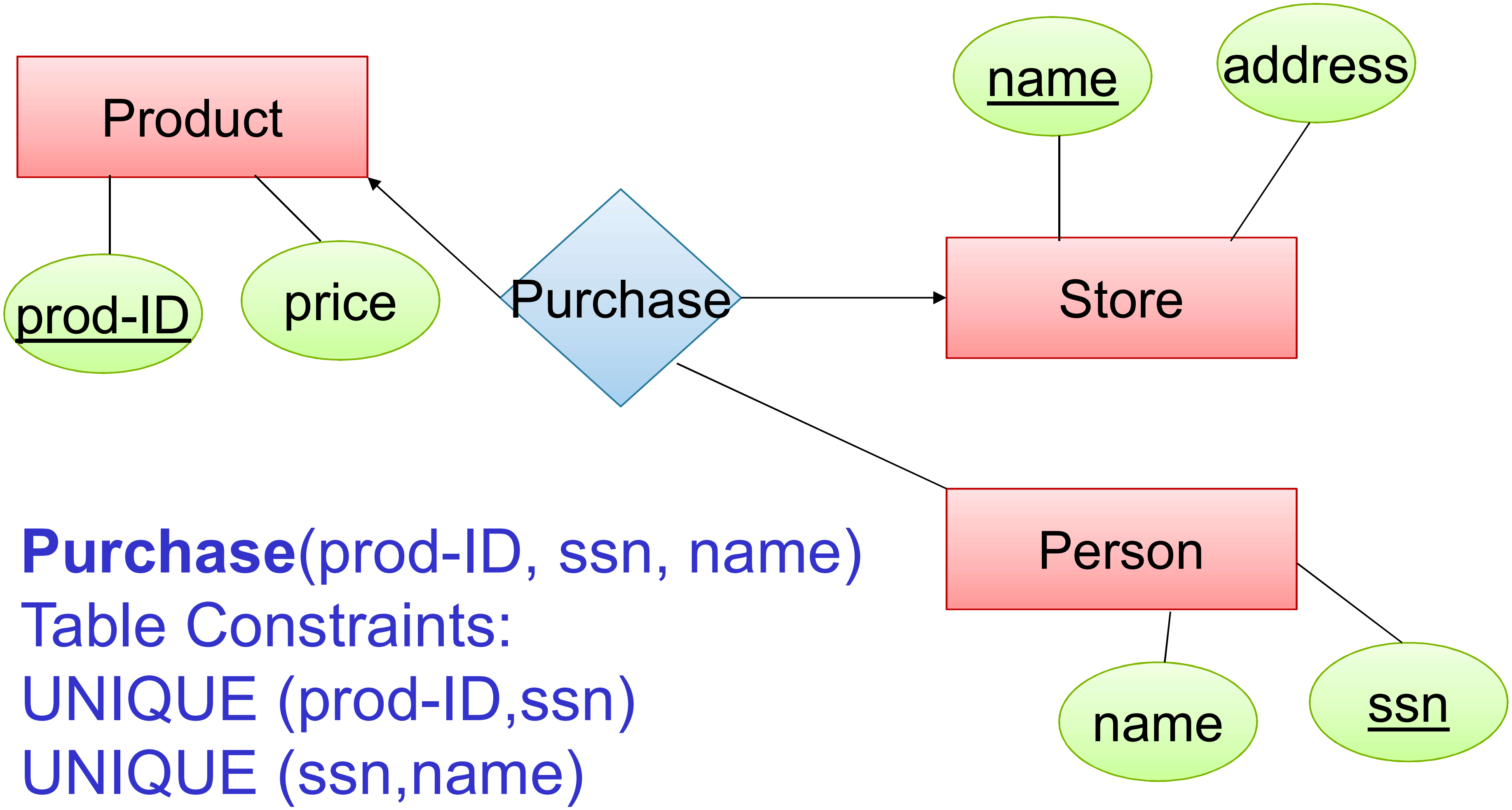
What about now?



Purchase(prod-ID, ssn, name)

Does Not Work. Why?

What about now?

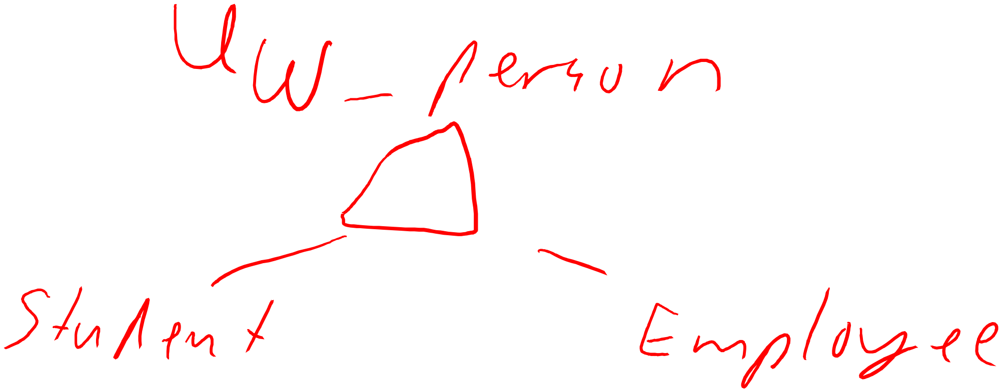


Purchase(prod-ID, ssn, name)

Table Constraints:

UNIQUE (prod-ID,ssn)

UNIQUE (ssn,name)



What makes good schemas?

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.

Most important issue in practice

- 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: can have only one genetic 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.
some values should not be NULL

Constraints in SQL

Constraints in SQL:

- Keys, foreign keys
- Attribute-level constraints
- Tuple-level constraints
- Global constraints: assertions

simplest

Most complex

- The more complex the constraint, the harder it is to check and to enforce…
  - (Still, performance is secondary to correctness.)

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

Key Constraints

Attribute Constraint

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

Tuple / Table Constraint

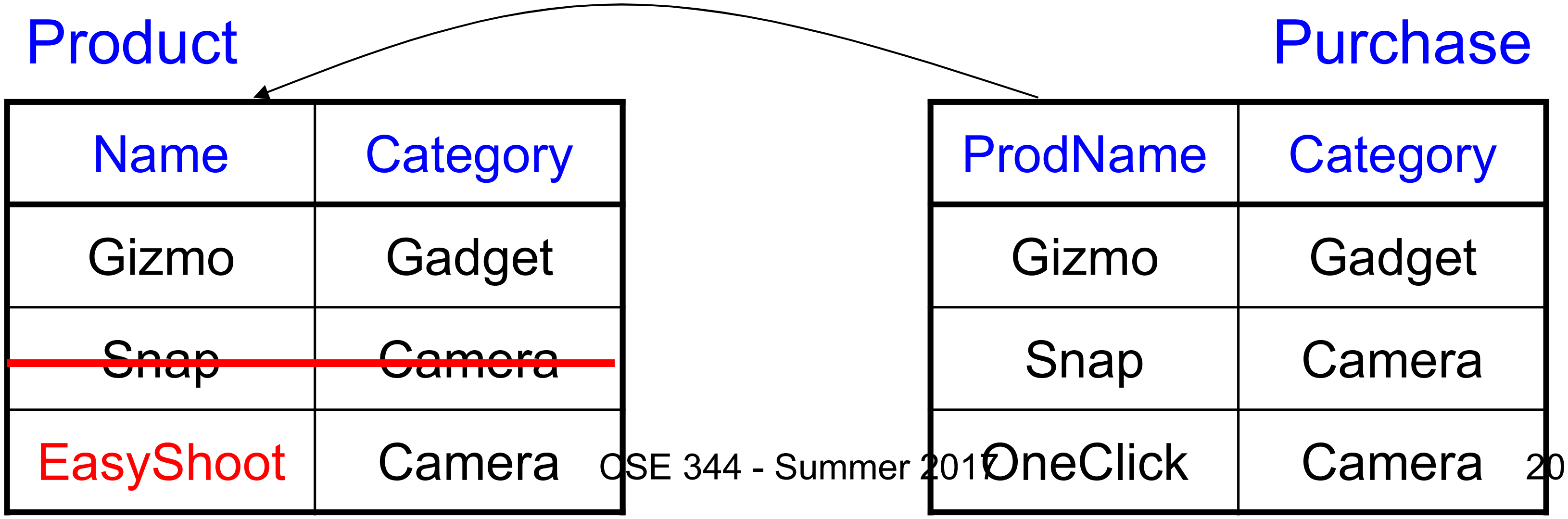
Second form need for multiple keys

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

Same for PRIMARY KEY and UNIQUE

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    )
```



Product

Name	Category
Gizmo	Gadget
~~Snap~~	~~Camera~~
EasyShoot	Camera

Purchase

ProdName	Category
Gizmo	Gadget
Snap	Camera
OneClick	Camera

Constraints on Attributes and Tuples

- Constraints on attributes:
  - NOT NULL --
  - CHECK condition --
- Constraints on tuples
  - CHECK condition

Checked when attribute changes

Checked when tuple/row changes

Constraints on Attributes and Tuples

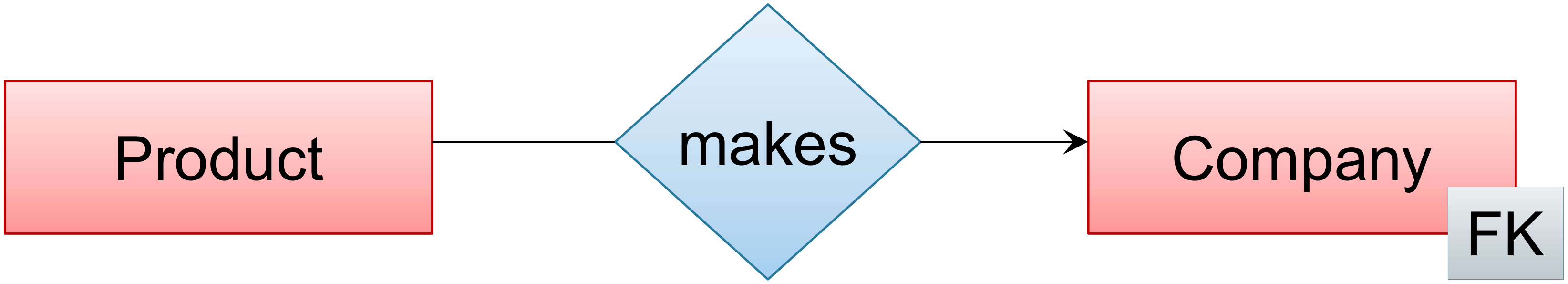
```
CREATE TABLE Product (
          productID  CHAR(10),
          name CHAR(30) NOT NULL,
          category VARCHAR(20)
             CHECK (category in (‘toy’,’gadget’,’apparel’)),
          price INT CHECK (price > 0),
          PRIMARY KEY (productID),
           CHECK  price < 10 and category = ‘toy’
)
```

Price>=10 or ~~(price<0~~ and cat = ''toy''

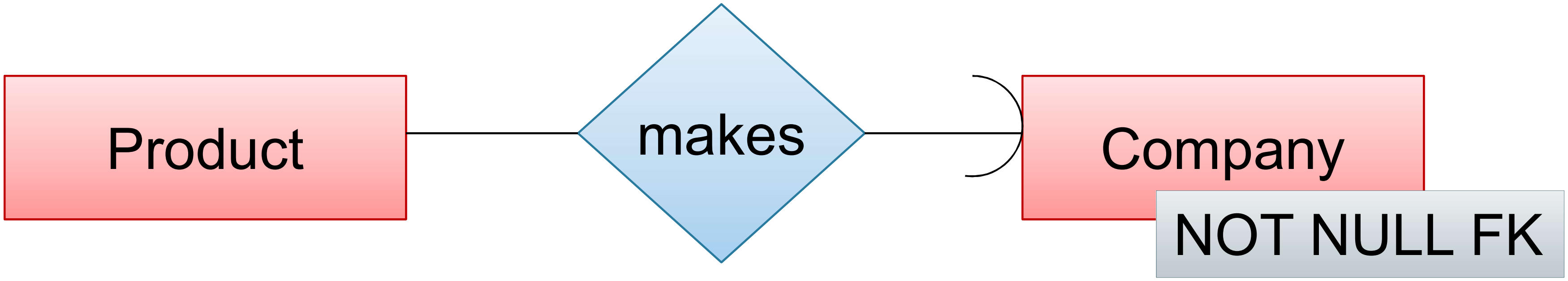
Attribute Constraint

Table Constraint. Why?

Referential Integrity Constraints

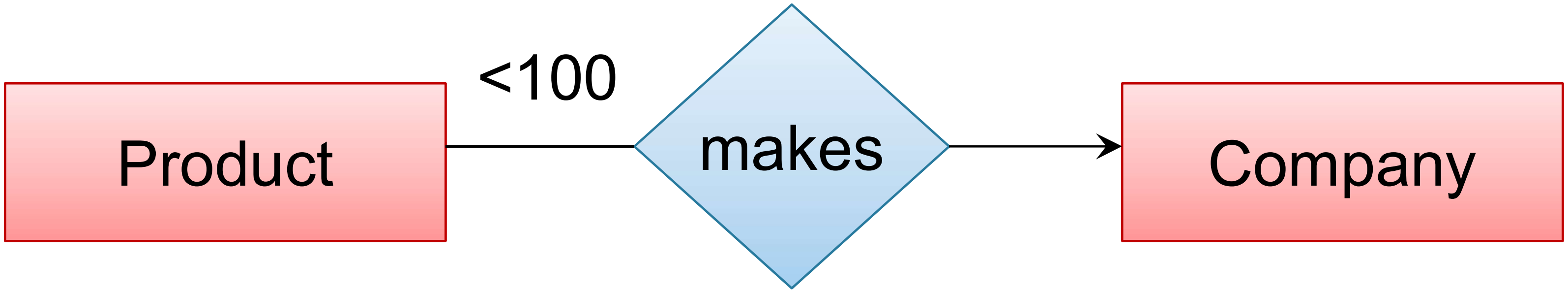


Each product made by at most one company.
Some products made by no company



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

Try at home: How would you implement this?

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)
```

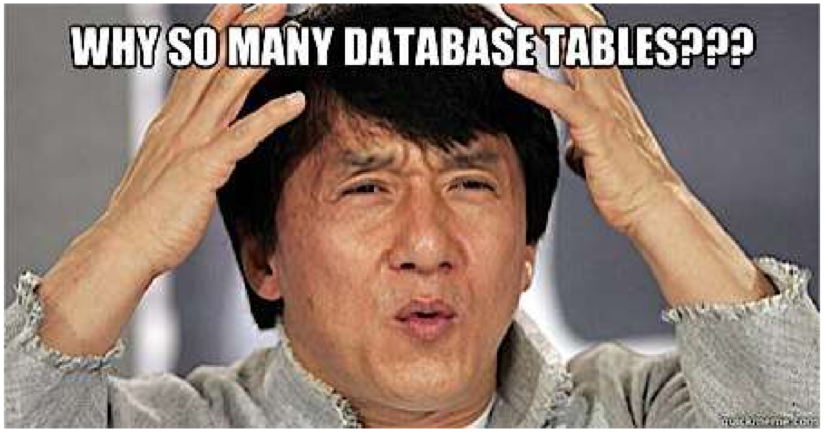
What is the difference from Foreign Key?

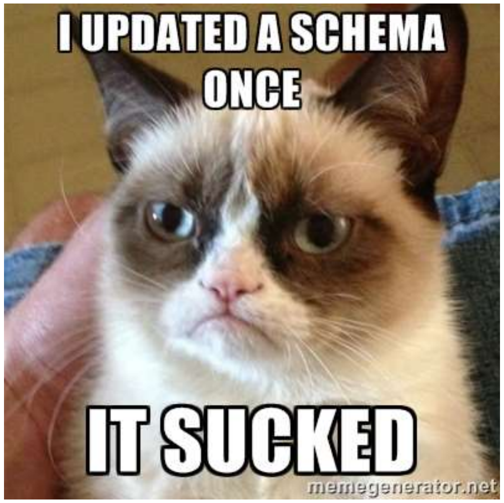
General Assertions

```
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

What makes good schemas?





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
What is the primary key?

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

These can cause bugs!

Worry most about later two.

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	Bellview ~~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 ***functional dependencies*** (FDs)
- Use FDs to ***normalize*** the relational schema

Functional Dependencies (FDs)

Definition

If two tuples agree on the attributes

$A_1, A_2, \ldots, A_n$

then they must also agree on the attributes

$B_1, B_2, \ldots, B_m$

Formally:

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

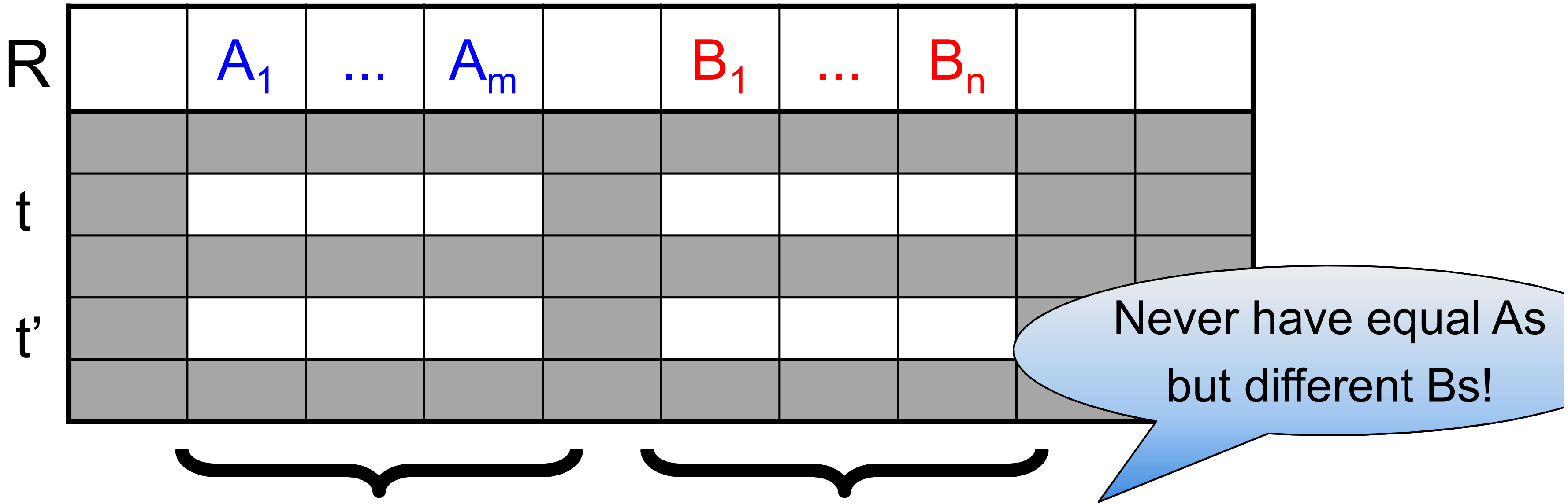
$A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$

Functional Dependencies (FDs)

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

for every pair of tuples t, t’ $\in$ R,

$(t.A_1 = t'.A_1 \text{ and } \ldots\ t.A_m = t'.A_m \rightarrow t.B_1 = t'.B_1 \text{ and } \ldots\ t.B_n = t'.B_n)$



Never have equal As but different Bs!

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	Green	Toys	49
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** (part of schema)

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	321-54-9876	908-321-1234	Westfield

These FD's all hold on given instance:

~~SSN -> PhoneNumbe~~

- Name, SSN -> City
- SSN -> Name, City
- PhoneNumber -> City
- SSN -> City
- ~~City -> Name~~

R ~~satisfies only one.~~
Need to reason about what the data means.

An Interesting Observation

If all these FDs are true:

name → color
category → department
color, category → price

name

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, …, B_n → C   is a FD and
             B_1, …, 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 }

name, color, category, 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, C, D, E }

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

Example

In class:

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

A, B → C
A, D → E
B → D
A, F → 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 → C
A, D → E
B → D
A, F → B

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

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

What is a 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 → C
A, D → B
B → 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 → Y, s.t. $Y \subseteq X^+$ and $X \cap Y = \emptyset$:

AB → CD, AD→BC, ABC → D, ABD → C, ACD → 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
  - 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 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 (super)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

A → B
B → C
C → A

or

AB→C
BC→A

or

A→BC
B→AC

$\{A\}^+ = \{A, B, C\}$
$\{B\}^+ = \{B, C, A\}$
$\{C\}^+ = \{C, A, B\}$

what are the keys here ?

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