# Database Management Systems INFO 210

## Design Theory
## Lecture 10

**Franz Wotawa**
TU Graz, Institut for Software Technologie
Inffeldgasse 16b/2
wotawa@ist.tugraz.at

# Today…

- Last Session:
  - SQL and its relation to Relational Algebra
- Today's Session:
  - Advanced Design concepts

# Outline

- 1st Normal Form = all tables' attributes are atomic
- 2nd Normal Form = will study, later
- Boyce Codd Normal Form = will study
- 3rd Normal Form = will study, later

# First Normal Form (1NF)

- A database schema is in First Normal Form (1NF) if the **domain** of each attribute contains only **atomic** values, and the value of each attribute contains only a **single value** from that **domain.**

# First Normal Form (1NF)

Customer

| Customer ID | First Name | Surname | Telephone Number |
|---|---|---|---|
| 123 | Robert | Ingram | 555-861-2025 |
| 456 | Jane | Wright | 555-403-1659 |
| 789 | Maria | Fernandez | 555-808-9633 |

# Not in First Normal Form (1NF)

Customer

| Customer ID | First Name | Surname | Telephone Number |
|---|---|---|---|
| 123 | Robert | Ingram | 555-861-2025 |
| 456 | Jane | Wright | 555-403-1659<br>555-776-4100 |
| 789 | Maria | Fernandez | 555-808-9633 |

# Now in First Normal Form (1NF)

Customer

| Customer ID | First Name | Surname | Telephone Number |
|---|---|---|---|
| 123 | Robert | Ingram | 555-861-2025 |
| 456 | Jane | Wright | 555-403-1659 |
| 456 | Jane | Wright | 555-776-4100 |
| 789 | Maria | Fernandez | 555-808-9633 |

# Now in First Normal Form (1NF)

| Customer ID | First Name | Surname |
|---|---|---|
| 123 | Robert | Ingram |
| 456 | Jane | Wright |
| 789 | Maria | Fernandez |

| Customer ID | Telephone Number |
|---|---|
| 123 | 555-861-2025 |
| 456 | 555-403-1659 |
| 456 | 555-776-4100 |
| 789 | 555-808-9633 |

# First Normal Form (1NF)

A database schema is in First Normal Form if all tables' attributes contain only **atomic** values.

Student

| Name | GPA | Courses |
|---|---|---|
| Alice | 3.8 | Math<br>DB<br>OS |
| Bob | 3.7 | DB<br>OS |
| Carol | 3.9 | Math<br>OS |

→

May need to add keys

Student

| Name | GPA |
|---|---|
| Alice | 3.8 |
| Bob | 3.7 |
| Carol | 3.9 |

Takes

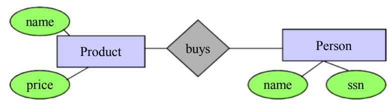
| Student | Course |
|---|---|
| Alice | Math |
| Carol | Math |
| Alice | DB |
| Bob | DB |
| Alice | OS |
| Carol | OS |

Course

| Course |
|---|
| Math |
| DB |
| OS |

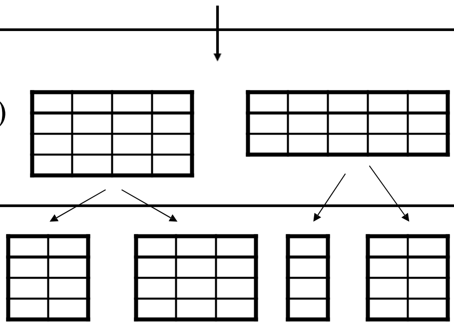
# Relational Schema Design

Conceptual Model:




Relational Model (in 1NF) plus FD's

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 | 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

Anomalies:

- Redundancy = repeated data
- Update anomalies = Fred moves to "Bellevue"
- Deletion anomalies = Joe deletes his phone number: what is his city ?

# 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 are 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***
- 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
- Also used in normalizing the relations

# Functional Dependencies

Definition:

If two tuples agree on the attributes

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

then they must also agree on the attributes

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

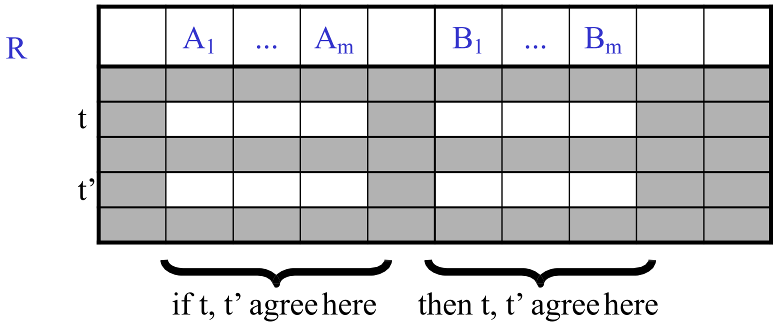
Formally:

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

# When Does a FD Hold

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

$\forall t, t' \in R, (t.A_1=t'.A_1 \wedge \ldots \wedge t.A_m=t'.A_m \Rightarrow t.B_1=t'.B_1 \wedge \ldots \wedge t.B_n=t'.B_n)$




# Example: Movie table

| Title | Year | Length | Genre | StudioName | StarName |
|---|---|---|---|---|---|
| Star Wars | 1977 | 124 | SciFi | Fox | Carrie Fisher |
| Star Wars | 1977 | 124 | SciFi | Fox | Mark Hamill |
| Star Wars | 1977 | 124 | SciFi | Fox | Harrison Ford |
| Gone With the Wind | 1939 | 231 | Drama | MGM | Vivien Leigh |
| Wayne's World | 1992 | 95 | Comedy | Paramount | Dana Carvey |
| Wayne's World | 1992 | 95 | Comedy | Paramount | Mike Meyers |

title, year → length

title, year → genre

title, year → studioName

title, year → length, genre, studioName

# Example: Movie table

| Title | Year | Length | Genre | StudioName | StarName |
|---|---|---|---|---|---|
| Star Wars | 1977 | 124 | SciFi | Fox | Carrie Fisher |
| Star Wars | 1977 | 124 | SciFi | Fox | Mark Hamill |
| Star Wars | 1977 | 124 | SciFi | Fox | Harrison Ford |
| Gone With the Wind | 1939 | 231 | Drama | MGM | Vivien Leigh |
| Wayne's World | 1992 | 95 | Comedy | Paramount | Dana Carvey |
| Wayne's World | 1992 | 95 | Comedy | Paramount | Mike Meyers |

How about?

~~title, year → starName~~

# Examples

An FD holds, or does not hold on an instance:

| EmpID | Name | Phone | Position |
|---|---|---|---|
| E0045 | Smith | 1234 | Clerk |
| E3542 | Mike | 9876 | Salesrep |
| E11 1 | Smith | 9876 | Salesrep |
| E9999 | Mary | 1234 | Lawyer |

EmpID → Name, Phone, Position

but not Name → EmpID

or Name → Phone

# Example

| EmpID | Name | Phone | Position |
|---|---|---|---|
| E0045 | Smith | 1234 | Clerk |
| E3542 | Mike | 9876 ← | Salesrep |
| E11 1 | Smith | 9876 ← | Salesrep |
| E9999 | Mary | 1234 | Lawyer |

Position → Phone

# Example

| EmpID | Name | Phone | Position |
|---|---|---|---|
| E0045 | Smith | 1234 → | Clerk |
| E3542 | Mike | 9876 | Salesrep |
| E11 1 | Smith | 9876 | Salesrep |
| E9999 | Mary | 1234 → | Lawyer |

but not Phone → Position

# Inferring other dependencies from a set of FDs

| name | category | color | department | price |
|---|---|---|---|---|
| Gizmo | Gadget | Green | Toys | 49 |
| Tweaker | Gadget | Black | Toys | 99 |
| Gizmo | Stationary | Green | Office-supp. | 59 |

name → color
category → department
color, category → price

name, category → price

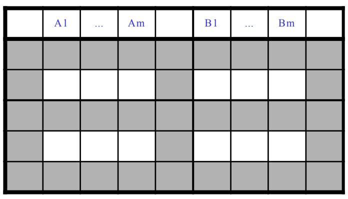
# Armstrong's Rules (1/3)

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

Is equivalent to

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

**Splitting rule and Combing rule**



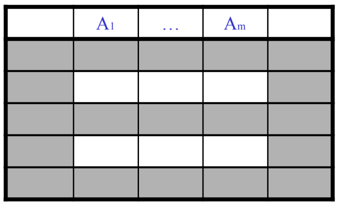
# Armstrong's Rules (2/3)

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

**Trivial Rule**

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

Why ?




# Armstrong's Rules (3/3)

**Transitive Closure Rule**

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

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

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

Why ?


# Inferring other dependencies from a set of FDs

| name | category | color | department | price |
|---|---|---|---|---|
| Gizmo | Gadget | Green | Toys | 49 |
| Tweaker | Gadget | Black | Toys | 99 |
| Gizmo | Stationary | Green | Office-supp. | 59 |

name → color
category → department
color, category → price

name, category → price

# Example (continued)

Start from the following FDs:

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

→ name, category → price

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

Infer the following FDs:

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

# Closure of a set of Attributes

**Given** a set of attributes $A_1, \ldots, A_n$ and a set of FDs

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

# Closures Example

| name | category | color | department | price |
|---|---|---|---|---|
| Gizmo | Gadget | Green | Toys | 49 |
| Tweaker | Gadget | Black | Toys | 99 |
| Gizmo | Stationary | Green | Office-supp. | 59 |

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

## More Examples

In class:

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

| | | |
|---|---|---|
| A, B | $\rightarrow$ | C |
| B, C | $\rightarrow$ | A, D |
| D | $\rightarrow$ | E |
| C, F | $\rightarrow$ | B |

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

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

## More Examples

In class:

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

| | | |
|---|---|---|
| A, B | $\rightarrow$ | C |
| B, C | $\rightarrow$ | A, D |
| D | $\rightarrow$ | E |
| C, F | $\rightarrow$ | B |

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

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

# Application of Closures

- Does a new FD logically follows from a set of FD
- Inferring All FDs that logically follows from a set of FD
- Finding all keys and superkeys

# (1) Does a new FD logically follows from a set of FDs?

$R(A_1, A_2, \ldots\ldots, A_m)$

$\boxed{\text{Set of FDs}} \Rightarrow \boxed{\text{A new FD}}$

- To check if $X \rightarrow A$ (new FD)
  - Using given set of FDs Compute $X^+$ i.e., $\{\text{left side}\}^+$
  - Check if $A \in X^+$

# Example

R(A,B,C,D)

| |
|---|
| A, B $\rightarrow$ C |
| B, C $\rightarrow$ D |
| C, D $\rightarrow$ A |
| A, D $\rightarrow$ B |

$\Rightarrow$ $\boxed{A, D \rightarrow C}$

1. Compute $\{A,D\}^+$

2. If it contains C then the new FD logically follows

# Example

R(A,B,C,D)

| |
|---|
| A, B $\rightarrow$ C |
| B, C $\rightarrow$ D |
| C, D $\rightarrow$ A |
| A, D $\rightarrow$ B |

$\Rightarrow$ $\boxed{B, D \rightarrow A}$

1. Compute $\{B, D\}^+ = \{B, D\}$

2. A is not a member of the set, hence it doesn't logically follow

# Using Closure to Infer ALL FDs

Example:

A, B → C
A, D → B
B → D

Step 1: Compute $X^+$, for every $X \subseteq \{A,B,C,D\}$:

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 → Y, s.t. $Y \subseteq X^+$ and $X \cap Y = \emptyset$

AB → CD, AD→BC, BC →D, ABC → D, ABD → C, ACD → 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]

# Application of Closure: Finding 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

# How many Superkeys?

Suppose R is a relation with attributes $A_1, A_2, \ldots A_n$. As a function of n, tell how many superkeys R has, if:

a) The only key is $A_1$

b) The only keys are $A_1$ and $A_2$

# How many Superkeys?

Suppose R is a relation with attributes $A_1, A_2, \ldots A_n$.
As a function of n, tell how many superkeys R has, if:

a) The only key is $A_1 \rightarrow 2^{n-1}$

b) The only keys are $A_1$ and $A_2 \rightarrow 3.2^{n-2}$

# Computing (Super)Keys

- Compute $X^+$ for all sets X
- If $X^+$ = all attributes, then X is a key
- List only the minimal X's

# Example 1

R(A,B,C,D)

$A, B \rightarrow C$
$B, C \rightarrow D$
$C, D \rightarrow A$
$A, D \rightarrow B$

What are all the keys?

What are all the superkeys that are not keys?

# Example

R(A,B,C,D)

$A, B \rightarrow C$
$B, C \rightarrow D$
$C, D \rightarrow A$
$A, D \rightarrow B$

Keys:
AB , AD, BC, CD

$A^+ = A$, $B^+ = B$, $C^+ = C$, $D^+ = D$
$AB^+ = ABCD$, $AC^+ = AC$, $AD^+ = ABCD$,
$BC^+ = ABCD$, $BD^+ = BD$, $CD^+ = ABCD$
$ABC^+ = ABD^+ = ACD^+ = ABCD$ (no need to compute– why ?)
$BCD^+ = ABCD$, $ABCD^+ = ABCD$

# Example

R(A,B,C,D)

A, B → C
B, C → D
C, D → A
A, D → B

Superkeys that are not Keys:
ABC , ABD, BCD, ACD,ABCD

$A^+ = A$, $B^+ = B$, $C^+ = C$, $D^+ = D$
$AB^+$ =ABCD, $AC^+$=AC,$AD^+$=ABCD,
$BC^+$=ABCD, $BD^+$=BD, $CD^+$=ABCD
$ABC^+ = ABD^+ = ACD^+$ = ABCD (no need to compute– why ?)
$BCD^+$ = ABCD, $ABCD^+$ =ABCD

# Example 2

A, B → C
A, D → B
B → D

Keys:
AB ,AD

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

# Example 2

A, B → C
A, D → B
B → D

Superkeys that are not Keys:
ABC , ABD, ACD,ABCD

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

# 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)

# Key or Keys ?

Can we have more than one key ?

Given R(A,B,C) define FD's s.t. there are two keys

| AB→C<br>BC→A | or | A→BC<br>B→AC |
|---|---|---|

what are the keys here ?

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

# Eliminating Anomalies

Main idea:

- X → A is OK if X is a (super)key
- X → 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

# 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, that is the left side of every nontrivial FD must be a super key.

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

# All two-attribute relations are in BCNF

R(A,B)

Case 1: Suppose there is no dependency

Nothing is being violated, so fine

Case 2: Suppose A → B is the only dependency

$A^+ = AB$, so left side of A → B is a key

Case 3: Suppose B → A is the only dependency

$B^+ = AB$, so left side of B → A is a key

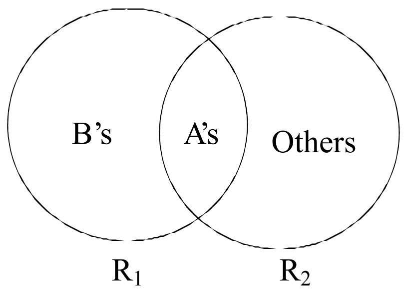
Case 4: Suppose A→ B and B → A are two dependencies

$A^+ = AB$, $B^+ = AB$,

so left side of B → A is a key and left side of A→B is also a key

# 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 $R_2(A_1, \ldots, A_m, [\text{others}])$
  continue with both $R_1$ and $R_2$
**until** no more violations




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

R1(SSN, Name, City)
R2(SSN, PhoneNumber)

# 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 ?

# 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**

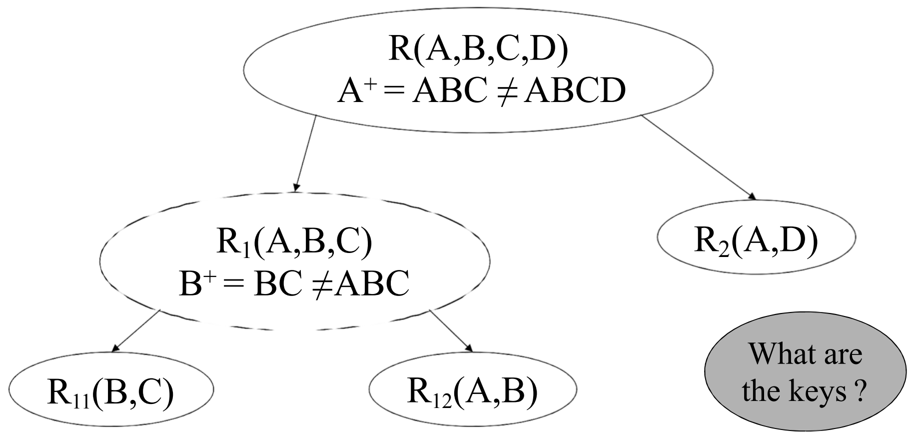
$Y = X^+ - X$

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

R(A,B,C,D) Find X s.t.: $X \neq X^+ \neq$ [all attributes]

# Example

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



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

# Example2: 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)

Iteration 2: P

$SSN^+$ = SSN, name, age, hairColor

age+ = age, hairColor

Decompose: People(SSN, name, age)

Hair(age, hairColor)

Phone(SSN, phoneNumber)

What are the keys ?

# 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 $A_1, ..., A_n, C_1, ..., C_p$

# Next Class

Conceptual Modeling