

Introduction to Data Management

CSE 344

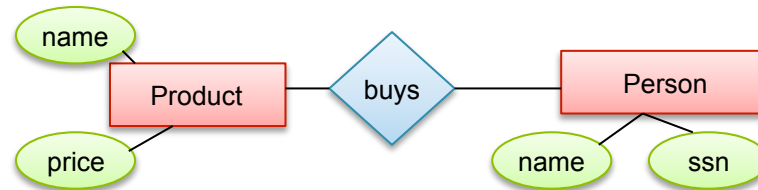
Lectures 17-18: Design Theory

Announcements

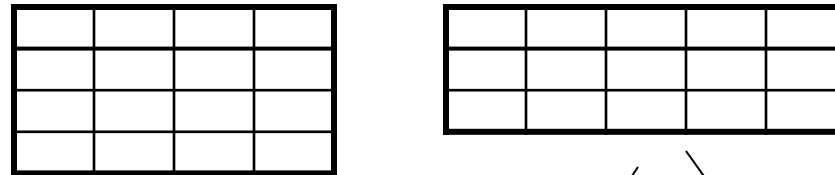
- Midterm on Monday
 - Everything through datalog/ra/rc. XML will be on the final (along with many other things, of course)
 - Review Sunday, Nov. 2 at 2 pm **STANDARD TIME**, EEB 037 (clocks turn back an hour at 2am Sunday)
 - Closed book
- Next web quiz and homework 5 out now, due next week right after the midterm
- Today: Design theory (3.1-3.4)

Relational Schema Design

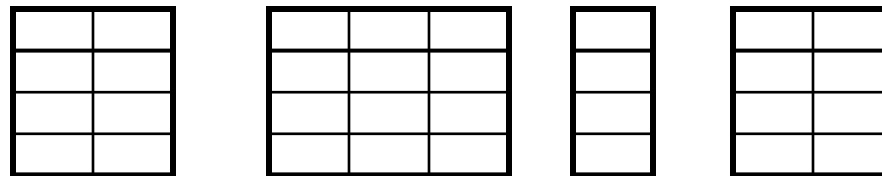
Conceptual Model:



Relational Model:
plus FD's



Normalization:
Eliminates **anomalies**



Relational Schema Design

Name	<u>SSN</u>	<u>PhoneNumber</u>	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	<u>SSN</u>	<u>PhoneNumber</u>	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

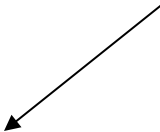
Anomalies:

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

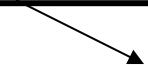
Relation Decomposition

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	<u>SSN</u>	City
Fred	123-45-6789	Seattle
Joe	987-65-4321	Westfield



<u>SSN</u>	<u>PhoneNumber</u>
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, \dots, A_n

then they must also agree on the attributes

B_1, B_2, \dots, B_m

Formally:

$A_1 \dots A_n$ determines $B_1 \dots B_m$

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

Functional Dependencies (FDs)

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

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

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

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 \rightarrow color
category \rightarrow department
color, category \rightarrow 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:

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

Then this FD also holds:

$\text{name, category} \rightarrow \text{price}$

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, \dots, A_n

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

Example:

1. name \rightarrow color
2. category \rightarrow department
3. color, category \rightarrow price

Closures:

$$\text{name}^+ = \{\text{name}, \text{color}\}$$

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

$$\text{color}^+ = \{\text{color}\}$$

Closure Algorithm

$X = \{A_1, \dots, A_n\}$.

Repeat until X doesn't change do:
if $B_1, \dots, B_n \rightarrow C$ is a FD **and**
 B_1, \dots, B_n are all in X
then add C to X.

Example:

1. name \rightarrow color
2. category \rightarrow department
3. color, category \rightarrow price

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

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

Example

In class:

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

A, B	→	C
A, D	→	E
B	→	D
A, F	→	B

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

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

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

Practice at Home

Find all FD's implied by:

$$\begin{array}{l} A, B \rightarrow C \\ A, D \rightarrow B \\ B \rightarrow D \end{array}$$

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, \dots, A_n s.t. for any other attribute B , we have $A_1, \dots, A_n \rightarrow B$
- 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 $X^+ = [\text{all attributes}]$, 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 ?

Example

Product(name, price, category, color)

name, category \rightarrow price
category \rightarrow color

What is the key ?

$(\text{name, category})^+ = \{ \text{name, category, price, color} \}$

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:

- $X \rightarrow A$ is OK if X is a (super)key
- $X \rightarrow A$ 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 $X^+ = [\text{all attributes}]$

BCNF Decomposition Algorithm

Normalize(R)

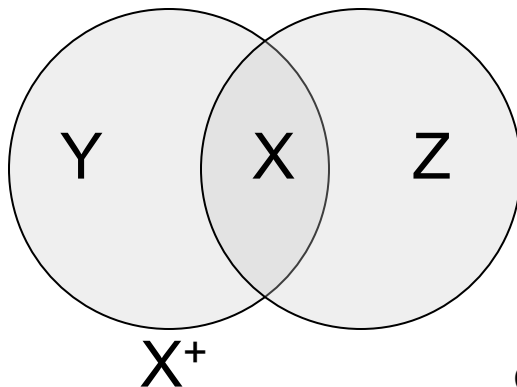
find X s.t.: $X \neq X^+ \neq$ [all attributes]

if (not found) **then** “R is in BCNF”

let $Y = X^+ - X$; $Z =$ [all attributes] $- X^+$

decompose R into $R_1(X \cup Y)$ and $R_2(X \cup Z)$

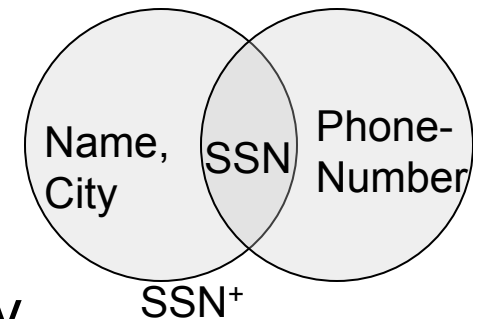
Normalize(R_1); Normalize(R_2);



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

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



The only key is: $\{SSN, \text{PhoneNumber}\}$

Hence $SSN \rightarrow \text{Name, City}$ is a “bad” dependency

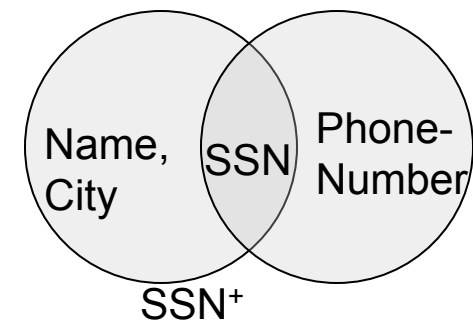
In other words:

$SSN^+ = SSN, \text{Name, City}$ and is neither SSN nor All Attributes

Example BCNF Decomposition

<u>Name</u>	<u>SSN</u>	<u>City</u>
Fred	123-45-6789	Seattle
Joe	987-65-4321	Westfield

SSN \rightarrow Name, City



<u>SSN</u>	<u>PhoneNumber</u>
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 ?

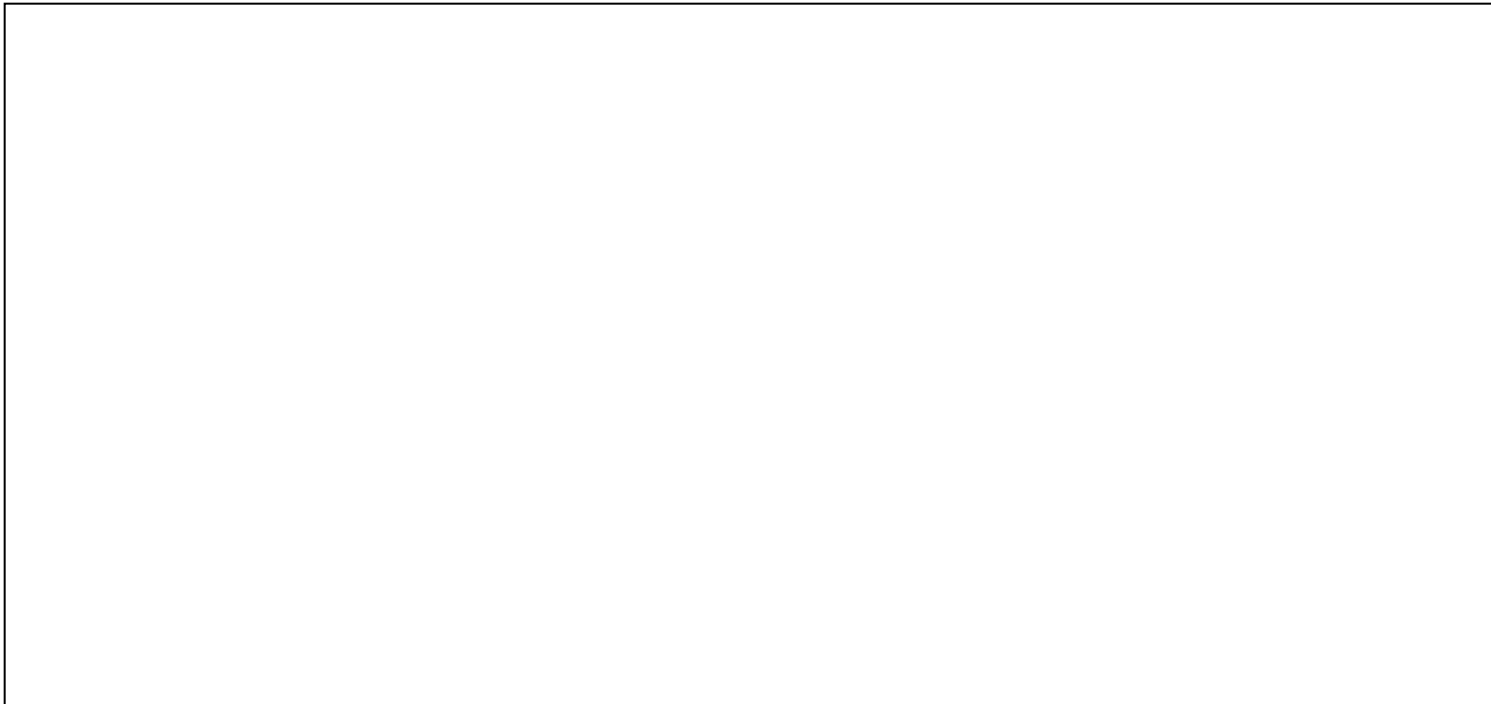
Find X s.t.: $X \neq X^+ \neq$ [all attributes]

Example BCNF Decomposition

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

SSN \rightarrow name, age

age \rightarrow hairColor



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

Example BCNF Decomposition

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

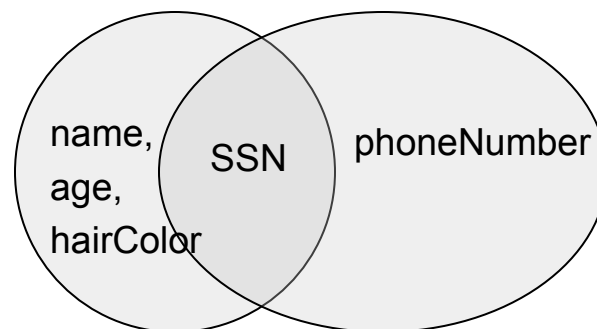
SSN \rightarrow name, age

age \rightarrow 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 \rightarrow name, age

age \rightarrow 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 \rightarrow name, age

age \rightarrow hairColor

Note 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)

R(A,B,C,D)

Example: BCNF

A	→	B
B	→	C

R(A,B,C,D)

$R(A,B,C,D)$

Example: BCNF

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

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

$R(A,B,C,D)$

R(A,B,C,D)

Example: BCNF

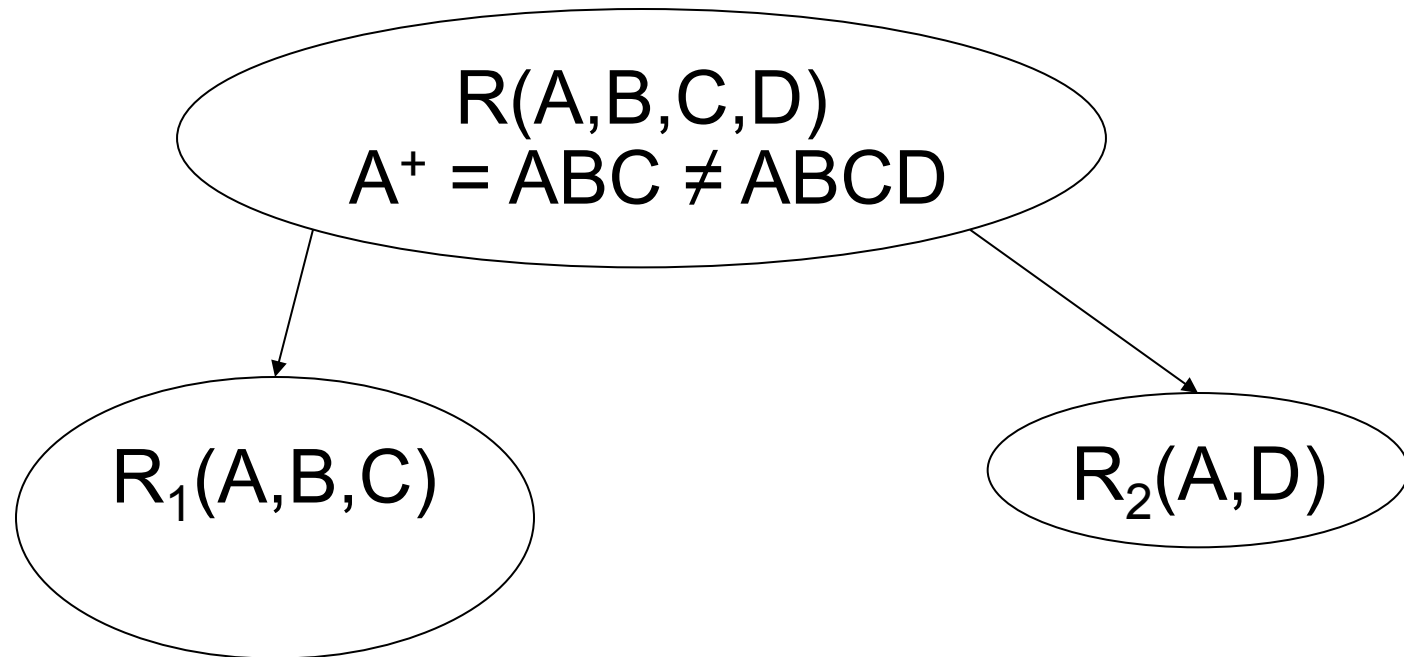
A	→	B
B	→	C

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

R(A,B,C,D)

A → B
B → C

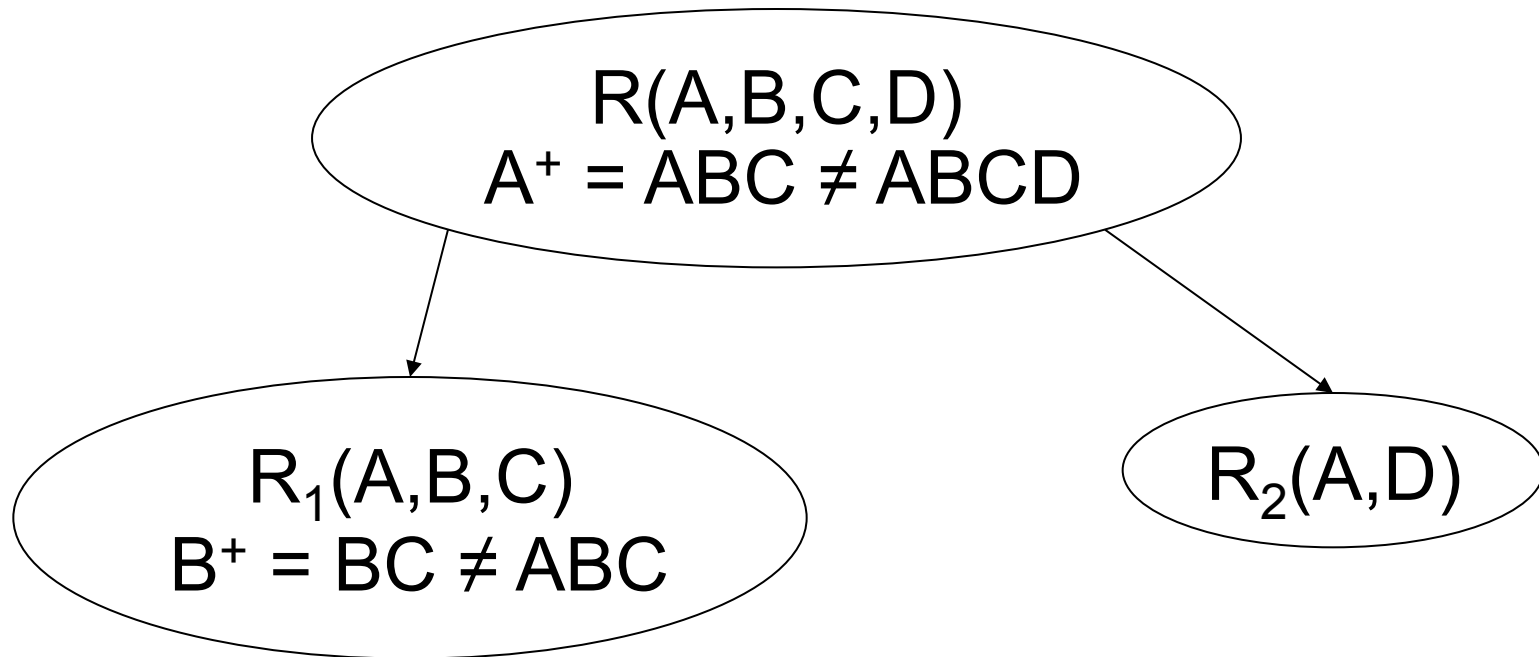
Example: BCNF



R(A,B,C,D)

A → B
B → C

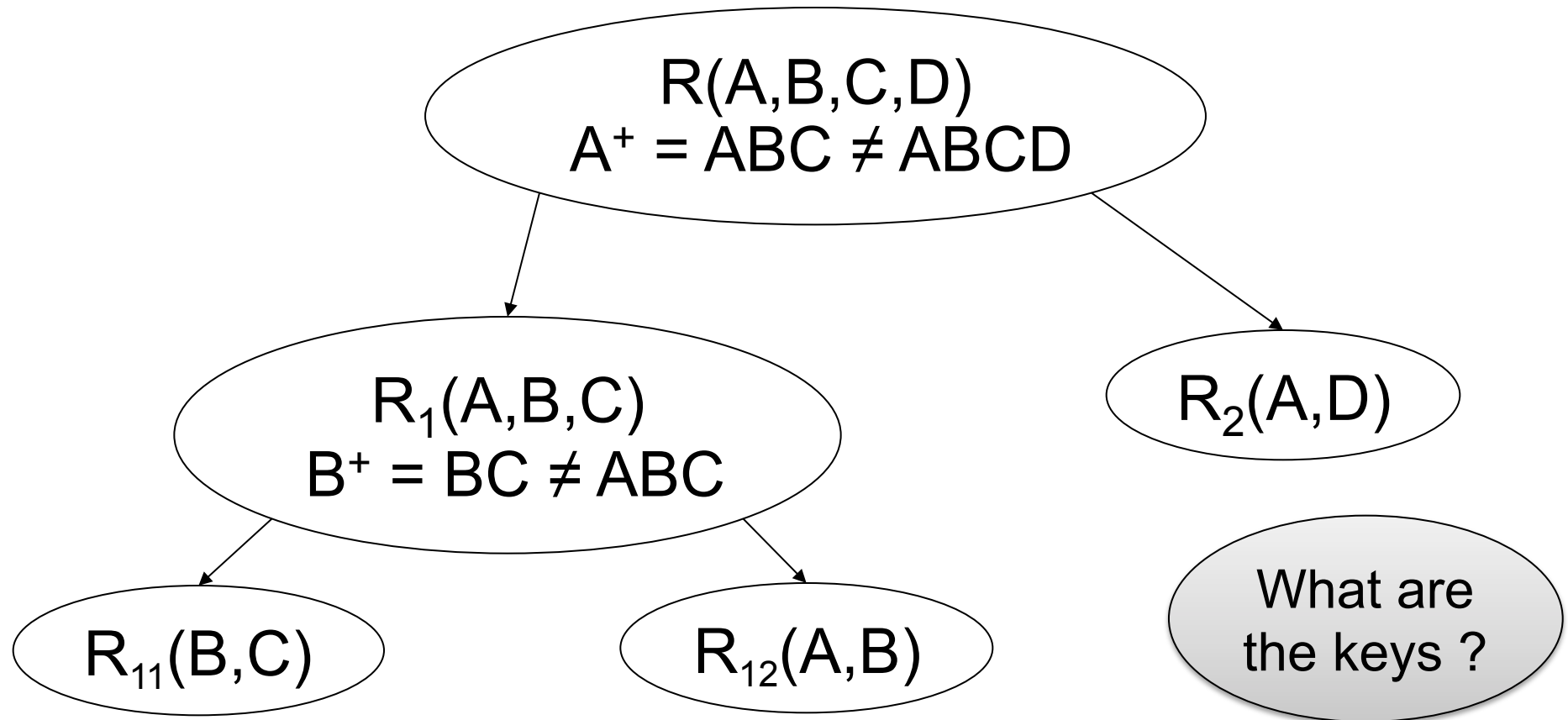
Example: BCNF



R(A,B,C,D)

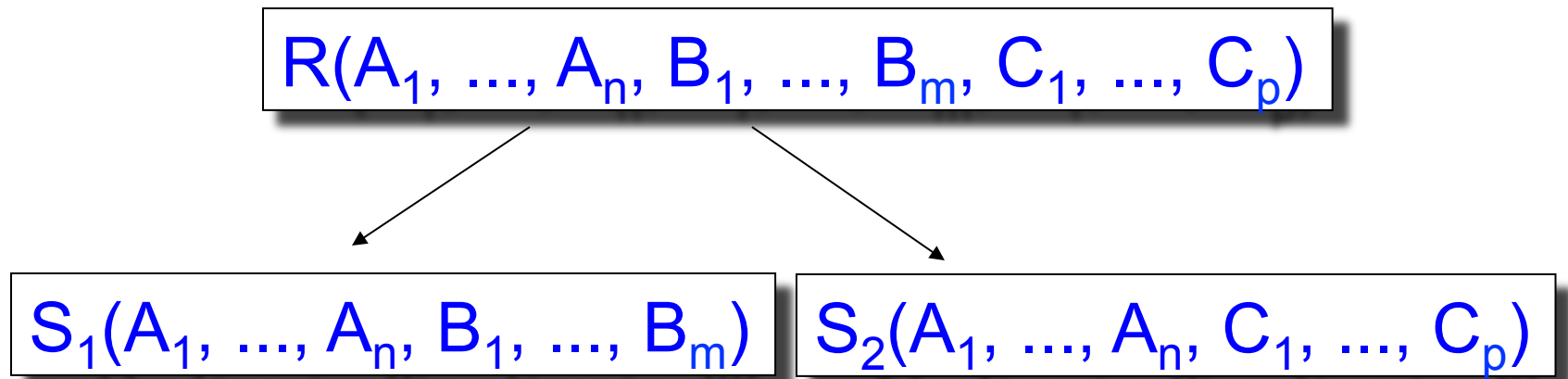
A → B
B → C

Example: BCNF



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

Decompositions in General



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

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

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

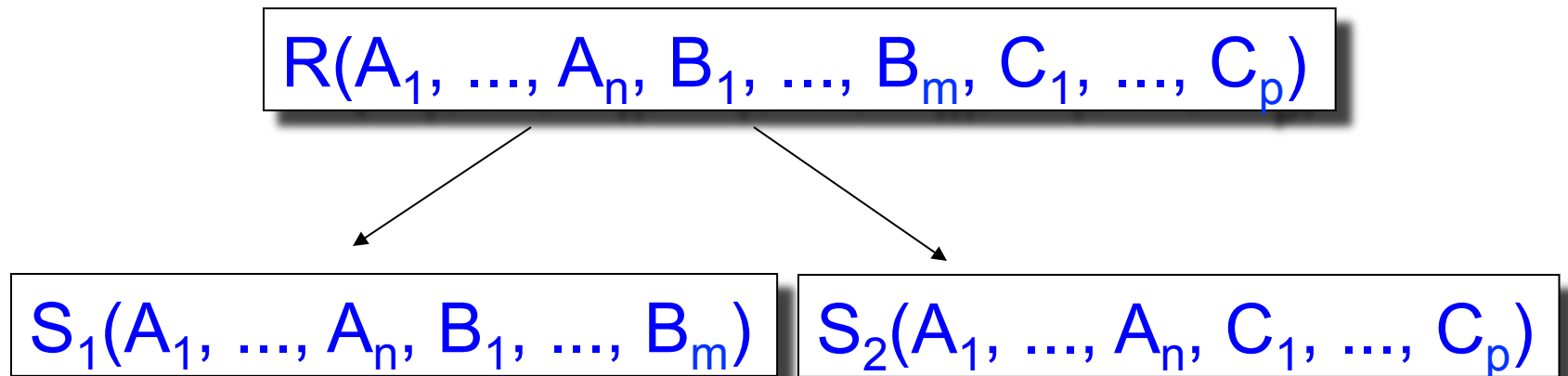
Lossy Decomposition

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



Let: S_1 = projection of R on $A_1, \dots, A_n, B_1, \dots, B_m$
 S_2 = projection of R on $A_1, \dots, A_n, C_1, \dots, C_p$

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

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

It follows that every BCNF decomposition is lossless

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 after joining the relation may not satisfy all original 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

Goal: Find ALL Functional Dependencies

- Anomalies occur when certain “bad” FDs hold
- We know some of the FDs
- Need to find *all* FDs
- Then look for the bad ones

Armstrong's Rules (1/3)

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

Is equivalent to

**Splitting rule
and
Combing rule**

$$\begin{array}{l} A_1, A_2, \dots, A_n \rightarrow B_1 \\ A_1, A_2, \dots, A_n \rightarrow B_2 \\ \dots \\ A_1, A_2, \dots, A_n \rightarrow B_m \end{array}$$

	A_1	...	A_m		B_1	...	B_m	

Armstrong's Rules (2/3)

$$A_1, A_2, \dots, A_n \rightarrow A_i$$

Trivial Rule

where $i = 1, 2, \dots, n$

Why ?

	A_1	...	A_m	

Armstrong's Rules (3/3)

Transitive Rule

If

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

and

$$B_1, B_2, \dots, B_m \rightarrow C_1, C_2, \dots, C_p$$

then

$$A_1, A_2, \dots, A_n \rightarrow C_1, C_2, \dots, C_p$$

Why ?

Armstrong's Rules (3/3)

Illustration

	A_1	...	A_m		B_1	...	B_m		C_1	...	C_p	

Example (continued)

Start from the following FDs:

1. name \rightarrow color
2. category \rightarrow department
3. color, category \rightarrow price

Infer the following FDs:

Inferred FD	Which Rule did we apply ?
4. name, category \rightarrow name	
5. name, category \rightarrow color	
6. name, category \rightarrow category	
7. name, category \rightarrow color, category	
8. name, category \rightarrow price	

Example (continued)

Answers:

1. name \rightarrow color
2. category \rightarrow department
3. color, category \rightarrow price

Inferred FD	Which Rule did we apply ?
4. name, category \rightarrow name	Trivial rule
5. name, category \rightarrow color	Transitivity on 4, 1
6. name, category \rightarrow category	Trivial rule
7. name, category \rightarrow color, category	Split/combine on 5, 6
8. name, category \rightarrow price	Transitivity on 3, 7

THIS IS TOO HARD ! Let's see an easier way.