Database Management Systems
CSEP 544

Lecture 8:
Conceptual Design

Announcements

- HW7 released
  - Pen and paper exercise on conceptual design
- RA6 released
- OH change this Friday from 5-6pm
  - Always check the class schedule page for up to date info

$T(a,b,c) = R(a,b) \bowtie S(b,c)$

σ select

Π project

Endgame for CSEP 544

- Last lecture next Tuesday!
  - Transactions
  - (Recovery)
  - Recap of class
- HW8 (last one) will be released next week
- Final during the weekend of 12/9-10
  - Designed to be 2-hour exam
  - You will have 5 hours to work on it
  - Sample examples posted on course webpage
- Course videos are posted on course webpage

Class overview

- Data models
  - Relational: SQL, RA, and Datalog
  - NoSQL: SQL++
- RDBMS internals
  - Query processing and optimization
  - Physical design
- Parallel query processing
  - Spark and Hadoop
- Conceptual design
  - E/R diagrams
  - Schema normalization
- Transactions
  - Locking and schedules
  - Writing DB applications

Data models

Query Processing

Using DBMS

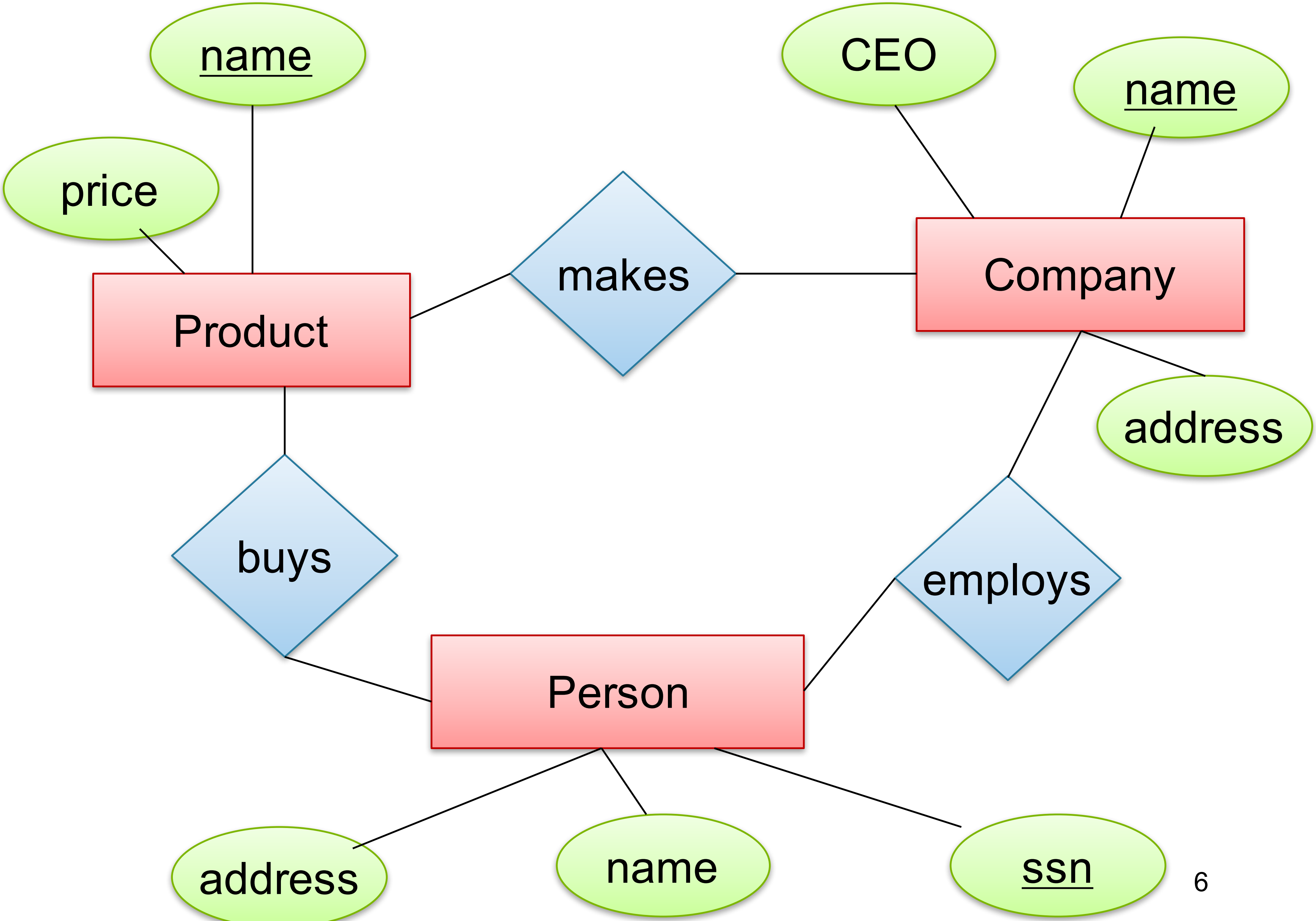
Database Design

What it is:

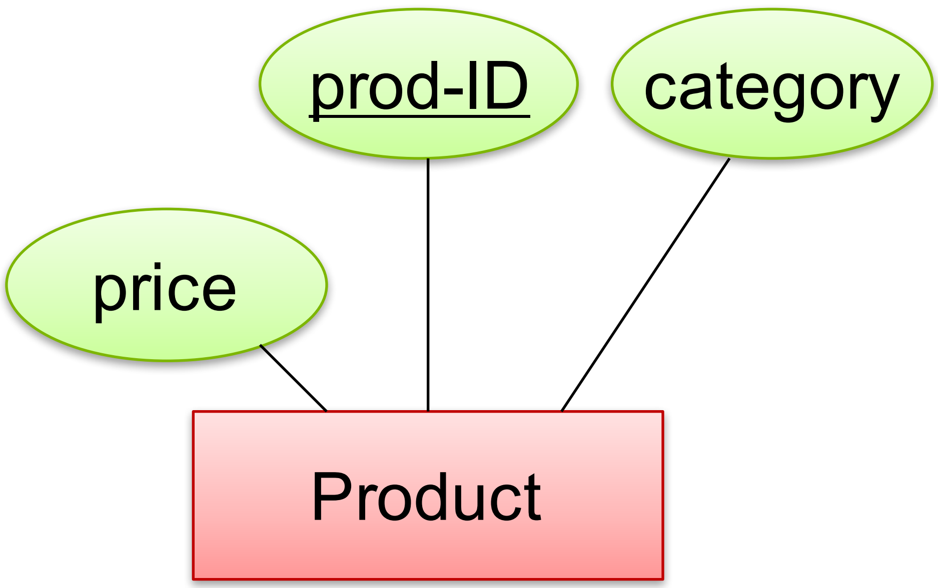
- Starting from scratch, design the database schema: relation, attributes, keys, foreign keys, constraints etc

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?)


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

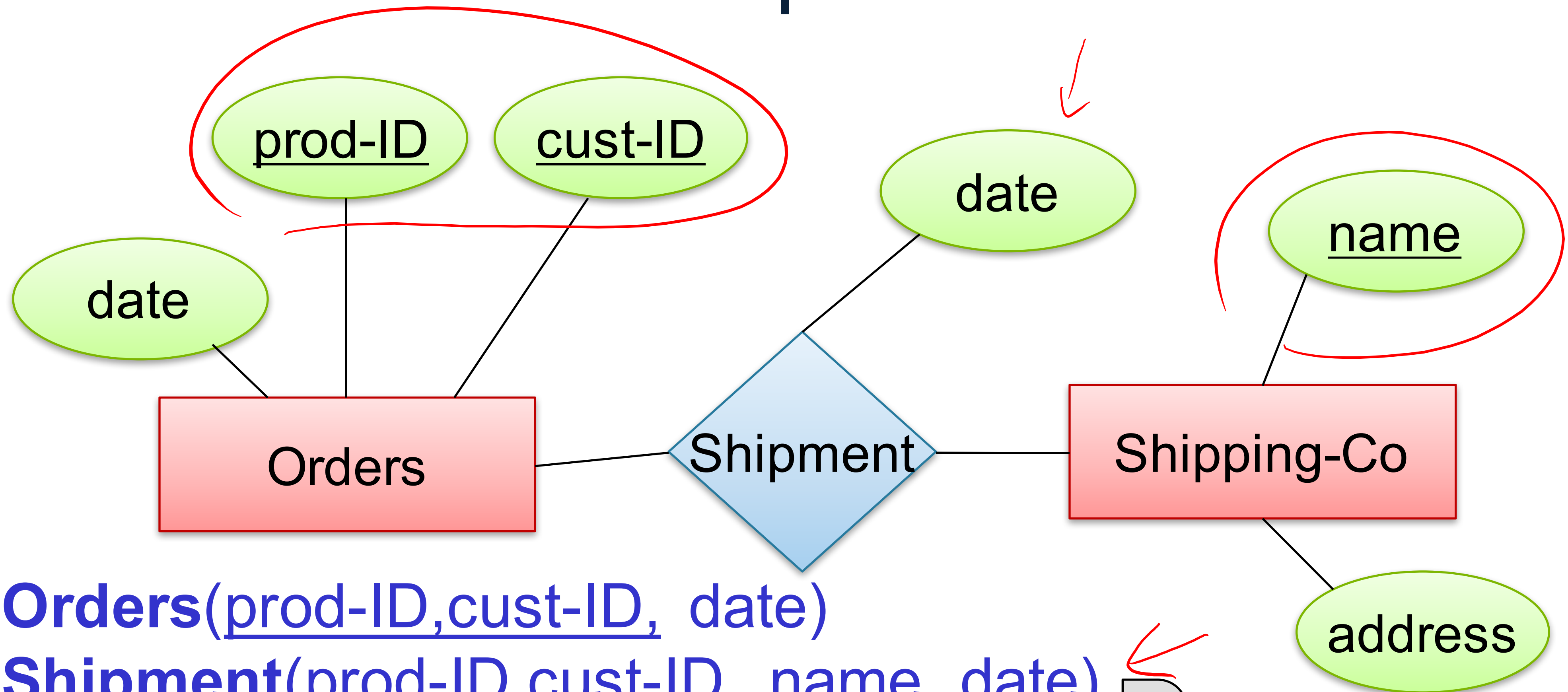
Entity Set to Relation



Product(prod-ID, category, price)

prod-ID	category	price
Gizmo55	Camera	99.99
Pokemn19	Toy	29.99

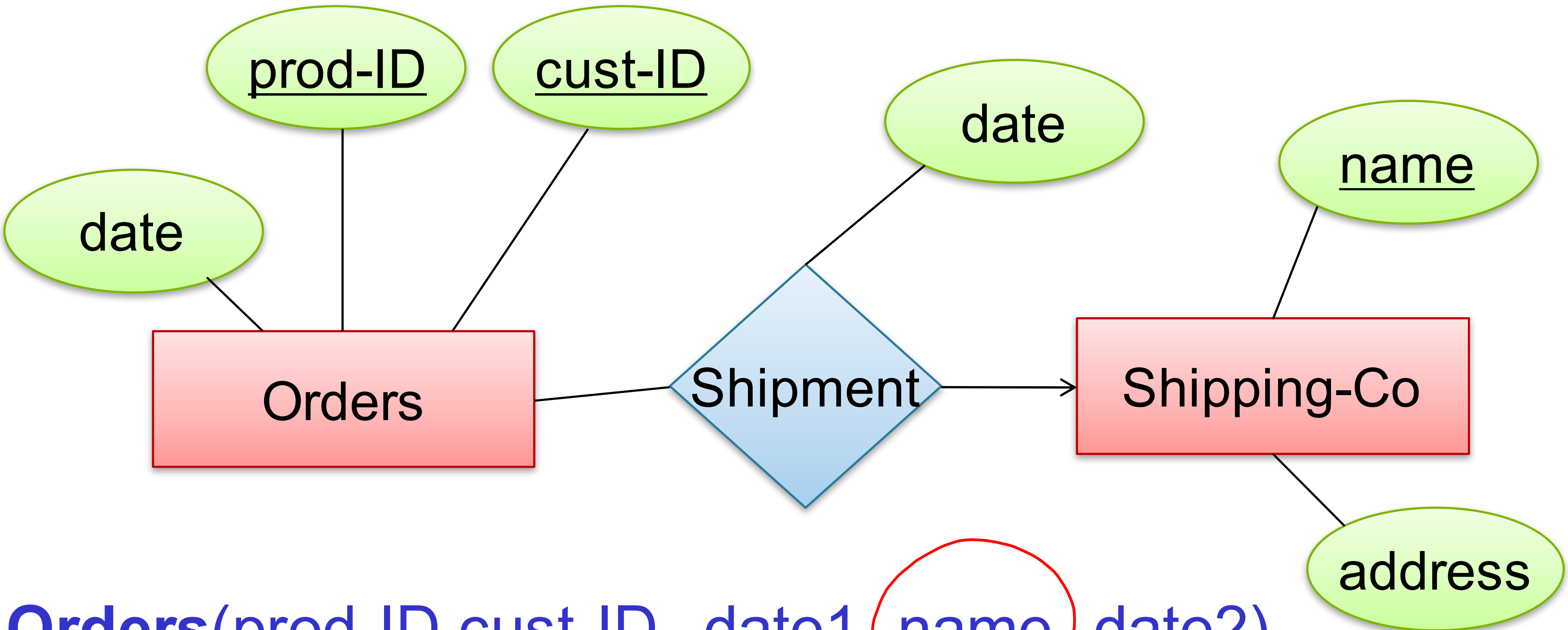
N-N Relationships to Relations



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

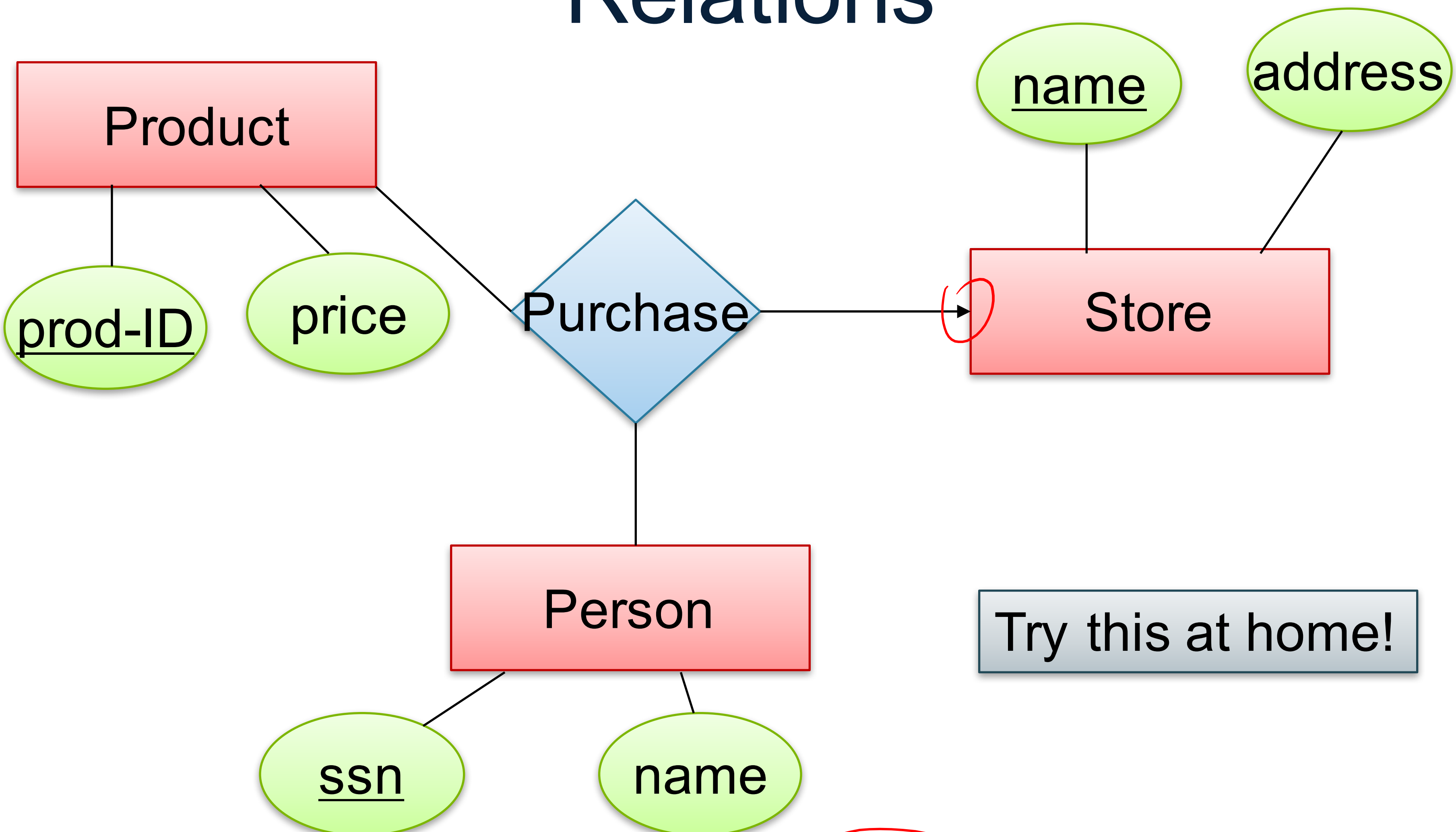
N-1 Relationships to Relations



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

Remember: no separate relations for many-one relationship

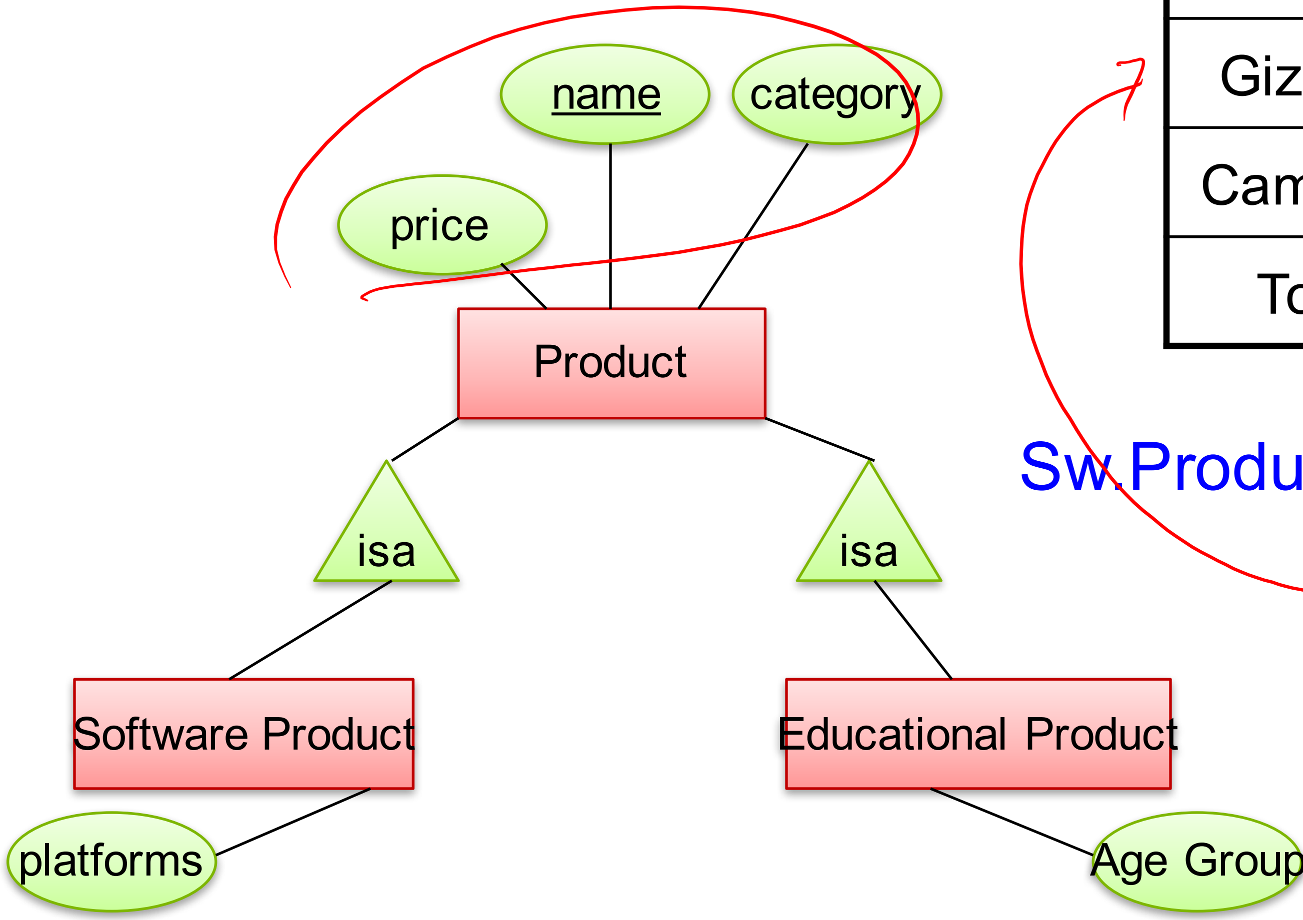
Multi-way Relationships to Relations



Try this at home!

Purchase(prod-ID, ssn, name)

Subclasses to Relations



Product

<u>Name</u>	Price	Category
Gizmo	99	gadget
Camera	49	photo
Toy	39	gadget

Sw.Product

<u>Name</u>	platforms
Gizmo	unix

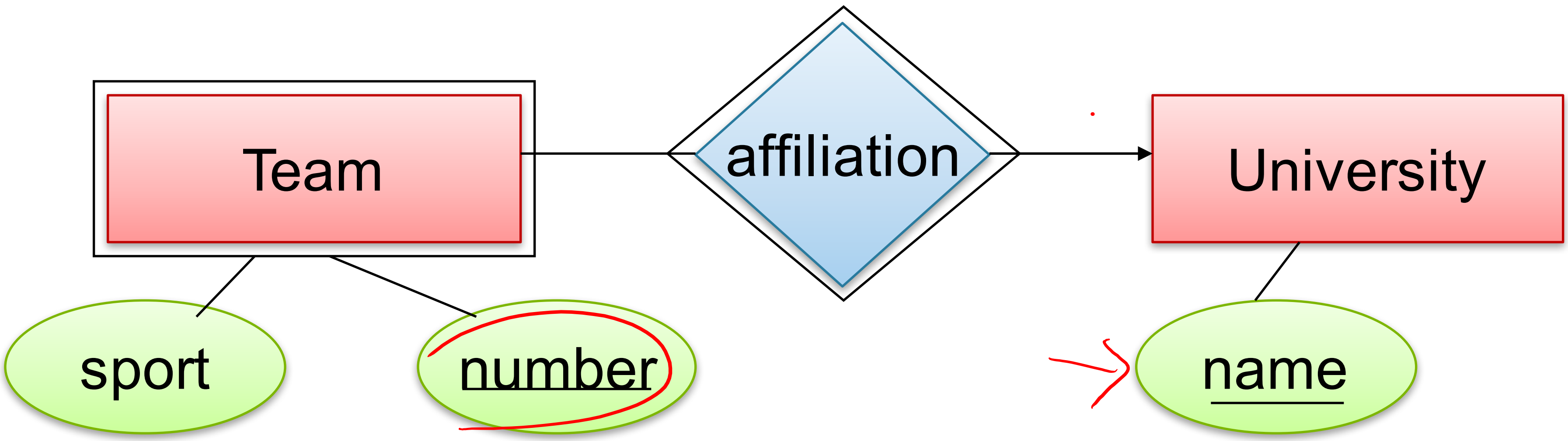
Ed.Product

<u>Name</u>	Age Group
Gizmo	toddler
Toy	retired

Other ways to convert are possible

Weak Entity Sets

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



Team(sport, <u>number, universityName</u>)
University(<u>name</u>)

Integrity Constraints

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.

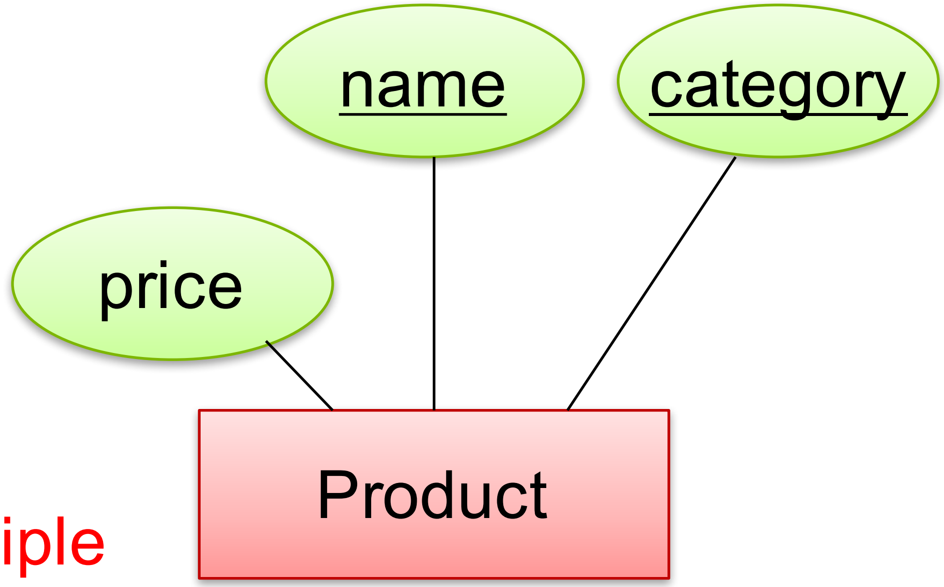
Single-value constraints: a person can have only one father.

Referential integrity constraints: if you work for a company, it must exist in the database.

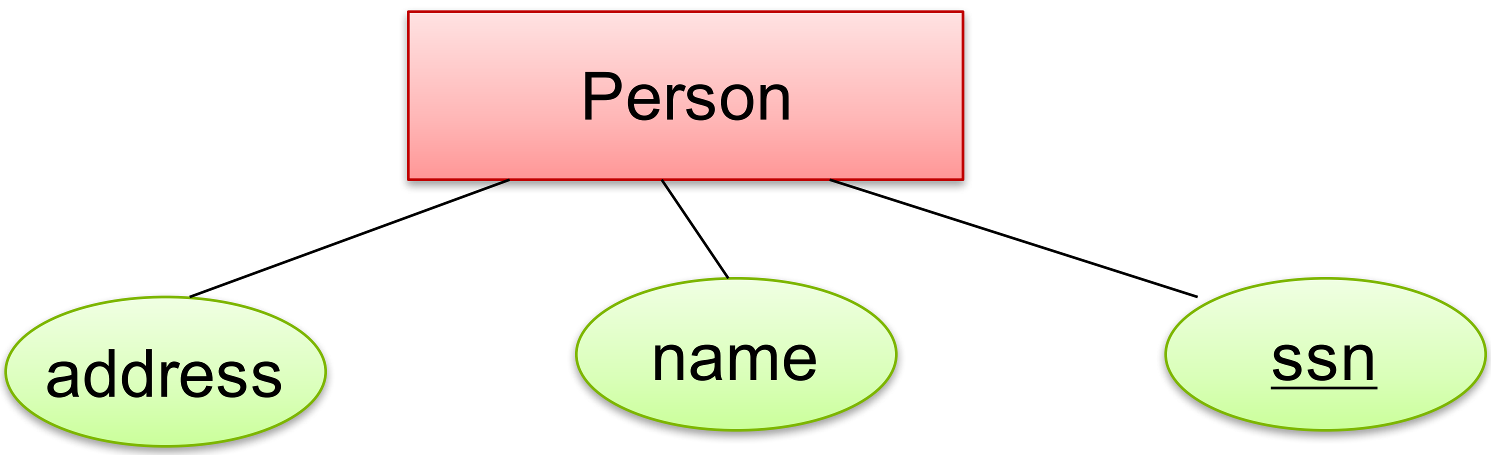
Other constraints: peoples' ages are between 0 and 150.

Keys in E/R Diagrams

Underline:



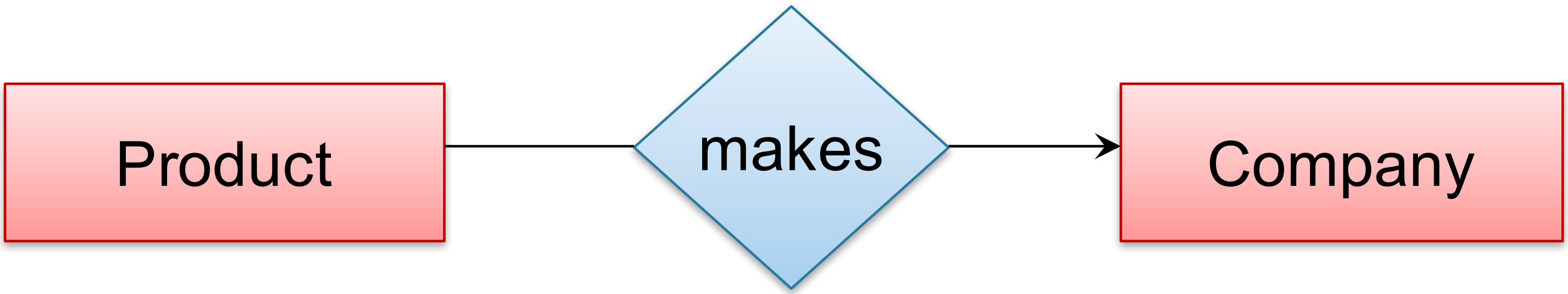
No formal way
to specify multiple
keys in E/R diagrams



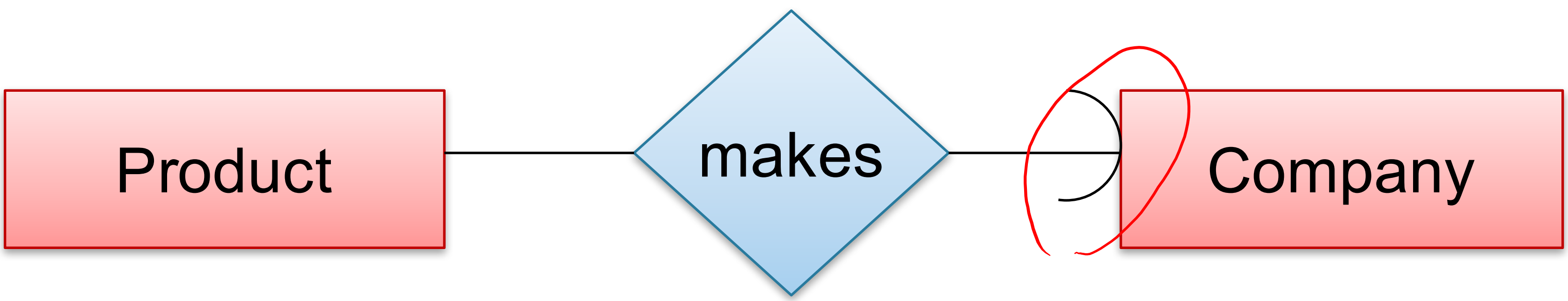
Single Value Constraints

vs.

Referential Integrity Constraints

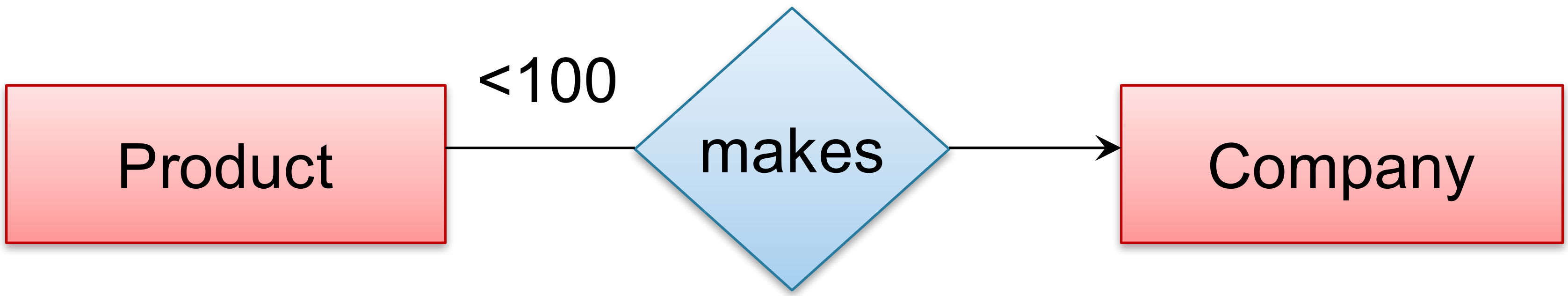


Each product made by at most one company.
Some products made by no company



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

simplest

Most complex

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

Key Constraints

Product(<u>name</u>, 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(<u>name, category</u>, 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

Product

Name	Category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

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  )
```

Product

Name	Category
~~Gizmo~~ ?	gadget
Camera	Photo
OneClick	Photo

Purchase

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 ?

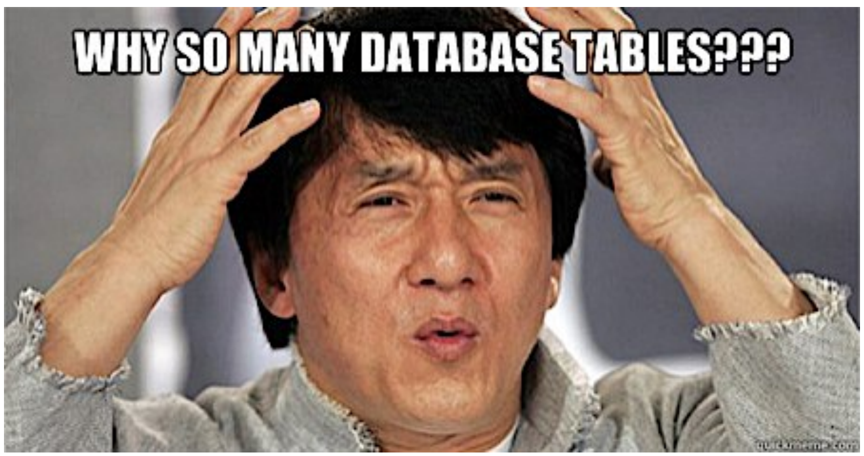
General Assertions

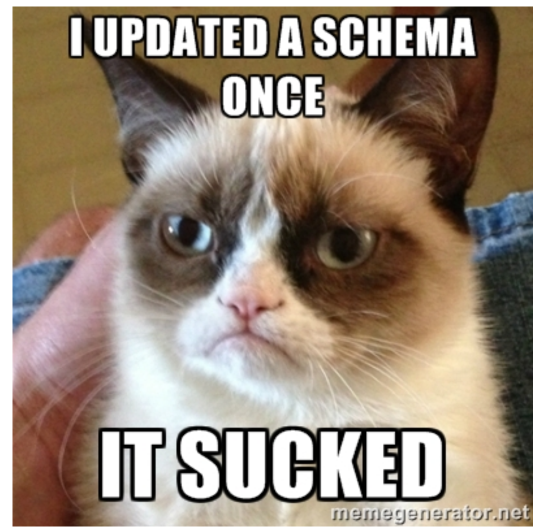
```
CREATE ASSERTION myAssert CHECK
  (NOT EXISTS(
        SELECT Product.name
        FROM Product, Purchase
        WHERE Product.name = Purchase.prodName
        GROUP BY Product.name
        HAVING count(*) > 200) )
```

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

Design Theory

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

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 ***functional dependencies*** (FDs)
- Use FDs to ***normalize*** 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** $B_1 .. B_m$

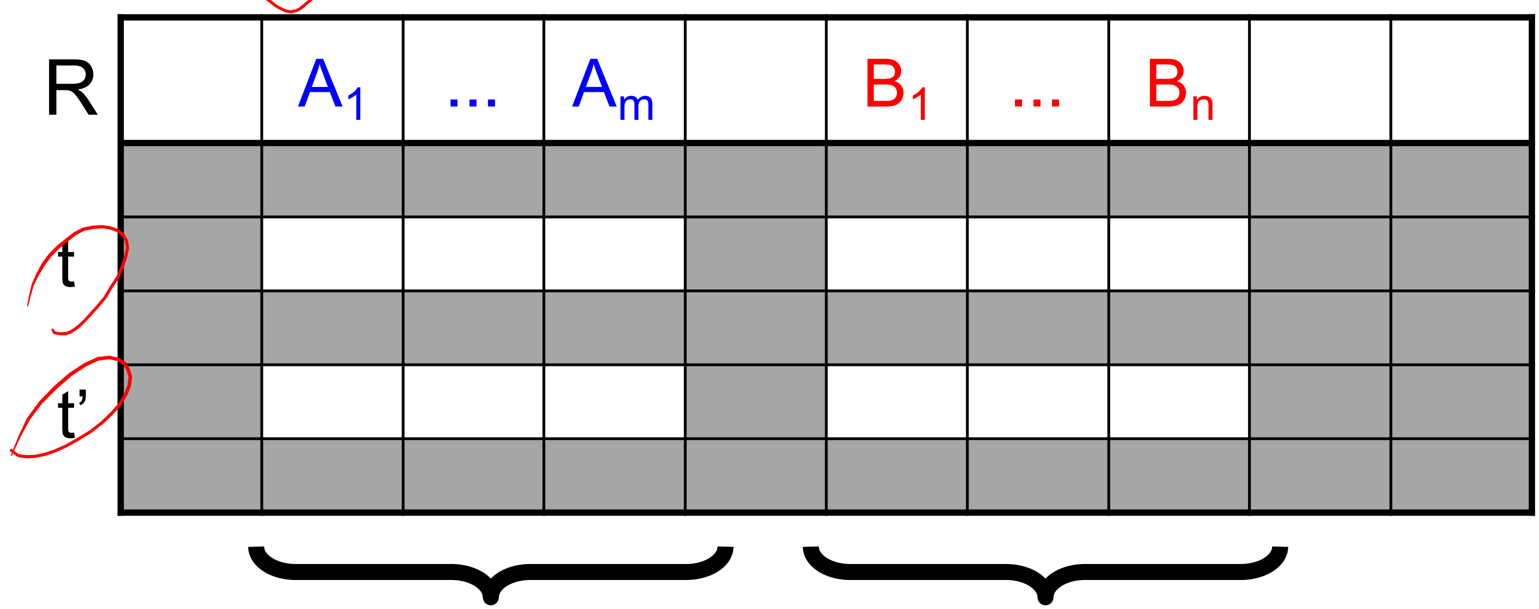
$\boxed{A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m}$

Functional Dependencies (FDs)

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

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

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



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 ?

Buzzwords

- 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**

Why bother with FDs?

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?

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** is the set of attributes B, notated $\{A_1, \ldots, A_n\}^+$,
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}$^+$ =
  { name, category, color, department, price }

Hence: $\boxed{\text{name, category} \rightarrow \text{color, department, price}}$

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 ?)

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

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

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

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 reducing to the minimal X's to get the key

Example

Product(name, price, category, color)

name, category → price
category → color

What is the key ?

Example

Product(name, price, category, color)

$$\boxed{\begin{array}{l} \text{name, category} \rightarrow \text{price} \\ \text{category} \rightarrow \text{color} \end{array}}$$

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 distinct 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 distinct 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

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?

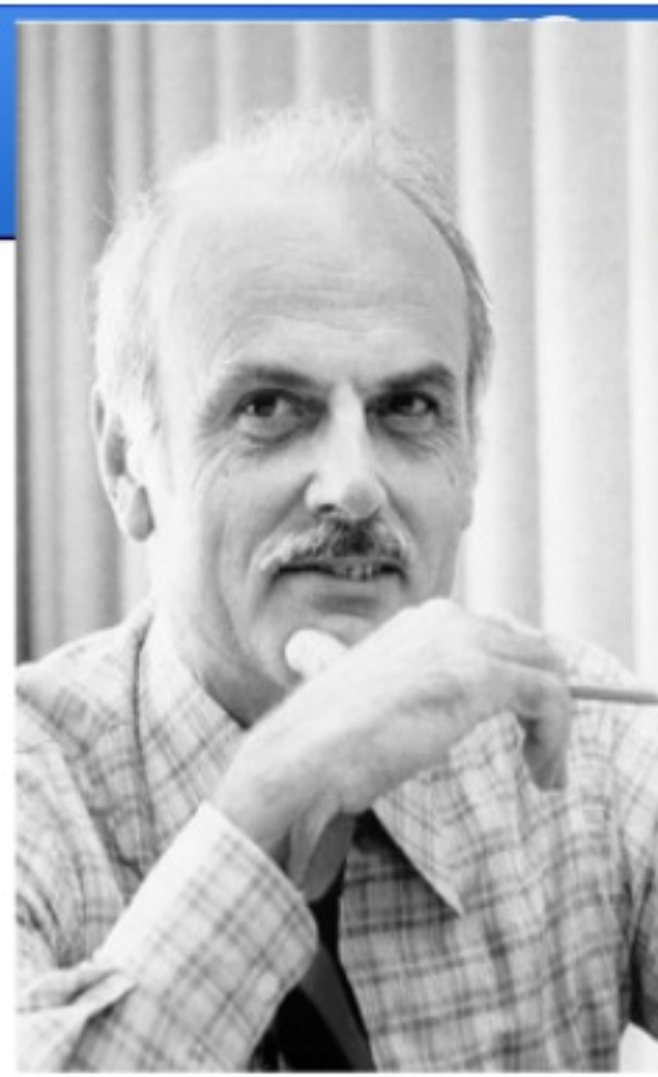
Boyce-Codd Normal Form

Boyce-Codd Normal Form

Dr. Raymond F. Boyce

Edgar Frank "Ted" Codd

"A Relational Model of Data for Large Shared Data Banks"



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.

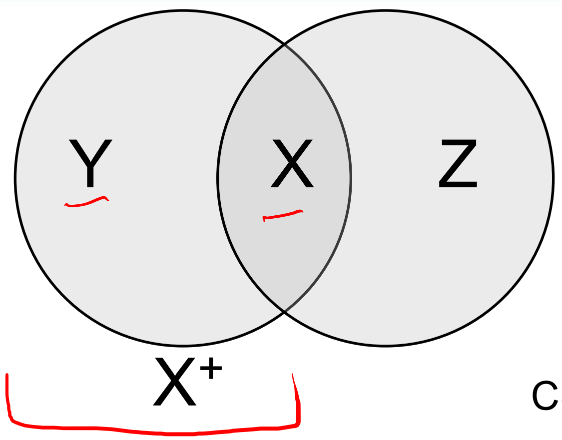
Equivalently:

Definition. A relation R is in BCNF if:

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

BCNF Decomposition Algorithm

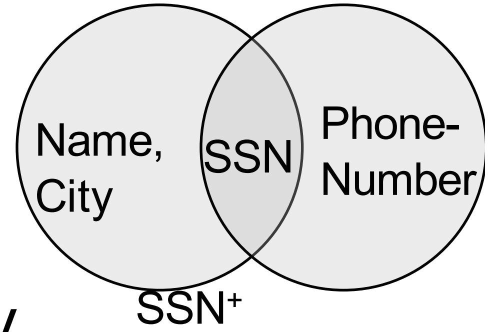
Normalize(R)
  find X s.t.: $X \neq X^+$ and $X^+ \neq$ [all attributes]
  <u>if</u> (not found) **<u>then</u>** "R is in BCNF"
  <u>let</u> $Y = X^+ - X$; $Z =$ [all attributes] $- X^+$
  decompose R into $R1(X \cup Y)$ and $R2(X \cup Z)$
  Normalize(R1); Normalize(R2);



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

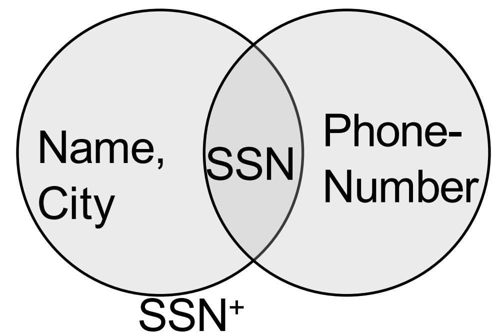
In other words:

SSN+ = 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 → 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.: $X \neq X^+$ and $X^+ \neq$ [all attributes]

Example BCNF Decomposition

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

SSN → name, age

age → hairColor

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

Example BCNF Decomposition

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

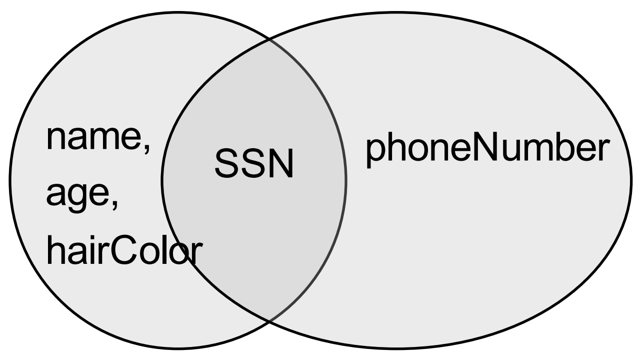
SSN → name, age

age → hairColor

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

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

Phone(SSN, phoneNumber)



Find X s.t.: $X \neq X^+$ and $X^+ \neq$ [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.: $X \neq X^+$ and $X^+ \neq$ [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)

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

Example: BCNF

R(A,B,C,D)

R(A,B,C,D)

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

Example: BCNF

Recall: find X s.t.
$X \subsetneq X^+ \subsetneq$ [all-attrs]

R(A,B,C,D)

R(A,B,C,D)

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

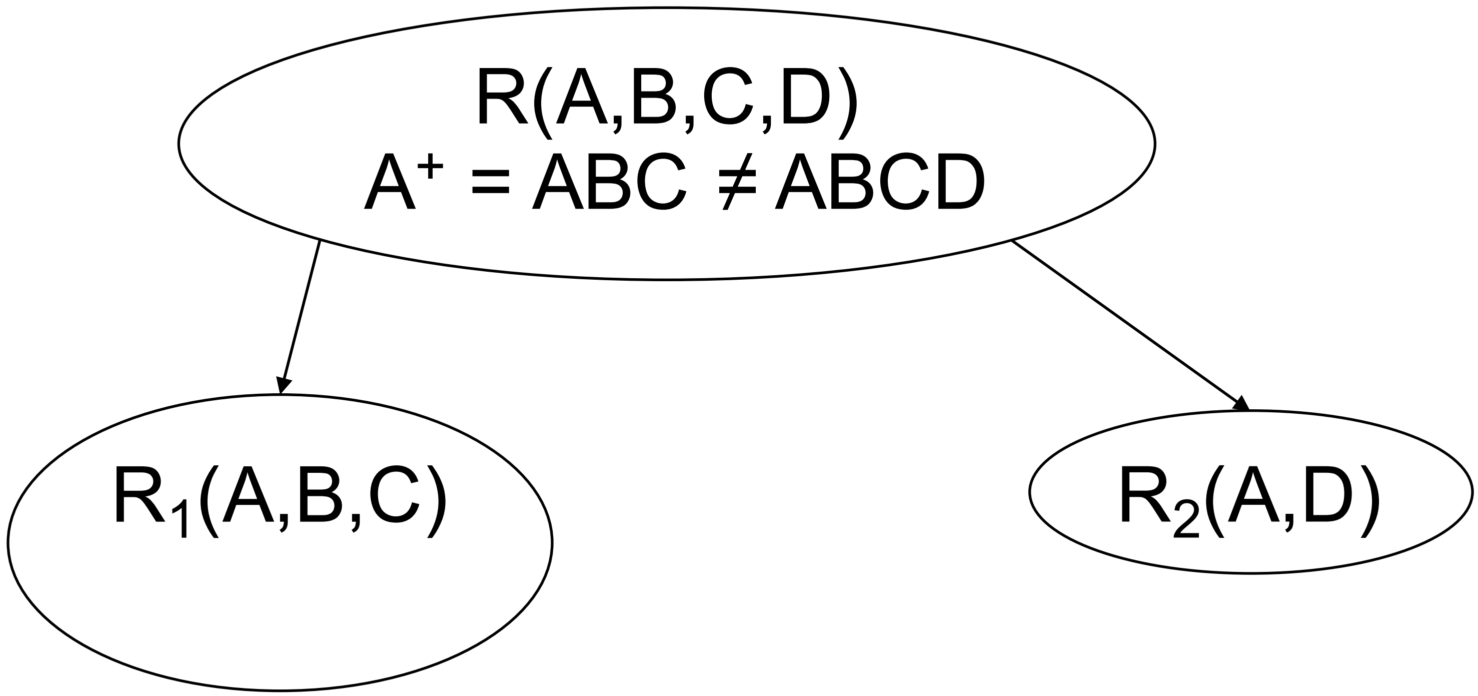
Example: BCNF

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

R(A,B,C,D)

Example: BCNF

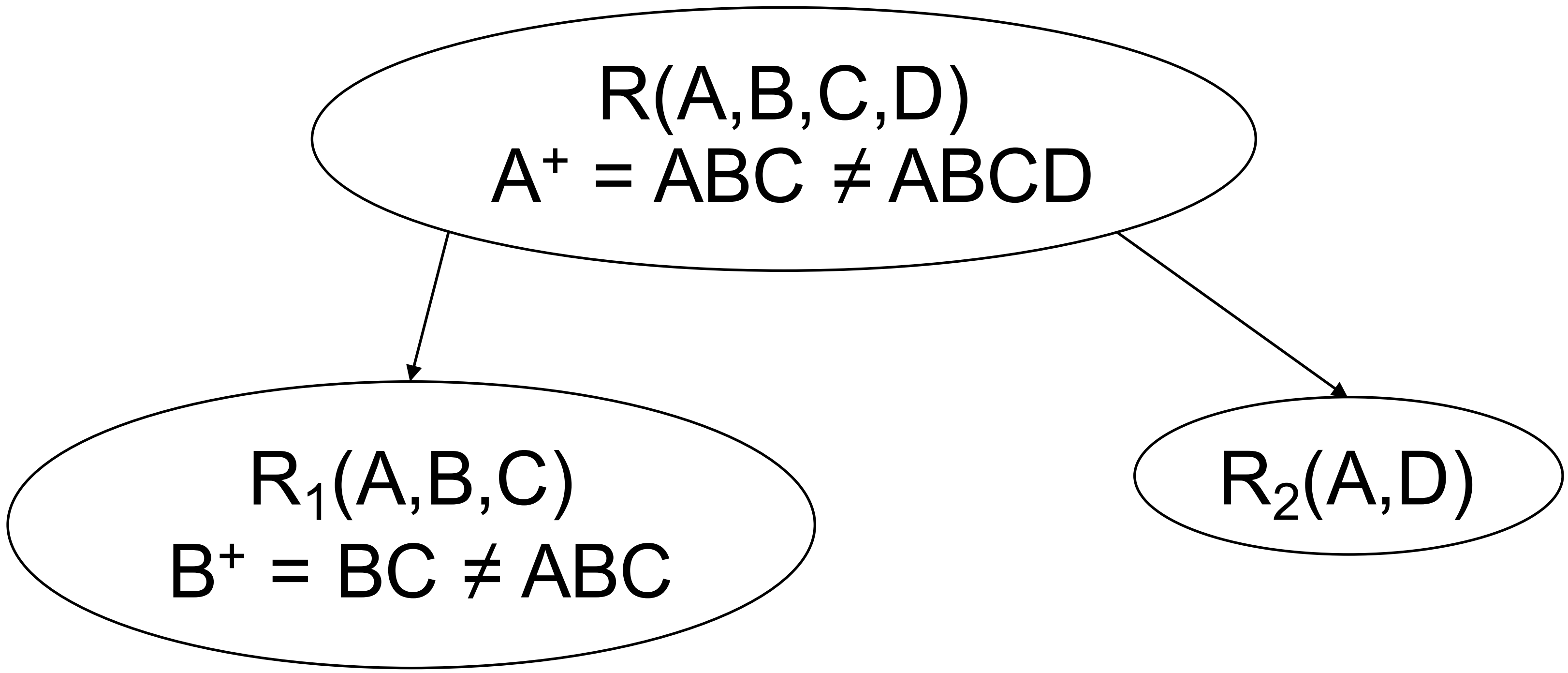
$A \rightarrow B$
$B \rightarrow C$



R(A,B,C,D)

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

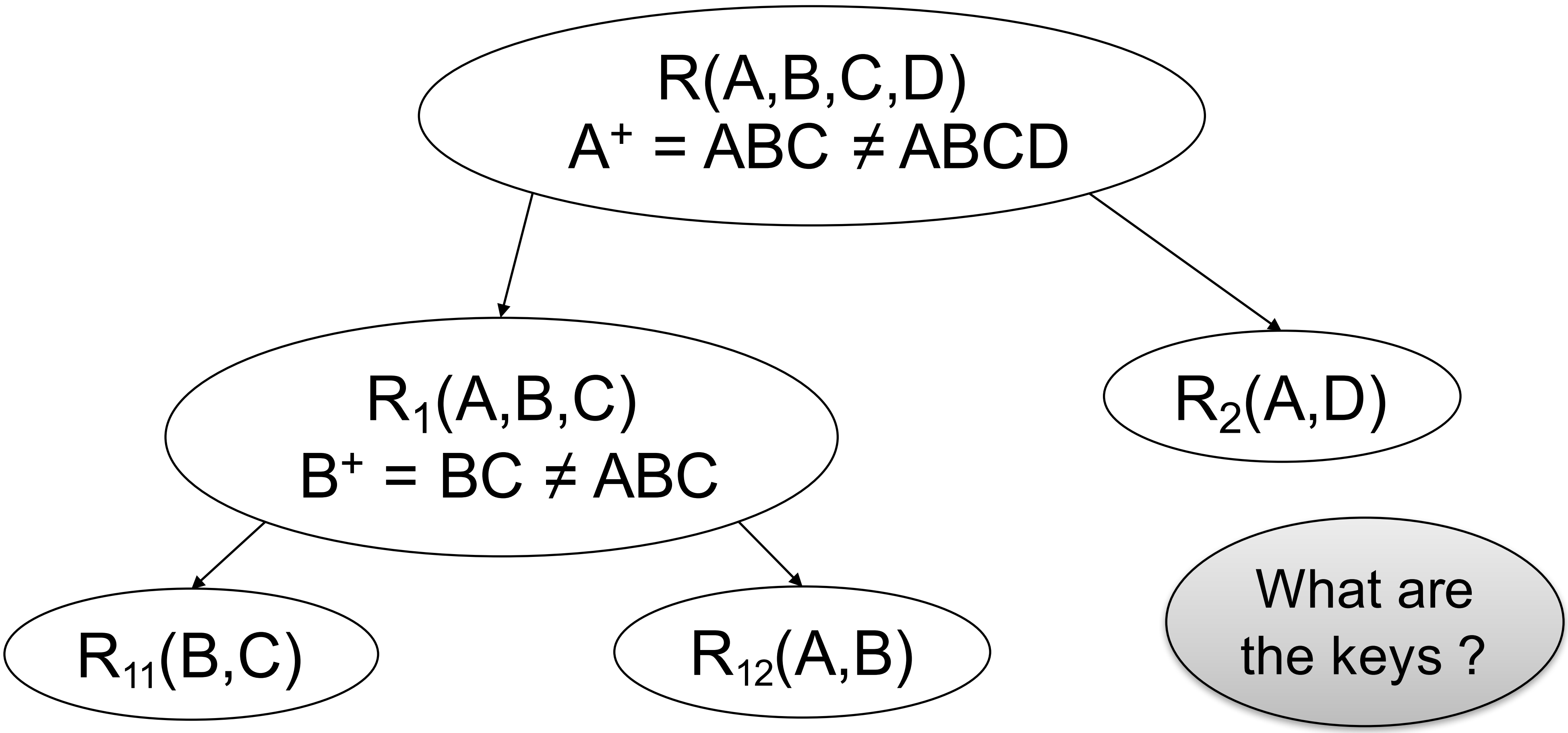
Example: BCNF



R(A,B,C,D)

Example: BCNF

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



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

Decompositions in General

$$\begin{tikzcd}[column sep=tiny] & \boxed{R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_p)} \arrow[dl] \arrow[dr] & \\ \boxed{S_1(A_1, \ldots, A_n, B_1, \ldots, B_m)} & & \boxed{S_2(A_1, \ldots, A_n, C_1, \ldots, C_p)} \end{tikzcd}$$

S_1 = projection of R on $A_1, \ldots, A_n, B_1, \ldots, B_m$

S_2 = projection of R on $A_1, \ldots, A_n, C_1, \ldots, C_p$

Lossless Decomposition

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
Gizmo	19.99	Camera

Name	Price
Gizmo	19.99
OneClick	24.99
~~Gizmo~~	~~19.99~~

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

Lossy Decomposition

What is lossy here?

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
Gizmo	19.99	Camera

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

Price	Category
19.99	Gadget
24.99	Camera
19.99	Camera

Lossy Decomposition

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
Gizmo	19.99	Camera

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

Price	Category
19.99	Gadget
24.99	Camera
19.99	Camera

Decomposition in General

$R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_p)$

$S_1(A_1, \ldots, A_n, B_1, \ldots, B_m)$ $S_2(A_1, \ldots, A_n, C_1, \ldots, C_p)$

Let: S_1 = projection of R on $A_1, \ldots, A_n, B_1, \ldots, B_m$

S_2 = projection of R on $A_1, \ldots, A_n, C_1, \ldots, C_p$

The decomposition is called *lossless* if $R = S_1 \bowtie S_2$

Fact: If $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$ then the decomposition is lossless

It follows that every BCNF decomposition is lossless

Testing for Lossless Join

If we decompose R into $\Pi_{S1}(R)$, $\Pi_{S2}(R)$, $\Pi_{S3}(R)$, …
Is it true that S1 $\bowtie$ S2 $\bowtie$ S3 $\bowtie$ … = R ?

That is true if we can show that:

$R \subseteq S1 \bowtie S2 \bowtie S3 \bowtie \ldots$

$R \supseteq S1 \bowtie S2 \bowtie S3 \bowtie \ldots$

Example from textbook Ch. 3.4.2

The Chase Test for Lossless Join

$R(A,B,C,D) = S1(A,D) \bowtie S2(A,C) \bowtie S3(B,C,D)$

R satisfies: $A \to B$, $B \to C$, $CD \to A$

$S1 = \Pi_{AD}(R)$, $S2 = \Pi_{AC}(R)$, $S3 = \Pi_{BCD}(R)$,

hence $R \subseteq S1 \bowtie S2 \bowtie S3$

Need to check: $R \supseteq S1 \bowtie S2 \bowtie S3$

Example from textbook Ch. 3.4.2

The Chase Test for Lossless Join

$R(A,B,C,D) = S1(A,D) \bowtie S2(A,C) \bowtie S3(B,C,D)$
R satisfies: $A \rightarrow B$, $B \rightarrow C$, $CD \rightarrow A$

$S1 = \Pi_{AD}(R)$, $S2 = \Pi_{AC}(R)$, $S3 = \Pi_{BCD}(R)$,
hence $R \subseteq S1 \bowtie S2 \bowtie S3$
Need to check: $R \supseteq S1 \bowtie S2 \bowtie S3$
Suppose $(a,b,c,d) \in S1 \bowtie S2 \bowtie S3$ Is it also in R?
R must contain the following tuples:

A	B	C	D
a	b1	c1	d

Why ?
$(a,d) \in S1 = \Pi_{AD}(R)$

Example from textbook Ch. 3.4.2

The Chase Test for Lossless Join

$R(A,B,C,D) = S1(A,D) \bowtie S2(A,C) \bowtie S3(B,C,D)$
R satisfies: $A \rightarrow B$, $B \rightarrow C$, $CD \rightarrow A$

$S1 = \Pi_{AD}(R)$, $S2 = \Pi_{AC}(R)$, $S3 = \Pi_{BCD}(R)$,
hence $R \subseteq S1 \bowtie S2 \bowtie S3$
Need to check: $R \supseteq S1 \bowtie S2 \bowtie S3$
Suppose $(a,b,c,d) \in S1 \bowtie S2 \bowtie S3$ Is it also in R?
R must contain the following tuples:

A	B	C	D	Why ?
a	b1	c1	d	$(a,d) \in S1 = \Pi_{AD}(R)$
a	b2	c	d2	$(a,c) \in S2 = \Pi_{BD}(R)$

Example from textbook Ch. 3.4.2

The Chase Test for Lossless Join

R(A,B,C,D) = S1(A,D) ⋈ S2(A,C) ⋈ S3(B,C,D)
R satisfies: A→B, B→C, CD→A

S1 = Π_{AD}(R), S2 = Π_{AC}(R), S3 = Π_{BCD}(R),
hence R $\subseteq$ S1 ⋈ S2 ⋈ S3
Need to check: R $\supseteq$ S1 ⋈ S2 ⋈ S3
Suppose (a,b,c,d) $\in$ S1 ⋈ S2 ⋈ S3 Is it also in R?
R must contain the following tuples:

A	B	C	D	Why ?
a	b1	c1	d	(a,d) $\in$ S1 = Π_{AD}(R)
a	b2	c	d2	(a,c) $\in$ S2 = Π_{BD}(R)
a3	b	c	d	(b,c,d) $\in$ S3 = Π_{BCD}(R)

Example from textbook Ch. 3.4.2

The Chase Test for Lossless Join

R(A,B,C,D) = S1(A,D) ⋈ S2(A,C) ⋈ S3(B,C,D)
R satisfies: A→B, B→C, CD→A

$S1 = \Pi_{AD}(R)$, $S2 = \Pi_{AC}(R)$, $S3 = \Pi_{BCD}(R)$,
hence $R \subseteq S1 \bowtie S2 \bowtie S3$
Need to check: $R \supseteq S1 \bowtie S2 \bowtie S3$
Suppose $(a,b,c,d) \in S1 \bowtie S2 \bowtie S3$ Is it also in R?
R must contain the following tuples:

A	B	C	D	Why ?
a	b1	c1	d	$(a,d) \in S1 = \Pi_{AD}(R)$
a	b2	c	d2	$(a,c) \in S2 = \Pi_{BD}(R)$
a3	b	c	d	$(b,c,d) \in S3 = \Pi_{BCD}(R)$

"Chase" them (apply FDs):

A→B

A	B	C	D
a	b1	c1	d
a	b1	c	d2
a3	b	c	d

Example from textbook Ch. 3.4.2

The Chase Test for Lossless Join

$R(A,B,C,D) = S1(A,D) \bowtie S2(A,C) \bowtie S3(B,C,D)$
R satisfies: $A \rightarrow B$, $B \rightarrow C$, $CD \rightarrow A$

$S1 = \Pi_{AD}(R)$, $S2 = \Pi_{AC}(R)$, $S3 = \Pi_{BCD}(R)$,
hence $R \subseteq S1 \bowtie S2 \bowtie S3$
Need to check: $R \supseteq S1 \bowtie S2 \bowtie S3$
Suppose $(a,b,c,d) \in S1 \bowtie S2 \bowtie S3$ Is it also in R?
R must contain the following tuples:

A	B	C	D	Why ?
a	b1	c1	d	$(a,d) \in S1 = \Pi_{AD}(R)$
a	b2	c	d2	$(a,c) \in S2 = \Pi_{BD}(R)$
a3	b	c	d	$(b,c,d) \in S3 = \Pi_{BCD}(R)$

"Chase" them (apply FDs):

$A \rightarrow B$

A	B	C	D
a	b1	c1	d
a	b1	c	d2
a3	b	c	d

$B \rightarrow C$

A	B	C	D
a	b1	c	d
a	b1	c	d2
a3	b	c	d

Example from textbook Ch. 3.4.2

The Chase Test for Lossless Join

$R(A,B,C,D) = S1(A,D) \bowtie S2(A,C) \bowtie S3(B,C,D)$
R satisfies: $A \rightarrow B$, $B \rightarrow C$, $CD \rightarrow A$

$S1 = \Pi_{AD}(R)$, $S2 = \Pi_{AC}(R)$, $S3 = \Pi_{BCD}(R)$,
hence $R \subseteq S1 \bowtie S2 \bowtie S3$
Need to check: $R \supseteq S1 \bowtie S2 \bowtie S3$
Suppose $(a,b,c,d) \in S1 \bowtie S2 \bowtie S3$ Is it also in R?
R must contain the following tuples:

A	B	C	D
a	b1	c1	d
a	b2	c	d2
a3	b	c	d

Why ?
$(a,d) \in S1 = \Pi_{AD}(R)$
$(a,c) \in S2 = \Pi_{BD}(R)$
$(b,c,d) \in S3 = \Pi_{BCD}(R)$

"Chase" them (apply FDs):

$A \rightarrow B$

A	B	C	D
a	b1	c1	d
a	b1	c	d2
a3	b	c	d

$B \rightarrow C$

A	B	C	D
a	b1	c	d
a	b1	c	d2
a3	b	c	d

$CD \rightarrow A$

A	B	C	D
a	b1	c	d
a	b1	c	d2
a	b	c	d

Hence R contains (a,b,c,d)

Schema Refinements = Normal Forms

- 1st Normal Form = all tables are flat
- 2nd Normal Form = obsolete
- Boyce Codd Normal Form = no bad FDs
- 3rd Normal Form = see book
  - BCNF is lossless but can cause loss of ability to check some FDs (see book 3.4.4)
  - 3NF fixes that (is lossless and dependency-preserving), but some tables might not be in BCNF – i.e., they may have redundancy anomalies

Getting Practical

How to implement normalization in SQL

Motivation

- We learned about how to normalize tables to avoid anomalies

- How can we implement normalization in SQL if we can’t modify existing tables?
  - This might be due to legacy applications that rely on previous schemas to run

Views

- A view in SQL =
  - A table computed from other tables, s.t., whenever the base tables are updated, the view is updated too
- More generally:
  - A view is derived data that keeps track of changes in the original data
- Compare:
  - A function computes a value from other values, but does not keep track of changes to the inputs

Purchase(customer, product, store)
Product(<u>pname</u>, price)

StorePrice(store, price)

A Simple View

Create a view that returns for each store
the prices of products purchased at that store

```
CREATE VIEW  StorePrice AS
  SELECT DISTINCT x.store, y.price
  FROM  Purchase x, Product y
  WHERE x.product = y.pname
```

This is like a new table
StorePrice(store,price)

Purchase(customer, product, store)
Product(pname, price)

StorePrice(store, price)

We Use a View Like Any Table

- A "high end" store is a store that sell some products over 1000.
- For each customer, return all the high end stores that they visit.

```
SELECT DISTINCT u.customer, u.store
FROM Purchase u, StorePrice v
WHERE u.store = v.store
    AND v.price > 1000
```

Types of Views

- Virtual views
  - Computed only on-demand – slow at runtime
  - Always up to date
- Materialized views
  - Pre-computed offline – fast at runtime
  - May have stale data (must recompute or update)
  - Indexes *are* materialized views
- A key component of physical tuning of databases is the selection of materialized views and indexes

Vertical Partitioning

Resumes

SSN	**Name**	**Address**	**Resume**	**Picture**
234234	Mary	Huston	Clob1…	Blob1…
345345	Sue	Seattle	Clob2…	Blob2…
345343	Joan	Seattle	Clob3…	Blob3…
432432	Ann	Portland	Clob4…	Blob4…

T1

SSN	**Name**	**Address**
234234	Mary	Huston
345345	Sue	Seattle
. . .		

T2

SSN	**Resume**
234234	Clob1…
345345	Clob2…

T3

SSN	**Picture**
234234	Blob1…
345345	Blob2…

T2.SSN is a key *and* a foreign key to **T1**.SSN. Same for **T3**.SSN

T1(ssn,name,address)
T2(ssn,resume)
T3(ssn,picture)

Resumes(ssn,name,address,resume,picture)

Vertical Partitioning

```
CREATE VIEW  Resumes  AS
  SELECT  T1.ssn, T1.name, T1.address,
          T2.resume, T3.picture
  FROM    T1,T2,T3
  WHERE   T1.ssn=T2.ssn AND T1.ssn=T3.ssn
```

T1(ssn,name,address)
T2(ssn,resume)
T3(ssn,picture)

Resumes(ssn,name,address,resume,picture)

Vertical Partitioning

```
CREATE VIEW Resumes AS
  SELECT  T1.ssn, T1.name, T1.address,
          T2.resume, T3.picture
  FROM    T1,T2,T3
  WHERE   T1.ssn=T2.ssn AND T1.ssn=T3.ssn
```

```
SELECT address
FROM   Resumes
WHERE  name = 'Sue'
```

T1(ssn,name,address)
T2(ssn,resume)
T3(ssn,picture)

Resumes(ssn,name,address,resume,picture)

Vertical Partitioning

```
CREATE VIEW Resumes AS
  SELECT  T1.ssn, T1.name, T1.address,
          T2.resume, T3.picture
  FROM    T1,T2,T3
  WHERE   T1.ssn=T2.ssn AND T1.ssn=T3.ssn
```

```
SELECT address
FROM   Resumes
WHERE  name = 'Sue'
```

Original query:

```
SELECT T1.address
FROM T1, T2, T3
WHERE T1.name = 'Sue'
   AND T1.SSN=T2.SSN
   AND T1.SSN = T3.SSN
```

T1(ssn,name,address)
T2(ssn,resume)
T3(ssn,picture)

Resumes(ssn,name,address,resume,picture)

Vertical Partitioning

```
CREATE VIEW  Resumes  AS
  SELECT  T1.ssn, T1.name, T1.address,
          T2.resume, T3.picture
  FROM    T1,T2,T3
  WHERE  T1.ssn=T2.ssn AND T1.ssn=T3.ssn
```

```
SELECT address
FROM    Resumes
WHERE   name = 'Sue'
```

Modified query:

SELECT T1.address
FROM T1, ~~T2, T3~~
WHERE T1.name = 'Sue'
~~AND T1.SSN=T2.SSN~~
~~AND T1.SSN = T3.SSN~~

Final query:

```
SELECT T1.address
FROM T1
WHERE T1.name = 'Sue'
```

Vertical Partitioning Applications

- Advantages
  - Speeds up queries that touch only a small fraction of columns
  - Single column can be compressed effectively, reducing disk I/O

- Disadvantages
  - Updates are expensive!
  - Need many joins to access many columns
  - Repeated key columns add overhead

Horizontal Partitioning

Customers

SSN	Name	City
234234	Mary	Houston
345345	Sue	Seattle
345343	Joan	Seattle
234234	Ann	Portland
--	Frank	Calgary
--	Jean	Montreal

CustomersInHouston

SSN	Name	City
234234	Mary	Houston

CustomersInSeattle

SSN	Name	City
345345	Sue	Seattle
345343	Joan	Seattle

. . . . .

CustomersInHouston(ssn,name,city)
CustomersInSeattle(ssn,name,city)
. . . . .

Customers(ssn,name,city)

Horizontal Partitioning

```
CREATE VIEW Customers AS
    CustomersInHouston
      UNION ALL
    CustomersInSeattle
      UNION ALL
     . . .
```

CustomersInHouston(ssn,name,city)
CustomersInSeattle(ssn,name,city)
. . . . .

Customers(ssn,name,city)

Horizontal Partitioning

```
SELECT name
FROM   Customers
WHERE  city = 'Seattle'
```

Which tables are inspected by the system ?

CustomersInHouston(<u>ssn</u>,name,city)
CustomersInSeattle(<u>ssn</u>,name,city)
. . . . .

Customers(<u>ssn</u>,name,city)

Horizontal Partitioning

Better: remove CustomerInHouston.city etc

```
CREATE VIEW  Customers  AS
    (SELECT SSN, name, 'Houston' as city
     FROM CustomersInHouston)
       UNION ALL
    (SELECT SSN, name, 'Seattle' as city
     FROM CustomersInSeattle)
       UNION ALL
     . . .
```

CustomersInHouston(ssn,name,city)
CustomersInSeattle(ssn,name,city)
. . . . . .

Customers(ssn,name,city)

Horizontal Partitioning

```
SELECT  name
FROM    Customers
WHERE   city = 'Seattle'
```

↓

```
SELECT name
FROM   CustomersInSeattle
```

Horizontal Partitioning Applications

- Performance optimization
  - Especially for data warehousing
  - E.g., one partition per month
  - E.g., archived applications and active applications
- Distributed and parallel databases
- Data integration

Conclusion

- Poor schemas can lead to performance inefficiencies
- E/R diagrams are means to structurally visualize and design relational schemas
- Normalization is a principled way of converting schemas into a form that avoid such problems
- BCNF is one of the most widely used normalized form in practice