Database Design and Implementation

CS 645

$\boxed{\text{Schema Refinement}}$

First Normal Form (1NF)

A schema is in 1NF if all tables are flat

Student

Name	GPA	Course
Alice	3.8	Math DB OS
Bob	3.7	DB OS
Carol	3.9	Math OS

→

May need to add keys

Student

Name	GPA
Alice	3.8
Bob	3.7
Carol	3.9

Takes

Student	Course
Alice	Math
Carol	Math
Alice	DB
Bob	DB
Alice	OS
Carol	OS

Course

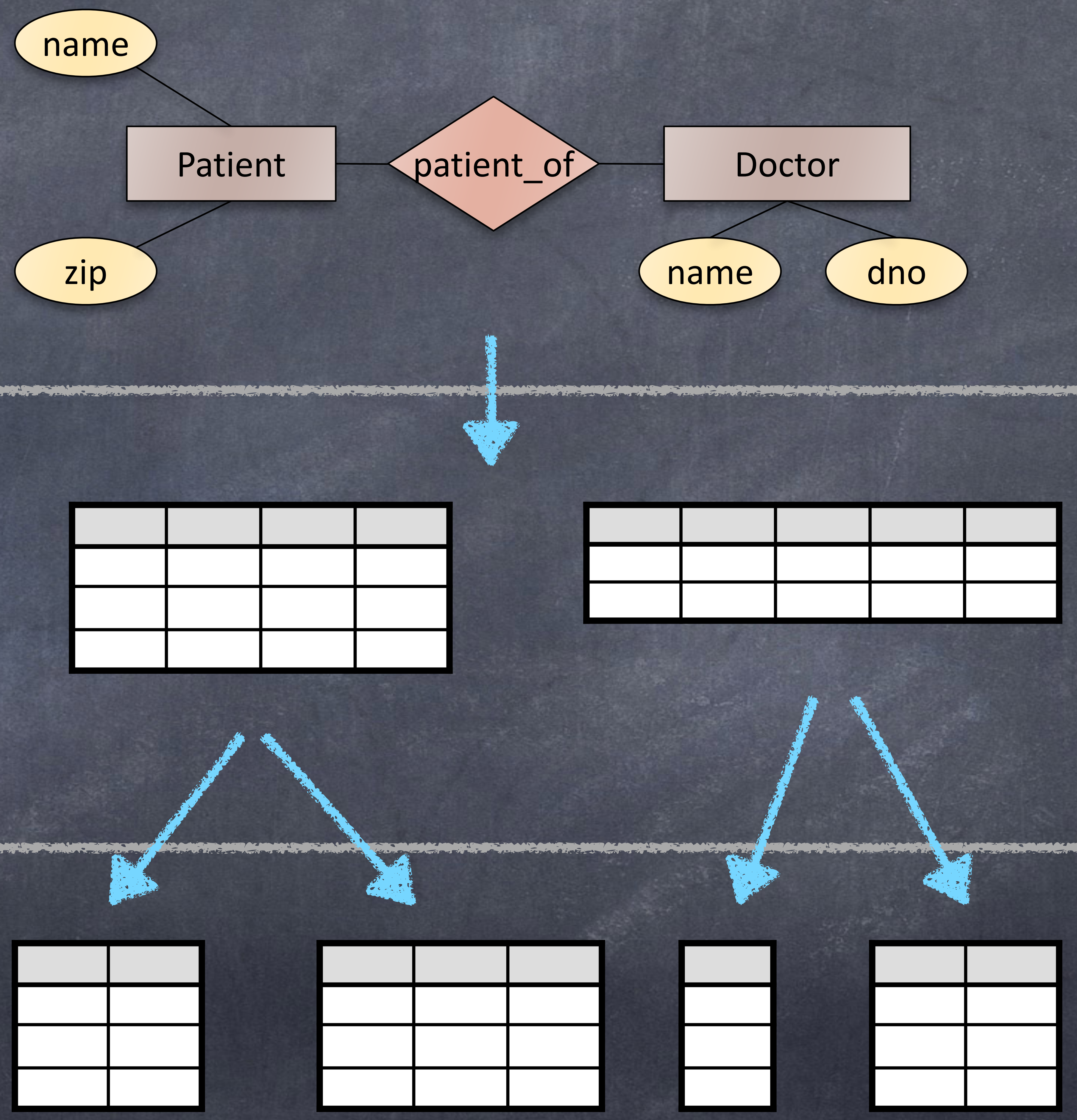
Course
Math
DB
OS

Schema design

Conceptual Model:

Relational Model:
plus FDs
(FD = Functional Dependency)

Normalization:
Eliminates anomalies



Data anomalies

- When a database is poorly designed we get anomalies:
  - *Redundancy*: data is repeated
  - *Update* anomalies: need to change in several places
  - *Delete* anomalies: may lose data when we don't want

Relational schema design

Recall set attributes (persons with several phones):

Name	SSN	PhoneNumber	City
Fred	123-45-6789	413-555-1234	Amherst
Fred	123-45-6789	413-555-6543	Amherst
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)

The above is in 1NF, but what is the problem with this schema?

Relational schema design

Recall set attributes (persons with several phones):

Name	SSN	PhoneNumber	City
Fred	123-45-6789	413-555-1234	Amherst
Fred	123-45-6789	413-555-6543	Amherst
Joe	987-65-4321	908-555-2121	Westfield

Anomalies:

- Redundancy = repeat data
- Update anomalies = what if Fred moves to "Boston"?
- Deletion anomalies = what if Joe deletes his phone number?
  (what if Joe had only one phone #)

Relation decomposition

Break the relation into two:

Name	SSN	PhoneNumber	City
Fred	123-45-6789	413-555-1234	Amherst
Fred	123-45-6789	413-555-6543	Amherst
Joe	987-65-4321	908-555-2121	Westfield

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

<u>SSN</u>	<u>PhoneNumber</u>
123-45-6789	413-555-1234
123-45-6789	413-555-6543
987-65-4321	908-555-2121

Anomalies have gone:

- No more repeated data
- Easy to move Fred to “Boston” (how ?)
- Easy to delete all Joe’s phone numbers (how ?)

Relational schema design (logical design)

- Main idea:
  - Start with some relational schema
  - Find out its functional dependencies (discussed next!)
  - Use them to design a better relational schema

Functional Dependencies

- A form of constraint
  - Hence, part of the schema
- Finding them is part of the database design
- Use them to normalize the relations

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

Example: streetName → zipCode

Example

A FD <u>holds</u>, or <u>does not hold</u> on an instance:

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

Example

A 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

Position → Phone

Example

A 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

But not: Phone → Position

Example

FD's are constraints:
- On some instances they hold
- On others they don't

name → color
category → department
color, category → price

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

Does this instance satisfy all the FDs ?

Example

FD's are constraints:
- On some instances they hold
- On others they don't

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
Gizmo	Stationary	Blue	Supplies	59

What about this one?

FDs and anomalies

- Anomalies occur when certain “bad” FDs hold
- How do we know if a “bad” FD holds?

An interesting observation

If all these FDs are true:

name → color
category → department
color, category → price

Then this FD also holds:

name, category → price

Why ??

Deriving all FDs: Armstrong's Rules

Splitting and combining

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

Trivial

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

Transitive

$$\begin{array}{l} A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m \\ B_1, \ldots, B_m \rightarrow C_1, \ldots, C_p \end{array} \Longrightarrow A_1, \ldots, A_n \rightarrow C_1, \ldots, C_p$$

Example (continued)

Start from the following FDs:

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

Infer the following FDs:

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

Example (continued)

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

Answers:

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

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

Deriving all FDs: Closure

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:

name → color
category → department
color, category → price

Closures:

name^+ = {name, color}

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

color^+ = {color}

Closure algorithm

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

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:

name → color
category → department
color, category → price

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

Hence: name, category → color, department, price

Example

In class:

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

$$\boxed{\begin{array}{ll} A, B & \rightarrow C \\ A, D & \rightarrow E \\ B & \rightarrow D \\ A, F & \rightarrow B \end{array}}$$

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

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

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

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

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

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

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

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

Example

In class:

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

$\boxed{\begin{array}{ll} A, B & \rightarrow C \\ A, D & \rightarrow E \\ B & \rightarrow D \\ A, F & \rightarrow B \end{array}}$

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

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

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

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

Why do we need closure?

- With closure we can find all FDs easily
- To check if $X \rightarrow A$
  - Compute X^+
  - Check if $A \in X^+$

Keys

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

$$A_1, \ldots, A_n \rightarrow B$$

- A key is a minimal superkey
  - i.e., set of attributes which is a superkey and for which no subset is a superkey

Computing (super)keys

- Compute X^+ for all sets X
- If X^+ = all attributes, then X is a superkey
- List only the minimal X's to get the keys

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 ?

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

Hence (name, category) is a key

Eliminating anomalies

Main idea:

- $X \to A$ is OK if X is a (super)key
- $X \to A$ is not OK otherwise

Example

SSN → Name, City

Name	SSN	PhoneNumber	City
Fred	123-45-6789	413-555-1234	Amherst
Fred	123-45-6789	413-555-6543	Amherst
Joe	987-65-4321	908-555-2121	Westfield
Joe	987-65-4321	908-555-1234	Westfield

What is the key? {SSN, PhoneNumber}

Hence SSN → Name,City is a "bad" dependency

Boyce-Codd Normal Form (BCNF)

A simple condition for removing anomalies from relations:

> A relation R is in BCNF if:
>
> If $A_1, \ldots, A_n \rightarrow B$ is a non-trivial dependency in R,
>
> then $\{A_1, \ldots, A_n\}$ is a superkey for R

In other words: there are no "bad" FDs

Equivalently:
for all X, either ($X^+ = X$) or ($X^+ =$ all attributes)

Example (revisited)

SSN → Name, City

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

SSN	PhoneNumber
123-45-6789	413-555-1234
123-45-6789	413-555-6543
987-65-4321	908-555-2121
987-65-4321	908-555-1234

Let's check anomalies:

- Redundancy ?
- Update ?
- Delete ?

Example decomposition

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

FD1: SSN $\rightarrow$ name, age
FD2: age $\rightarrow$ hairColor

Decompose into BCNF:

Example decomposition

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

FD1: SSN → name, age
FD2: age → hairColor

Decompose into BCNF:

What is the key? {SSN, phoneNumber}

But how to decompose?

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

or

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

SSN → name, age, hairColor

or ….

BCNF decomposition algorithm

BCNF_Decompose(R)

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

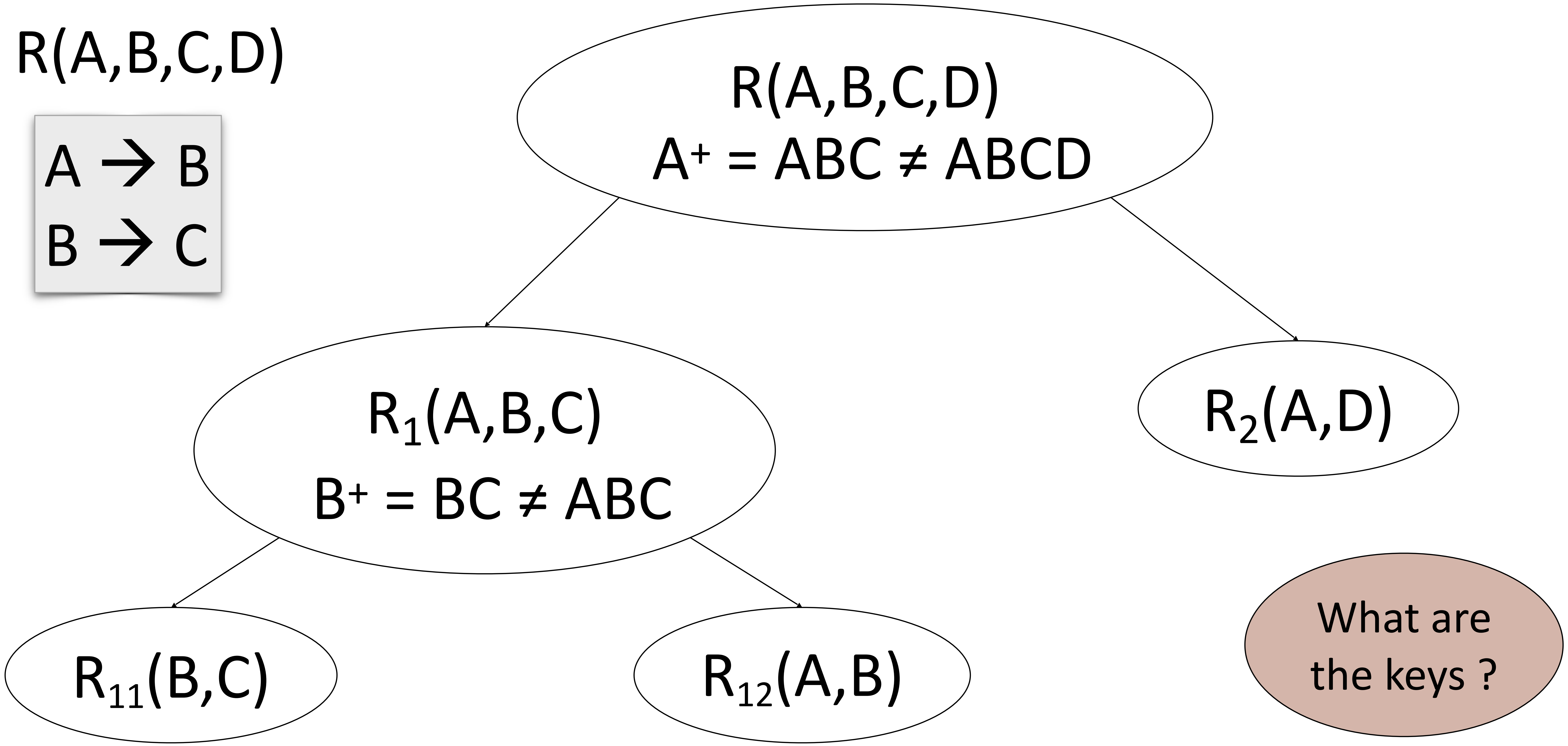
<u>if</u> (not found) **<u>then</u>** "R is in BCNF"

<u>let</u> $Y = X^+ - X$
<u>let</u> $Z =$ [all attributes] $- X^+$
decompose R into $R_1(X \cup Y)$ and $R_2(X \cup Z)$
continue to decompose recursively R_1 and R_2

Example

R(A,B,C,D)

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



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

Decompositions in general

$$
\begin{tikzcd}
& \boxed{R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_p)} \arrow[dl] \arrow[dr] & \\
\boxed{R_1(A_1, \ldots, A_n, B_1, \ldots, B_m)} & & \boxed{R_2(A_1, \ldots, A_n, C_1, \ldots, C_p)}
\end{tikzcd}
$$

R_1 = projection of R on $A_1, \ldots, A_n, B_1, \ldots, B_m$

R_2 = projection of R on $A_1, \ldots, A_n, C_1, \ldots, C_p$

Theory of decomposition

- Sometimes it is correct:

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

Name → Price

Name	Price
Gizmo	19.99
OneClick	24.99

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

Lossless decomposition

Incorrect decomposition

- Sometimes it is not:

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

Name → Price

What's incorrect ??

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

Price	Category
19.99	Gadget
24.99	Camera
19.99	Camera

Lossy decomposition

Decompositions in general

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

$R_1(A_1, \ldots, A_n, B_1, \ldots, B_m)$

$R_2(A_1, \ldots, A_n, C_1, \ldots, C_p)$

If $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$
Then the decomposition is lossless

Note: don't need $A_1, \ldots, A_n \rightarrow C_1, \ldots, C_p$

BCNF decomposition is always lossless. WHY ?

General decomposition goals

1. Elimination of anomalies
2. Recoverability of information
   - Can we get the original relation back?
3. Preservation of dependencies
   - Want to enforce FDs without performing joins

BCNF

3NF

Sometimes cannot decompose into BCNF without losing ability to check some FDs in single relation

BCNF and dependencies

Unit	Company	Product

Unit → Company
Company, Product → Unit

So, there is a BCNF violation, and we decompose.

Unit	Company

Unit → Company

Unit	Product

No FDs

In BCNF we lose the FD

Company, Product → Unit

3NF Motivation

A relation R is in 3rd normal form if :
Whenever there is a nontrivial dependency $A_1, A_2, ..., A_n \rightarrow B$ for R,
then $\{A_1, A_2, ..., A_n\}$ is a super-key for R, or B is part of a key.

Tradeoffs:
BCNF: no anomalies, but may lose some FDs
3NF: keeps all FDs, but may have some anomalies