Introduction to Database Systems CSE 444

Lectures 6-7: Database Design

Outline

- Design theory: 3.1-3.4
  - [Old edition: 3.4-3.6]

Schema Refinements = Normal Forms

- 1st Normal Form = all tables are flat
- 2nd Normal Form = obsolete
- Boyce Codd Normal Form = will study
- 3rd Normal Form = see book

First Normal Form (1NF)

- A database schema is in First Normal Form if all tables are flat

Student

Name	GPA	Courses
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

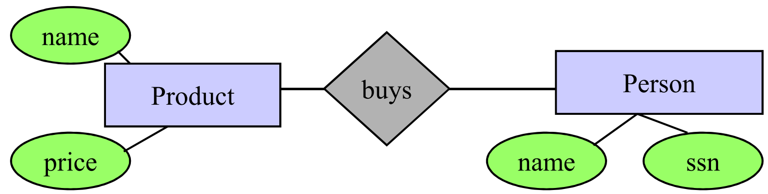
Student	Course
Alice	Math
Carol	Math
Alice	DB
Bob	DB
Alice	OS
Carol	OS

Course

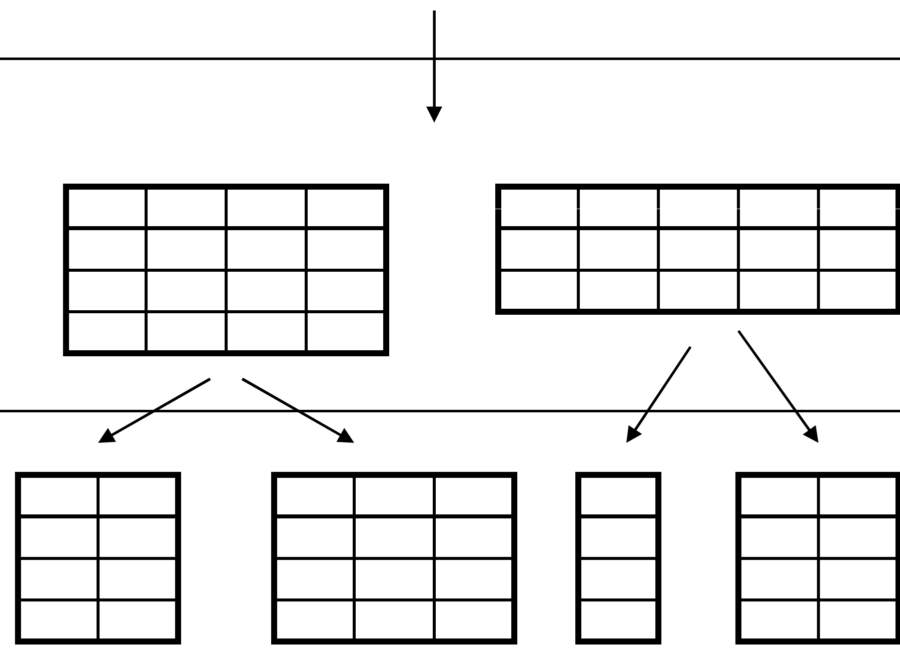
Course
Math
DB
OS

Relational Schema Design

Conceptual Model:



Relational Model:
plus FD's



Normalization:
Eliminates ***anomalies***

Data Anomalies

When a database is poorly designed we get anomalies:

Redundancy: data is repeated

Updated 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	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)

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

Relational Schema Design

Recall set attributes (persons with several phones):

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? (what if Joe had only one phone #)

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)

Main idea:

- Start with some relational schema
- Find out its ***functional dependencies***
  - They come from the application domain knowledge!
- 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:

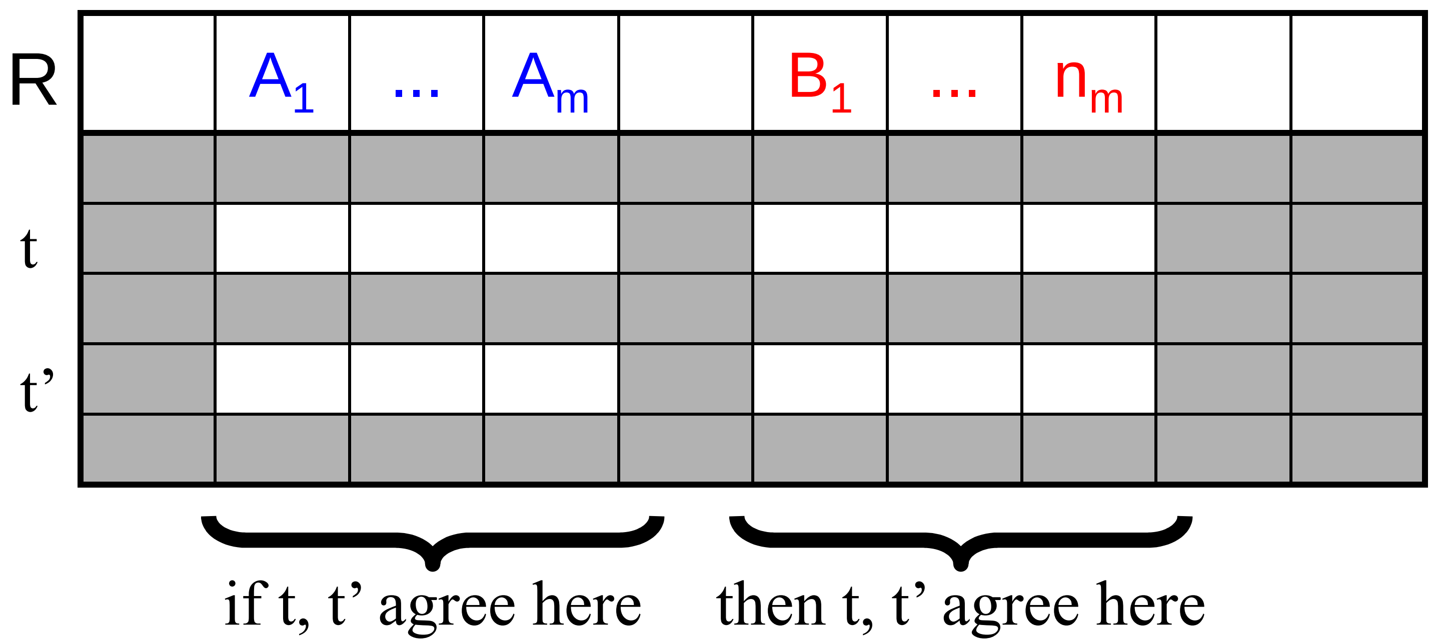
$$A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$$

When Does an FD Hold?

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

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

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



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 $\rightarrow$	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234 $\rightarrow$	Lawyer

But not Phone $\rightarrow$ 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

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 ?

When Does an FD Hold?

- 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:

name → color
category → department
color, category → price

Then this FD also holds:

name, category → price

Why ??

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

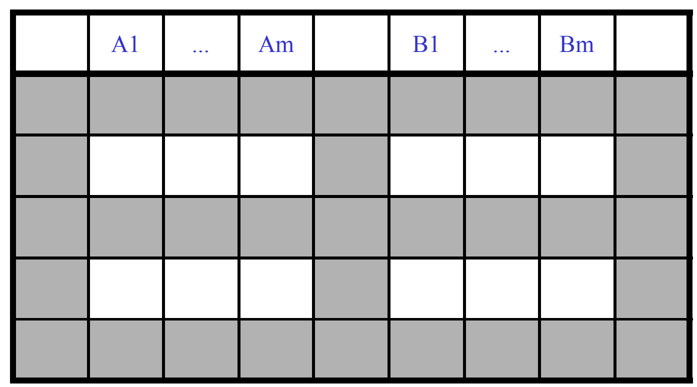
Armstrong's Rules (1/3)

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

Splitting rule and Combing rule

Is equivalent to

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

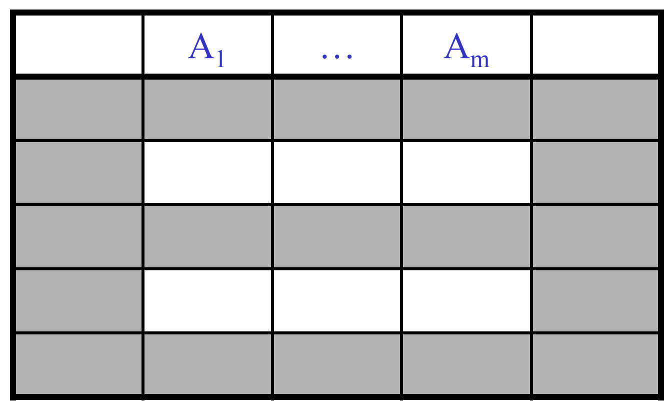


Armstrong's Rules (2/3)

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

Trivial Rule

where i = 1, 2, ..., n



Why ?

Armstrong's Rules (3/3)

Transitive Rule

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

and $B_1, B_2, \ldots, B_m \rightarrow C_1, C_2, \ldots, C_p$

then $A_1, A_2, \ldots, A_n \rightarrow C_1, C_2, \ldots, C_p$

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 → 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)

Answers:

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

Inferred FD	Which Rule did we apply ?
4. name, category → name	Trivial rule
5. name, category → color	Transitivity on 4, 1
6. name, category → category	Trivial rule
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.

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:

name → color
category → department
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:

name → color
category → department
color, category → price

{name, category}$^+$ =
  { name, category, color, 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, }

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	→	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 }

Why Do We Need Closure

- With closure we can find all FD's easily

- To check if $X \rightarrow A$
  - Compute X^+
  - Check if $A \in X^+$

Using Closure to Infer ALL FDs

Example:

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, BC$\rightarrow$D, ABC $\rightarrow$ D, ABD $\rightarrow$ C, ACD $\rightarrow$ B

Another Example

- Enrollment(student, major, course, room, time)

  student → major

  major, course → room

  course → time

  What else can we infer ? [in class, or at home]

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
  - 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)

name, category → price
category → color

What is the key ?

(name, category) + = { name, category, price, color }

Hence (name, category) is a key

Examples of Keys

Enrollment(student, address, course, room, time)

student → address
room, time → course
student, course → room, time

(find keys at home)

Eliminating Anomalies

Main idea:

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

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 → Name, City

What is the key?

{SSN, PhoneNumber}

Hence SSN → Name, City is a “bad” dependency

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

$AB \rightarrow C$ $BC \rightarrow A$	or	$A \rightarrow BC$ $B \rightarrow AC$

what are the keys here ?

Can you design FDs such that there are *three* keys ?

Boyce-Codd Normal Form

A simple condition for removing anomalies from relations:

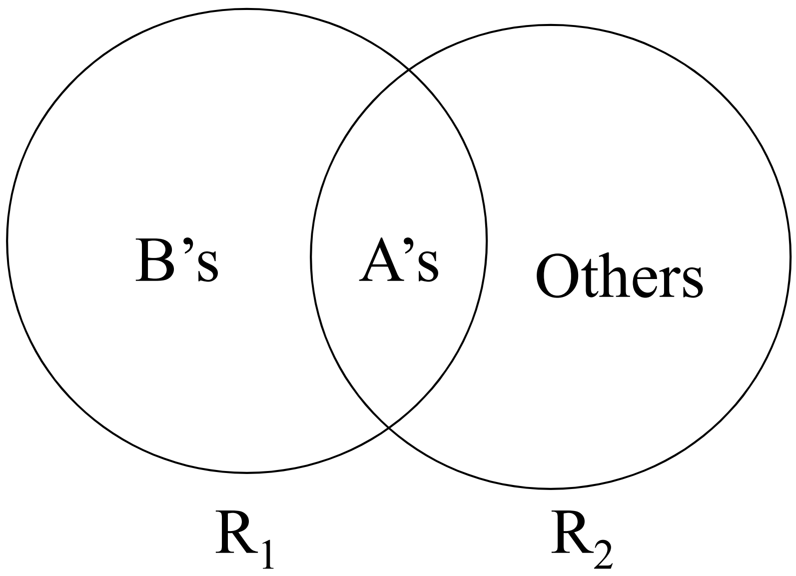
> A relation R is in BCNF if:
>
> If $A_1, ..., A_n \rightarrow B$ is a non-trivial dependency in R,
>
> then $\{A_1, ..., A_n\}$ is a superkey for R

In other words: there are no "bad" FDs

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

BCNF Decomposition Algorithm

repeat
choose $A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$ that violates BCNF
split R into $R_1(A_1, \ldots, A_m, B_1, \ldots, B_n)$ and $R_2(A_1, \ldots, A_m$, [others])
continue with both R_1 and R_2
until no more violations



Is there a 2-attribute relation that is not in BCNF ?

In practice, we have a better algorithm (coming up)

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 → Name, City

What is the key?

{SSN, PhoneNumber}

use SSN → Name, City to split

Example

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 ?

Example Decomposition

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

FD1: SSN → name, age

FD2: age → hairColor

Decompose in BCNF (in class):

Example Decomposition

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

FD1: SSN → name, age

FD2: age → hairColor

Decompose in BCNF (in class): 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)

Or ….

BCNF Decomposition Algorithm

BCNF_Decompose(R)

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

if (not found) **then** "R is in BCNF"

let $Y = X^+ - X$
let $Z =$ [all attributes] $- X^+$
decompose R into $R1(X \cup Y)$ and $R2(X \cup Z)$
continue to decompose recursively R1 and R2

Find X s.t.: $X \neq X^+ \neq$ [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(<u>SSN</u>, name, age, hairColor)

Phone(<u>SSN</u>, phoneNumber)

Iteration 2: P

age+ = age, hairColor

Decompose: People(<u>SSN</u>, name, age)

Hair(<u>age</u>, hairColor)

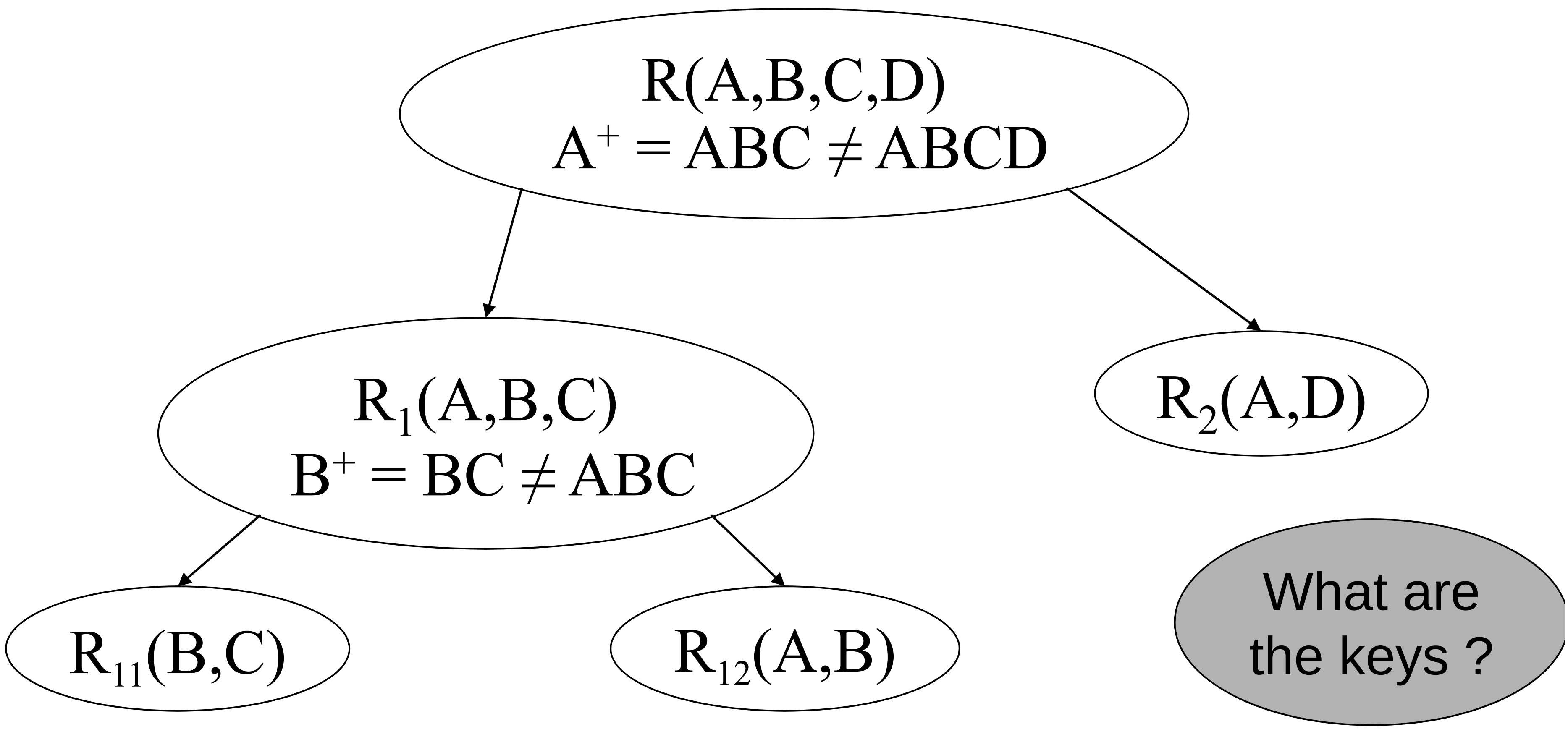
Phone(<u>SSN</u>, phoneNumber)

What are the keys ?

R(A,B,C,D)

Example

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



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

Decompositions in General

$R(A_1, ..., A_n, B_1, ..., B_m, C_1, ..., C_p)$

$R_1(A_1, ..., A_n, B_1, ..., B_m)$

$R_2(A_1, ..., A_n, C_1, ..., C_p)$

R_1 = projection of R on $A_1, ..., A_n, B_1, ..., B_m$
R_2 = projection of R on $A_1, ..., A_n, C_1, ..., C_p$

Theory of Decomposition

- Sometimes it is correct:

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

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

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

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 ?

Optional

- The following four slides are optional
- The content will not be on any exam

- But please take a look because they motivate the need for 3NF

- It's good to know at least why 3NF exists

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

Sometimes cannot decomposed into BCNF without losing ability to check some FDs

BCNF and Dependencies

Unit	Company	Product

FD's: Unit $\rightarrow$ Company; Company, Product $\rightarrow$ Unit

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

BCNF and Dependencies

Unit	Company	Product

FD's: 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 dep. $A_1, A_2, ..., A_n \rightarrow B$ 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