Introduction to Database Systems

Design Theory

(Ch. 3.1, 3.3-4)

Database Design Process

Conceptual Model:

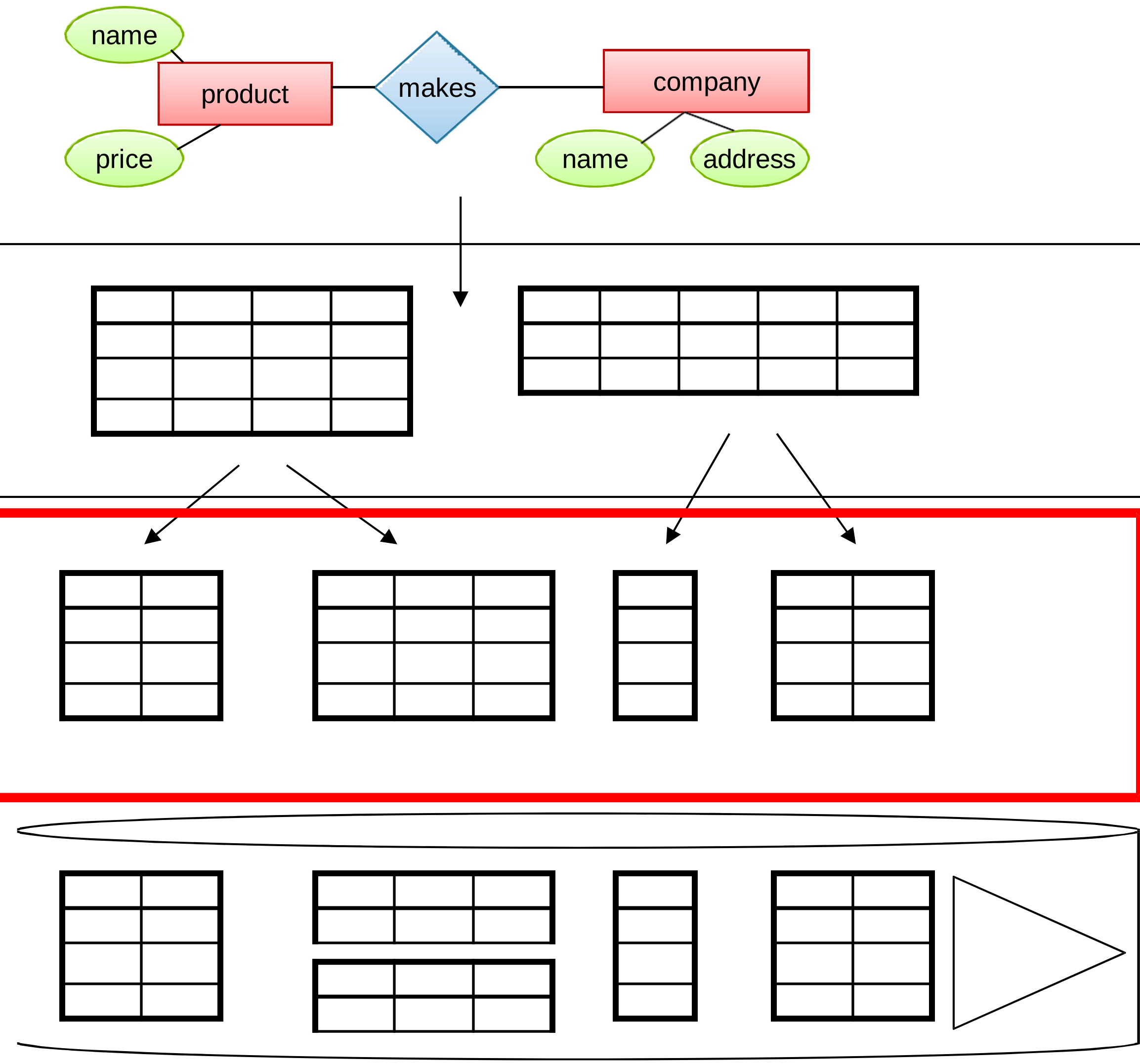
Relational Model:
Tables + constraints
And also functional dep.

Normalization:
Eliminates anomalies

Conceptual Schema

Physical storage details

Physical 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

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

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	<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

$$\boxed{A_1, A_2, \ldots, A_n}$$

then they must also agree on the attributes

$$\boxed{B_1, B_2, \ldots, B_m}$$

Formally:

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

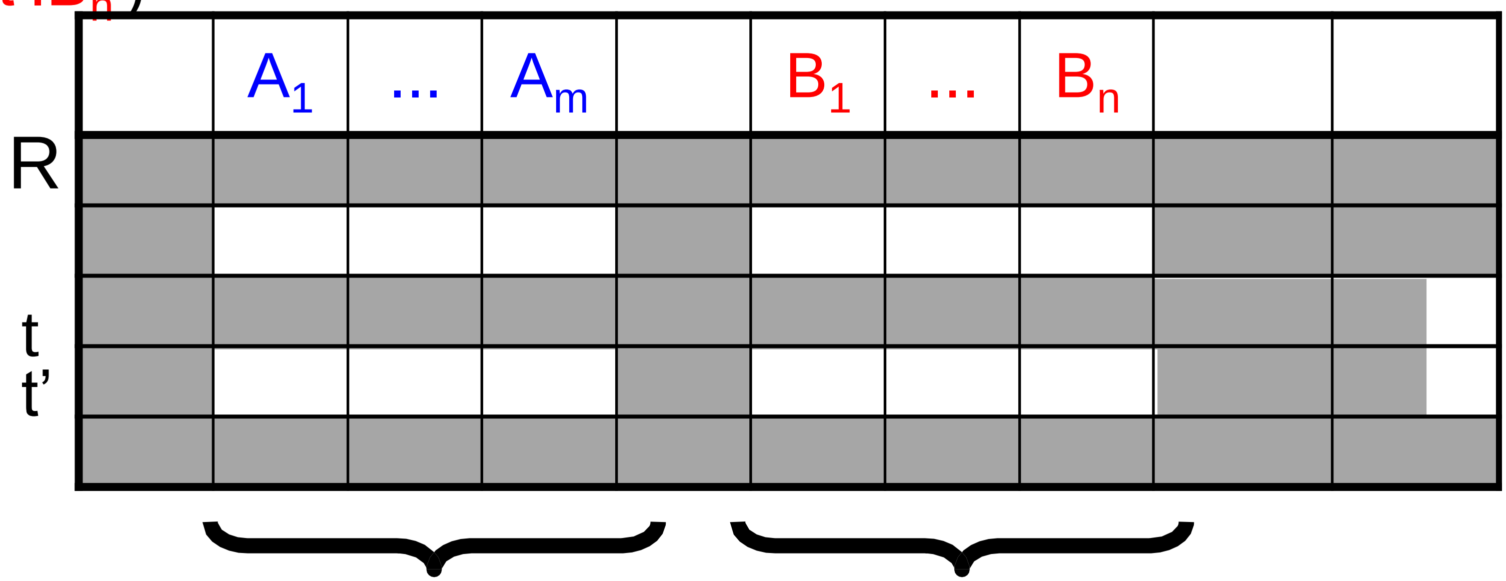
$$\boxed{A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m}$$

Functional Dependencies (FDs)

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

for every pair of tuples t, t' ∈ R,

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



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 $\rightarrow$ Name, Phone, Position

Position $\rightarrow$ Phone

but not Phone $\rightarrow$ 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)

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

Example:

1. name $\rightarrow$ color
2. category $\rightarrow$ department
3. color, category $\rightarrow$ 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: 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 → C
A, D → E
B → D
A, F → 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 }

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:

$A, B \rightarrow C$

$A, D \rightarrow B$

$B \rightarrow D$

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 $A_1, \ldots, A_n$ s.t. for any other attribute B, we have $A_1, \ldots, A_n \rightarrow B$

- A **key** is a *minimal* superkey
  - 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

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

what are the keys here ?

Eliminating Anomalies

Main idea:

- $X \rightarrow A$ 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

Normal Forms

First Normal Form

- Every attribute is singled value attribute

Second Normal Form

- No partial dependency

Third Normal Form

- No transitive dependency

Boyce-Codd Normal Form

- Every non-trivial functional dependency X –> Y, X is a super key

Boyce-Codd Normal Form

There are no "bad" FDs:

Definition. A relation R is in BCNF if:

Whenever $X \rightarrow A$ 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)

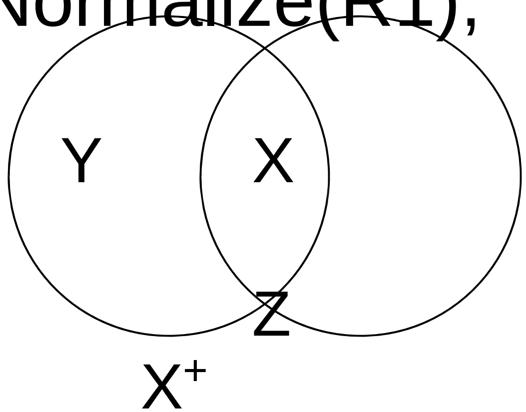
find X s.t.: X ≠ X^+ and X^+ ≠ [all attributes]

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

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

decompose R into R1(X ∪ Y) and R2(X ∪ Z)

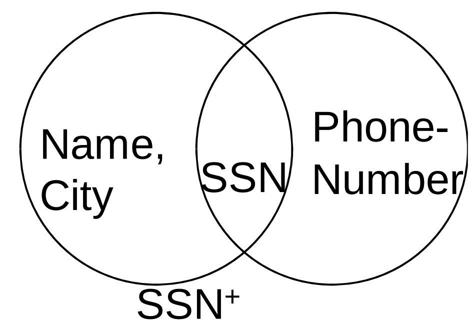
Normalize(R1); Normalize(R2);



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

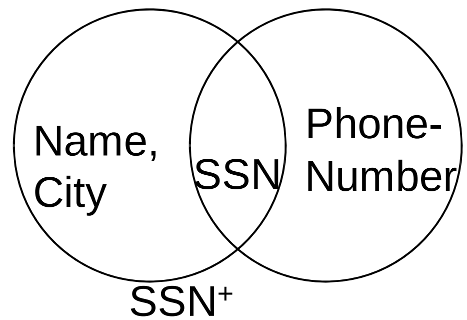
In other words:

SSN^+ = SSN, Name, City and is neither SSN nor All Attributes

Example BCNF Decomposition

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

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

Example BCNF Decomposition

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

SSN $\rightarrow$ name, race

race $\rightarrow$ hairColor

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

Example BCNF Decomposition

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

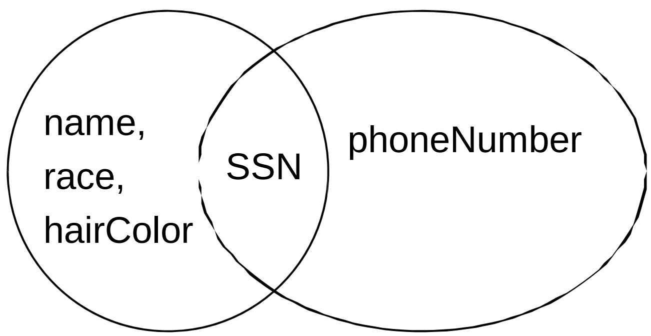
SSN $\rightarrow$ name, race

race $\rightarrow$ hairColor

Iteration 1: Person: SSN^+ = SSN, name, race, hairColor

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

Phone(SSN, phoneNumber)



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

Example BCNF Decomposition

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

SSN → name, race

race → hairColor

What are the keys ?

Iteration 1: Person: SSN^+ = SSN, name, race, hairColor

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

Phone(SSN, phoneNumber)

Iteration 2: P: $race^+$ = race, hairColor

Decompose: People(SSN, name, race)

Hair(race, hairColor)

Phone(SSN, phoneNumber)

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

Example BCNF Decomposition

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

SSN $\rightarrow$ name, race

race $\rightarrow$ hairColor

Note the keys!

Iteration 1: Person: SSN^+ = SSN, name, race, hairColor

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

Phone(SSN, phoneNumber)

Iteration 2: P: $race^+$ = race, hairColor

Decompose: People(SSN, name, race)

Hair(race, 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)

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

Example: BCNF

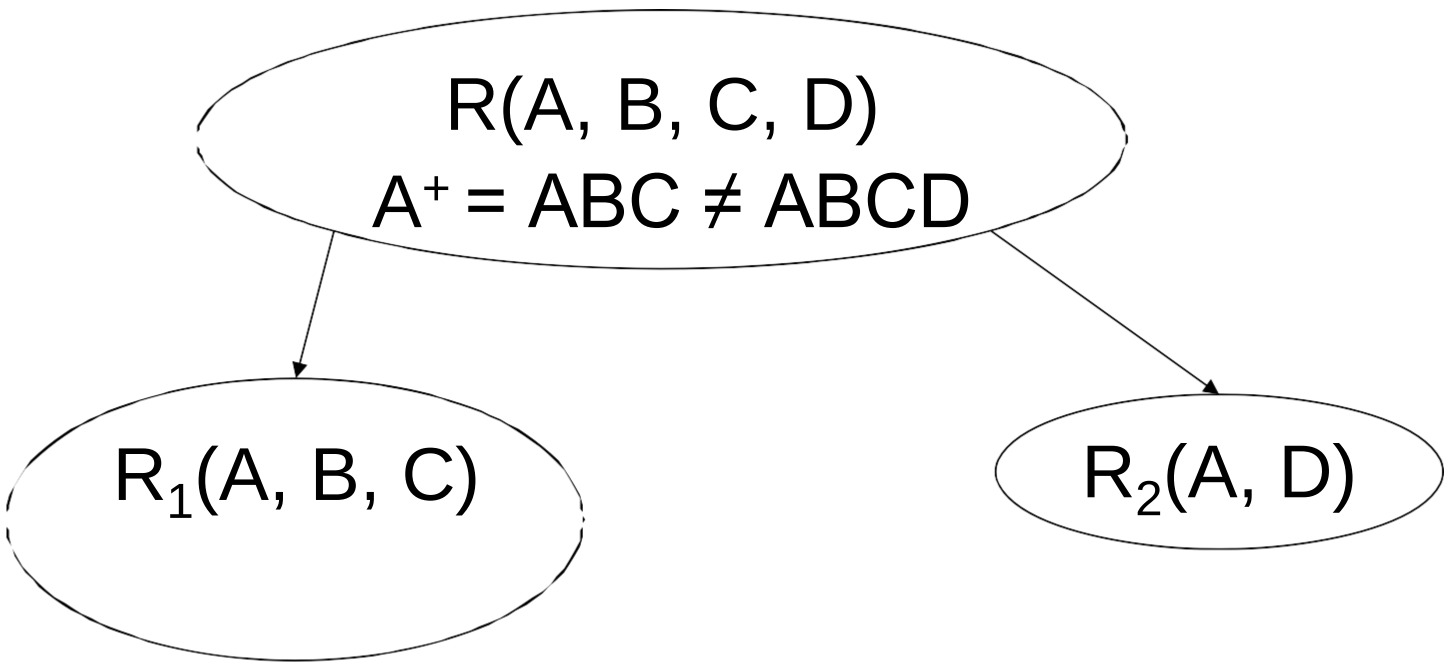
R(A, B, C, D)

$A^+ = ABC \neq ABCD$

R(A, B, C, D)

$A \to B$
$B \to C$

Example: BCNF



R(A, B, C, D)

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

Example: BCNF

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

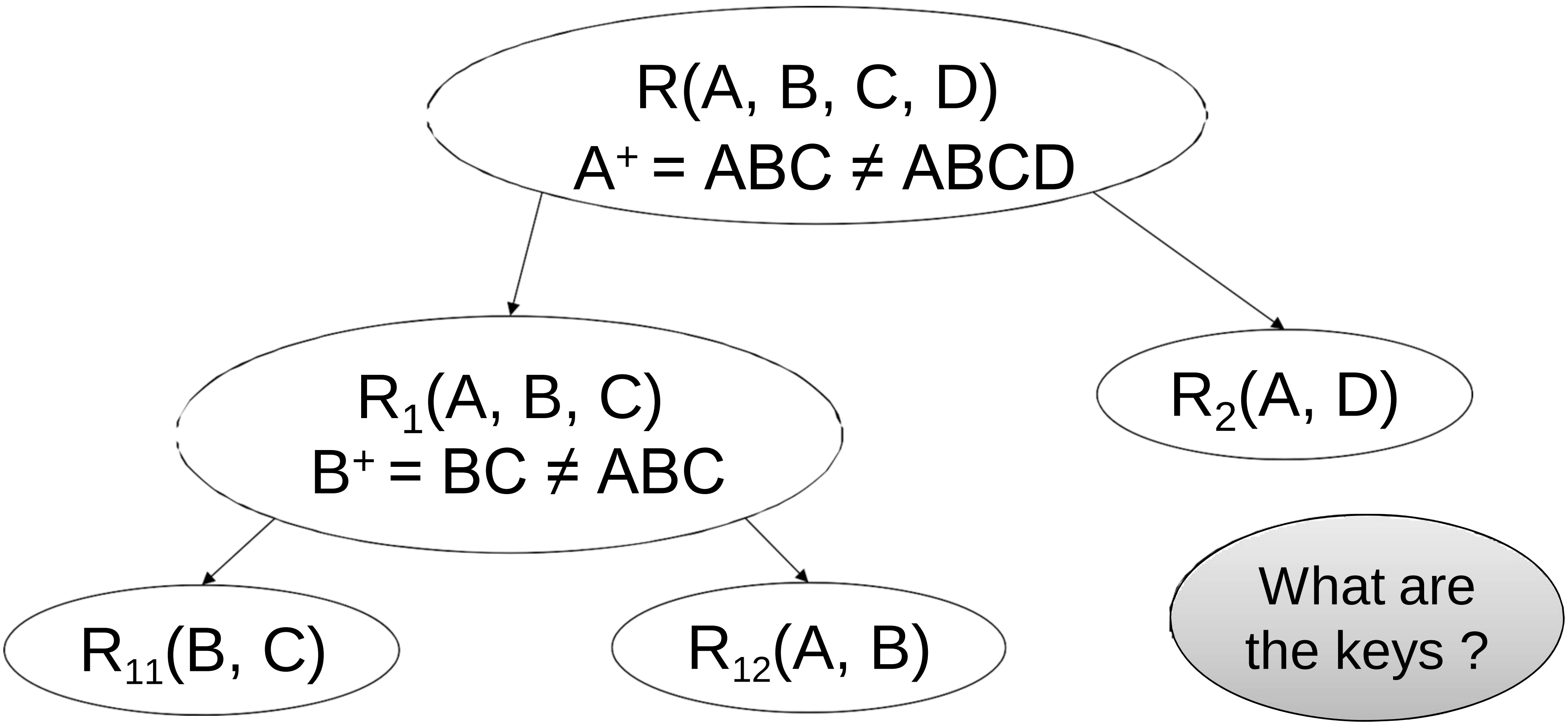
$R_1(A, B, C)$
$B^+ = BC \neq ABC$

$R_2(A, D)$

R(A, B, C, D)

$A \to B$
$B \to C$

Example: BCNF



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

Decompositions in General

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

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

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 $\rightarrow$ price, but not category

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

Schema Refinements = Normal Forms

- 1st Normal Form = all tables are flat (no list values)
- 2nd Normal Form = obsolete
- Boyce Codd Normal Form = no bad FDs
- 3rd Normal Form = see book
  - BCNF is lossless but can cause lose ability to check some FDs without a join (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