



→ | 알 범위 | ↓

## Chapter 7: Normalization

Database System Concepts, 7<sup>th</sup> Ed.

©Silberschatz, Korth and Sudarshan

See [www.db-book.com](http://www.db-book.com) for conditions on re-use



## Outline

- Features of Good Relational Design
- Functional Dependencies
- Decomposition Using Functional Dependencies
- Normal Forms
- Functional Dependency Theory
- Algorithms for Decomposition using Functional Dependencies
- Decomposition Using Multivalued Dependencies
- More Normal Form
- Atomic Domains and First Normal Form
- Database-Design Process
- Modeling Temporal Data



in relation "is 1D e name dependency

## Features of Good Relational Designs

- Suppose we combine *instructor* and *department* into *in\_dep*, which represents the natural join on the relations *instructor* and *department*

ID	name	salary	dept_name	building	budget
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

classroom(building, room\_number, capacity)  
 department(dept\_name, building, budget)  
 course(course\_id, title, dept\_name, credits)  
 instructor(ID, name, dept\_name, salary)  
 section(course\_id, sec\_id, semester, year, building, room\_number, time\_slot\_id)  
 teaches(ID, course\_id, sec\_id, semester, year)  
 student(ID, name, dept\_name, tot\_cred)  
 takes(ID, course\_id, sec\_id, semester, year, grade)  
 advisor(s\_ID, i\_ID)  
 time\_slot(time\_slot\_id, day, start\_time, end\_time)  
 prereq(course\_id, prereq\_id)

Figure 7.1 Database schema for the university example.

by *instructor*

- There is repetition of information
- Need to use null values (if we add a new department with no instructors)
  - 重复 of 1D.
  - ↓
  - primary key ID ← 1D + 6 = 7D.



## Decomposition

提高性能 ...

导致丢失信息 ...

- The only way to avoid the repetition-of-information problem in the *in\_dep* schema is to decompose it into two schemas – *instructor* and *department* schemas.
- Not all decompositions are good. Suppose we decompose

into

*employee1* (ID, name)

*employee2* (name, street, city, salary)

if there are two employee with the same name, it's problem.

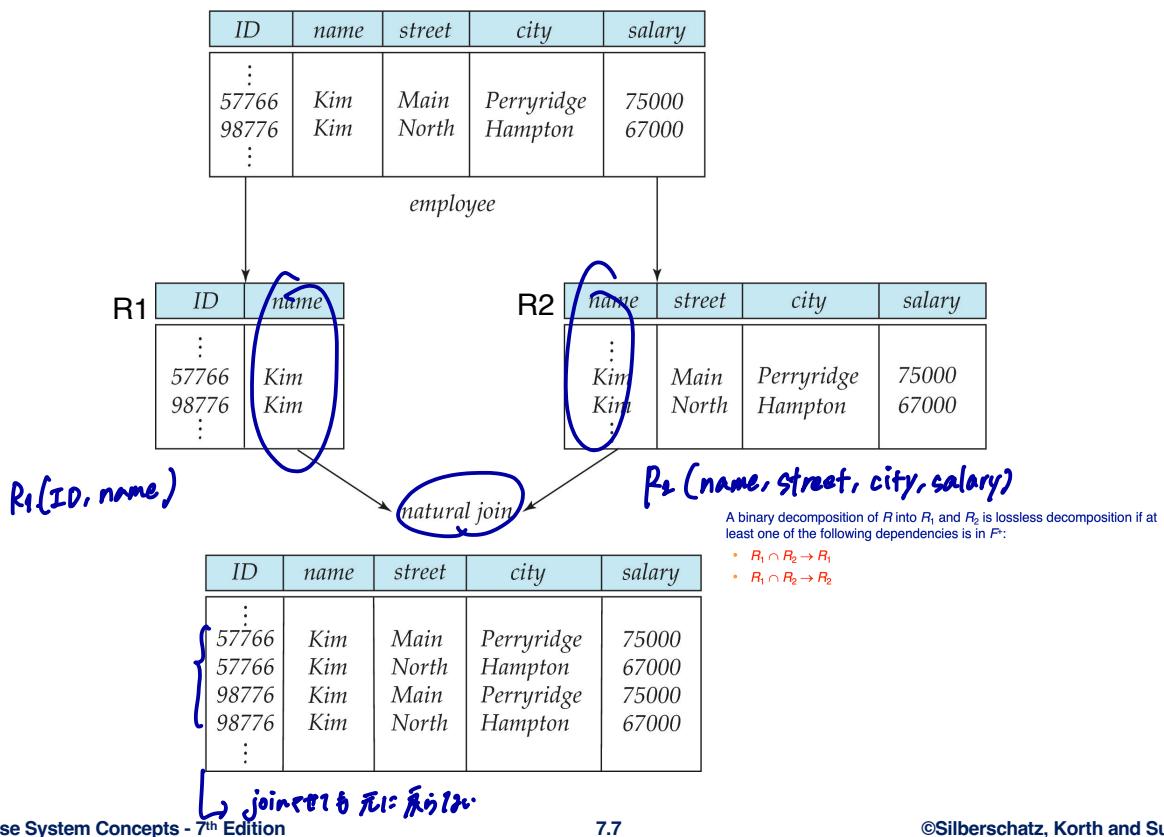
The problem arises when we have two employees with the same name

- The next slide shows how we lose information -- we cannot reconstruct the original *employee* relation -- and so, this is a **lossy decomposition**.



# A Lossy Decomposition

$F^+ = \{ID \rightarrow name, street, city, salary\}$



# Lossless Decomposition

- Let  $R$  be a relation schema and let  $R_1$  and  $R_2$  form a decomposition of  $R$ . That is  $R = R_1 \cup R_2$
  - We say that the decomposition is a **lossless decomposition** if there is no loss of information by replacing  $R$  with the two relation schemas  $R_1 \cup R_2$ . *即如果通过替换R为两个关系模式R1和R2，信息没有丢失。*
  - Formally,
- $\Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) = r$
- And, conversely a decomposition is lossy if
- $r \subset \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) = r$



## Example of Lossless Decomposition

- Decomposition of  $R = (A, B, C)$

$$R_1 = (A, B) \quad R_2 = (B, C)$$

A	B	C
$\alpha$	1	A
$\beta$	2	B

r

A	B
$\alpha$	1
$\beta$	2

$\Pi_{A,B}(r)$

B	C
1	A
2	B

$\Pi_{B,C}(r)$

$\Pi_A(r) \bowtie \Pi_B(r)$	A	B	C
	$\alpha$	1	A
	$\beta$	2	B



## Normalization Theory

- Decide whether a particular relation  $R$  is in “good” form.
- In the case that a relation  $R$  is not in “good” form, decompose it into set of relations  $\{R_1, R_2, \dots, R_n\}$  such that
  - Each relation is in good form
  - The decomposition is a lossless decomposition
- Our theory is based on:
  - Functional dependencies
  - Multivalued dependencies

good form = 3NF · 4NF



# Functional Dependencies

- There are usually a variety of constraints (rules) on the data in the real world.
- For example, some of the constraints that are expected to hold in a university database are:
  - Students and instructors are uniquely identified by their **ID**.  
 $ID \rightarrow \{name, address\}$   
 $\rightarrow \text{Last name} \text{ and } \text{attribute} \{name\} = \text{First name} \{ID\}$
  - Each student and instructor has only one name.
  - Each instructor and student is (primarily) associated with only one department.
  - Each department has only one value for its budget, and only one associated building.



## Functional Dependencies (Cont.)

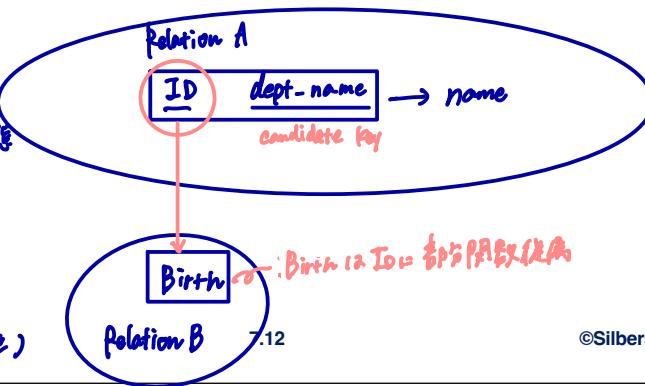
关系的合法实例是那些满足所有现实世界约束的实例。

- An instance of a relation that satisfies all such real-world constraints is called a **legal instance** of the relation;
- A legal instance of a database is one where all the relation instances are legal instances
- Constraints on the set of legal relations.  
 $(\text{legal }) \{ID\} \rightarrow \{name\}$
- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes.
- A functional dependency is a generalization of the notion of a key.

$X \rightarrow Y$   
若  $X$  的值唯一确定  $Y$  的值，则  $X$  是  $Y$  的函数。  
 $X \rightarrow ID$   $ID \rightarrow name$

$ex: X \rightarrow ID$   
 $Y \rightarrow Name$

$X \rightarrow Y$ ,  $Y \rightarrow Z$   
candidate key  $\{X\} = \{Y\} \rightarrow \{Z\}$





## Functional Dependencies Definition

- Let  $R$  be a relation schema  
 $\alpha \subseteq R$  and  $\beta \subseteq R$
- The **functional dependency**  
 $\alpha \rightarrow \beta$

**holds on  $R$**  if and only if for any legal relations  $r(R)$ , whenever any two tuples  $t_1$  and  $t_2$  of  $r$  agree on the attributes  $\alpha$ , they also agree on the attributes  $\beta$ . That is,

$$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$

- Example: Consider  $r(A,B)$  with the following instance of  $r$ .

$t_1[1] = t_2[1]$   
 $\nexists t_1[4] \neq t_2[5]$

A	B
1	4
1	5
3	7

*for 2nd tuple check.*

- On this instance,  $B \rightarrow A$  hold;  $A \rightarrow B$  does **NOT** hold,

$$A = 1 \text{ 且 } 4 \neq 5?$$



## Closure of a Set of Functional Dependencies

- 推导的函数依赖**
- Given a set  $F$  of functional dependencies, there are certain other functional dependencies that are logically implied by  $F$ .
    - If  $A \rightarrow B$  and  $B \rightarrow C$ , then we can infer that  $A \rightarrow C$
    - etc.
  - The set of **all** functional dependencies logically implied by  $F$  is the **closure** of  $F$ .
  - We denote the *closure* of  $F$  by  $F^+$ .



## Keys and Functional Dependencies

subset

- $K$  is a superkey for relation schema  $R$  if and only if  $K \rightarrow R$
- $K$  is a candidate key for  $R$  if and only if
  - $K \rightarrow R$ , and
  - for no  $\alpha \subset K$ ,  $\alpha \rightarrow R$
- Functional dependencies allow us to express constraints that cannot be expressed using superkeys. Consider the schema:

*in\_dep (ID, name, salary, dept\_name, building, budget).*

We expect these functional dependencies to hold:

$dept\_name \rightarrow building$

$ID \rightarrow building$

*building*

but would not expect the following to hold:

$dept\_name \rightarrow salary$     *superkey*  $= ID \cup dept\_name$



## Use of Functional Dependencies

- We use functional dependencies to:
  - To test relations to see if they are legal under a given set of functional dependencies.
    - If a relation  $r$  is legal under a set  $F$  of functional dependencies, we say that  $r$  **satisfies**  $F$ .
  - To specify constraints on the set of legal relations
    - We say that  $F$  **holds on**  $R$  if all legal relations on  $R$  satisfy the set of functional dependencies  $F$ .
- Note: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances.
  - For example, a specific instance of *instructor* may, by chance, satisfy  $name \rightarrow ID$ .



## Trivial Functional Dependencies

自明的属性.

- Some types of functional dependencies (e.g.,  $A \rightarrow A$ ,  $AB \rightarrow A$ ) are said to be **trivial** if they are satisfied by all relations
- Example:
  - $\underline{\text{name}} \rightarrow \underline{\text{name}}$
  - $\underline{ID}, \underline{\text{name}} \rightarrow \underline{ID}$
- In general,  $\alpha \rightarrow \beta$  is trivial if  $\beta \subseteq \alpha$



函数依赖的性质是自明的.  
或  
主键是所有属性的函数依赖.



## Lossless Decomposition

自明的.

如果  $R_1$  和  $R_2$  都是空集

- We can use functional dependencies to show when certain decompositions are lossless.
- For the case of  $R = (R_1, R_2)$ , we require that for all possible relations  $r$  on schema  $R$

$$r = \prod_{R_1}(r) \bowtie \prod_{R_2}(r)$$

- A binary decomposition of  $R$  into  $R_1$  and  $R_2$  is **lossless decomposition** if at least one of the following dependencies is in  $F^+$ :

$R_1 \cap R_2 \rightarrow R_1$  \* + 例 \*  
or  
 $R_1 \cap R_2 \rightarrow R_2$  \* + 例 \*  $R_1 \rightarrow R_2$  \* + 例 \*  $R_2 \rightarrow R_1$  \* + 例 \*

- The above functional dependencies are a sufficient condition for lossless join decomposition; the dependencies are a necessary condition only if all constraints are functional dependencies

- If  $((R_1 \cap R_2 \rightarrow R_1) \vee (R_1 \cap R_2 \rightarrow R_2))$ , then a binary decomposition of  $R$  into  $R_1$  and  $R_2$  is lossless decomposition (True)
  - If a binary decomposition of  $R$  into  $R_1$  and  $R_2$  is lossless decomposition, then  $((R_1 \cap R_2 \rightarrow R_1) \vee (R_1 \cap R_2 \rightarrow R_2))$  (???)



## Example

- $R = (A, B, C)$   
 $F = \{A \rightarrow B, B \rightarrow C\}$ .
- $R_1 = (A, B), R_2 = (B, C)$ 
  - Lossless decomposition:  
 $R_1 \cap R_2 = \{B\}$  and  $B \rightarrow BC$
- $R_1 = (A, B), R_2 = (A, C) \quad R_1 \cap R_2 \rightarrow R_2$ 
  - Lossless decomposition:  
 $R_1 \cap R_2 = \{A\}$  and  $A \rightarrow AB$
- Note:  
 $R_1 \cap R_2 \rightarrow R_1$ 
  - $B \rightarrow BC$   
is a shorthand notation for
  - $B \rightarrow \{B, C\}$



?



## [functional dependency] Dependency Preservation functional dependency & 1NF 2NF 3NF?

- Testing functional dependency constraints each time the database is updated can be costly
- It is useful to design the database in a way that constraints can be tested efficiently.
- If testing a functional dependency can be done by considering just one relation, then the cost of testing this constraint is low
  - When decomposing a relation it is possible that it is no longer possible to do the testing without having to perform a Cartesian Product.
- A decomposition that makes it computationally hard to enforce functional dependency is said to be NOT dependency preserving.
  - If a relation has a primary key.
    - If a relation has a primary key & foreign key &  $\text{INTO} = \text{FROM}$ .



## Dependency Preservation Example

- Consider a schema:  $\begin{array}{c} \text{dept\_advisor} \\ \text{s\_ID, i\_ID, dept\_name} \\ \text{PK} \end{array}$
- With function dependencies:
$$i\_ID \rightarrow dept\_name$$
$$s\_ID, dept\_name \rightarrow i\_ID$$
- In the above design we are forced to repeat the department name once for each time an instructor participates in a dept\_advisor relationship.
- To fix this, we need to decompose *dept\_advisor*
- Any decomposition will not include all the attributes in
$$s\_ID, dept\_name \rightarrow i\_ID$$
- Thus, the composition NOT be dependency preserving
- $R_1 = \{s\_ID, i\_ID\}, R_2 = \{i\_ID, dept\_name\}$

A binary decomposition of  $R$  into  $R_1$  and  $R_2$  is lossless decomposition if at least one of the following dependencies is in  $F^+$ :

- $R_1 \cap R_2 \rightarrow R_1$
- $R_1 \cap R_2 \rightarrow R_2$

4/17



Boyce-Codd Normal Form

- A relation schema  $R$  is in BCNF with respect to a set  $F$  of functional dependencies if for all functional dependencies in  $F^+$  of the form

$$\alpha \rightarrow \beta$$

where  $\alpha \subseteq R$  and  $\beta \subseteq R$ , at least one of the following holds:

- $\alpha \rightarrow \beta$  is trivial (i.e.,  $\beta \subseteq \alpha$ )
- $\alpha$  is a superkey for  $R$



## Boyce-Codd Normal Form (Cont.)

- Example schema that is **not** in BCNF:  
 $in\_dep (ID, name, salary, dept\_name, building, budget)$   
because :
  - $dept\_name \rightarrow building, budget$ 
    - holds on  $in\_dep$
    - but
  - $dept\_name$  is not a superkey
- When decompose  $in\_dep$  into  $instructor$  and  $department$ 
  - $instructor$  is in BCNF
    - $instructor (ID, name, dept\_name, salary) F^+ = \{ID \rightarrow name, dept\_name, salary\}$
  - $department$  is in BCNF
    - $dept (dept\_name, building, budget) F^+ = \{dept\_name \rightarrow building, budget\}$



## Decomposing a Schema into BCNF

- Let  $R$  be a schema  $R$  that is not in BCNF. Let  $\alpha \rightarrow \beta$  be the FD that causes a violation of BCNF. *BCNF違反を引き起こすFDを分解する方法。*
- We decompose  $R$  into:
  - $(\alpha \cup \beta)$
  - $(R - (\beta - \alpha))$
- In our example of  $in\_dep$ ,
  - $in\_dep (ID, name, salary, dept\_name, building, budget)$
  - $\alpha = dept\_name$
  - $\beta = building, budget$and  $in\_dep$  is replaced by
  - $(\alpha \cup \beta) = (dept\_name, building, budget)$
  - $(R - (\beta - \alpha)) = (ID, name, dept\_name, salary)$

*BCNF違反 because dept-name is not superkey  
 $dept\_name \rightarrow building, budget$*



## Example

- $R = (A, B, C)$   
 $F = \{A \rightarrow B, B \rightarrow C\}$   $F^+ = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$
- $R_1 = (A, B), R_2 = (B, C)$   $\begin{array}{l} R_1: A \rightarrow B \\ R_2: B \rightarrow C \end{array} \} \text{ cartesian product} \Rightarrow R(A, B, C)$   
 • Lossless-join decomposition:  $\Rightarrow A \rightarrow C$  *in schema & R(A, B, C)*
- $R_1 = (A, B), R_2 = (A, C)$  *in R<sub>1</sub> B → C is not superkey*  
 • Lossless-join decomposition:  
 $R_1: A \rightarrow B$   $R_1 \cap R_2 = \{A\}$  and  $A \rightarrow AB$   
 $R_2: A \rightarrow C$    
 • Not dependency preserving  
 (cannot check  $B \rightarrow C$  without computing  $R_1 \bowtie R_2$ )

$AB \quad AC \quad F = \{A \rightarrow B, B \rightarrow C\}$



## BCNF and Dependency Preservation

- It is not always possible to achieve both BCNF and dependency preservation
- Consider a schema:  
 $dept\_advisor(s\_ID, i\_ID, dept\_name)$
- With function dependencies:  $R1(i\_ID, dept\_name)$ ,  $R2(s\_ID, i\_ID)$   
 $i\_ID \rightarrow dept\_name \Rightarrow i\_ID \text{ is not superkey} \Rightarrow \text{not BCNF}$   
 $s\_ID, dept\_name \rightarrow i\_ID \text{ not BCNF}$
- $dept\_advisor$  is not in BCNF
  - $i\_ID$  is not a superkey.
- Any decomposition of  $dept\_advisor$  will not include all the attributes in  
 $s\_ID, dept\_name \rightarrow i\_ID$  *not dependency preservation*
- Thus, the decomposition is NOT be dependency preserving
  - Motivation for 3<sup>rd</sup> Normal Form



preserving BCNF  
3NF

## Third Normal Form

- A relation schema  $R$  is in **third normal form (3NF)** if for all:

$$\alpha \rightarrow \beta \text{ in } F^+$$

at least one of the following holds:

- $\alpha \rightarrow \beta$  is trivial (i.e.,  $\beta \in \alpha$ )
- $\alpha$  is a superkey for  $R$
- Each attribute  $A$  in  $\beta - \alpha$  is contained in a candidate key for  $R$ .

(NOTE: each attribute may be in a different candidate key)

- If a relation is in BCNF, it is in 3NF (since in BCNF one of the first two conditions above must hold).
- Third condition is a minimal relaxation of BCNF to ensure dependency preservation (will see why later).
- BCNF  $\rightarrow$  3NF (O) 3NF  $\rightarrow$  BCNF (X)



## 3NF Example

A relation schema  $R$  is in **third normal form (3NF)** if for all:

$$\alpha \rightarrow \beta \text{ in } F^+$$

at least one of the following holds:

- $\alpha \rightarrow \beta$  is trivial (i.e.,  $\beta \in \alpha$ )
- $\alpha$  is a superkey for  $R$
- Each attribute  $A$  in  $\beta - \alpha$  is contained in a candidate key for  $R$ .

(NOTE: each attribute may be in a different candidate key)

- Consider a schema:  
 $dept\_advisor(s\_ID, i\_ID, dept\_name)$
- With function dependencies:
  - $i\_ID \rightarrow dept\_name$  nontrivial,  $i\_ID$  is not superkey ,  $\beta - \alpha = \{dept\_name\}$  is candidate key
  - $s\_ID, dept\_name \rightarrow i\_ID$   $\{s\_ID, dept\_name\}$  is superkey.
- Two candidate keys =  $\{s\_ID, dept\_name\}, \{s\_ID, i\_ID\}$
- We have seen before that  $dept\_advisor$  is not in BCNF
- $dept\_advisor$ , however, is in 3NF
  - $s\_ID, dept\_name$  is a superkey
  - $i\_ID \rightarrow dept\_name$  and  $i\_ID$  is NOT a superkey, but:
    - $\{dept\_name\} - \{i\_ID\} = \{dept\_name\}$  and,
    - $dept\_name$  is contained in a candidate key  $\{s\_ID, dept\_name\}$



## Redundancy in 3NF

- Consider the schema R below, which is in 3NF

- $R = (J, K, L)$
- $F = \{JK \rightarrow L, L \rightarrow K\}$
- And an instance table:

3NF は満たす  
重複等の問題が発生する

	J	L	K
$j_1$	$l_1$	$k_1$	
$j_2$	$l_1$	$k_1$	
$j_3$	$l_1$	$k_1$	
null	$l_2$	$k_2$	

- What is wrong with the table?

- Repetition of information
- Need to use null values (e.g., to represent the relationship  $l_2, k_2$  where there is no corresponding value for  $J$ )

$J \rightarrow K$  は満たさない。  
 $J \rightarrow K$  が NULL の場合は lossless.  $\rightarrow$  Join すると null tuple が生じる。  
でも  $J$  が NULL の場合は  $K$  が null となる場合がある。



## Comparison of BCNF and 3NF

- Advantages to 3NF over BCNF. It is always possible to obtain a 3NF design without sacrificing losslessness or dependency preservation.
- Disadvantages to 3NF.
  - We may have to use null values to represent some of the possible meaningful relationships among data items.
  - There is the problem of repetition of information.

X/9.

7.31



## Goals of Normalization

- Let  $R$  be a relation scheme with a set  $F$  of functional dependencies.
- Decide whether a relation scheme  $R$  is in “good” form.
- In the case that a relation scheme  $R$  is not in “good” form, need to decompose it into a set of relation scheme  $\{R_1, R_2, \dots, R_n\}$  such that:
  - Each relation scheme is in good form
  - The decomposition is a lossless decomposition *good form can call this*
  - Preferably, the decomposition should be dependency preserving.



## How good is BCNF?

- There are database schemas in BCNF that do not seem to be sufficiently normalized
- Consider a relation
 

*inst\_info (ID, child\_name, phone) primary key*

  - where an instructor may have more than one phone and can have multiple children
  - Instance of *inst\_info*

<i>ID</i>	<i>child_name</i>	<i>phone</i>
99999	David	512-555-1234
99999	David	512-555-4321
99999	William	512-555-1234
99999	William	512-555-4321

*ID = phone + first 2 child\_name = BCNF*



## How good is BCNF? (Cont.)

- There are no non-trivial functional dependencies and therefore the relation is in BCNF
- Insertion anomalies – i.e., if we add a phone 981-992-3443 to 99999, we need to add two tuples

(99999, David, 981-992-3443) ) *重复插入*.  
(99999, William, 981-992-3443)

*fv.  
p1 (ID, phone) 互独立.*



## Higher Normal Forms

- It is better to decompose *inst\_info* into:
  - *inst\_child*:

ID	child_name
99999	David
99999	William

- *inst\_phone*:

ID	phone
99999	512-555-1234
99999	512-555-4321

- This suggests the need for higher normal forms, such as Fourth Normal Form (4NF), which we shall see later



## Functional-Dependency Theory Roadmap

- ① We now consider the formal theory that tells us which functional dependencies are implied logically by a given set of functional dependencies.
- ② We then develop algorithms to generate lossless decompositions into BCNF and 3NF
- ③ We then develop algorithms to test if a decomposition is dependency-preserving



## Closure of a Set of Functional Dependencies

- Given a set  $F$  of functional dependencies, there are certain other functional dependencies that are logically implied by  $F$ .
  - If  $A \rightarrow B$  and  $B \rightarrow C$ , then we can infer that  $A \rightarrow C$
  - etc.
- The set of all functional dependencies logically implied by  $F$  is the **closure** of  $F$ .
- We denote the *closure* of  $F$  by  $F^+$ .



# Closure of a Set of Functional Dependencies

- We can compute  $F^+$ , the closure of  $F$ , by repeatedly applying **Armstrong's Axioms**:
  - **Reflexive rule:** if  $\beta \subseteq \alpha$ , then  $\alpha \rightarrow \beta$
  - **Augmentation rule:** if  $\alpha \rightarrow \beta$ , then  $\gamma\alpha \rightarrow \gamma\beta$
  - **Transitivity rule:** if  $\alpha \rightarrow \beta$ , and  $\beta \rightarrow \gamma$ , then  $\alpha \rightarrow \gamma$
- These rules are
  - **Sound** -- generate only functional dependencies that actually hold, and
  - **Complete** -- generate all functional dependencies that hold.



## Example of $F^+$

### Transitivity

- $R = (A, B, C, G, H, I)$   
 $F = \{ A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H \}$   
 $AG \rightarrow CG \rightarrow I$
- **Reflexive rule:** if  $\beta \subseteq \alpha$ , then  $\alpha \rightarrow \beta$
- **Augmentation rule:** if  $\alpha \rightarrow \beta$ , then  $\gamma\alpha \rightarrow \gamma\beta$
- **Transitivity rule:** if  $\alpha \rightarrow \beta$ , and  $\beta \rightarrow \gamma$ , then  $\alpha \rightarrow \gamma$
- Some members of  $F^+$ 
  - $A \rightarrow H$ 
    - by transitivity from  $A \rightarrow B$  and  $B \rightarrow H$
  - $AG \rightarrow I$ 
    - by augmenting  $A \rightarrow C$  with  $G$ , to get  $AG \rightarrow CG$  and then transitivity with  $CG \rightarrow I$
  - $CG \rightarrow HI$ 
    - by augmenting  $CG \rightarrow I$  to infer  $CG \rightarrow CGI$ , and augmenting of  $CG \rightarrow H$  to infer  $CGI \rightarrow HI$ , and then transitivity ( $CG \rightarrow CGI$  and  $CGI \rightarrow HI$ , then  $CG \rightarrow HI$ )



## Closure of Functional Dependencies (Cont.)

Functional dependencies

- Additional rules:

- Union rule:** If  $\alpha \rightarrow \beta$  holds and  $\alpha \rightarrow \gamma$  holds, then  $\alpha \rightarrow \beta\gamma$  holds.

▪  $\alpha\gamma \rightarrow \beta\gamma$  by Augmentation

▪  $\alpha \rightarrow \alpha\gamma$  by Augmentation  $\alpha \rightarrow \alpha\alpha \rightarrow \beta\alpha$

▪  $\alpha \rightarrow \beta\gamma$  by transitivity

- Decomposition rule:** If  $\alpha \rightarrow \beta\gamma$  holds, then  $\alpha \rightarrow \beta$  holds and  $\alpha \rightarrow \gamma$  holds.

- Pseudotransitivity rule:** If  $\alpha \rightarrow \beta$  holds and  $\gamma\beta \rightarrow \delta$  holds, then  $\alpha\gamma \rightarrow \delta$  holds.

▪  $\alpha \rightarrow \beta \rightarrow \alpha\gamma \rightarrow \beta\gamma$  by Augmentation

▪  $\alpha\gamma \rightarrow \beta\gamma$  and  $\gamma\beta \rightarrow \delta \rightarrow \alpha\gamma \rightarrow \delta$  by transitivity

- The above rules can be inferred from Armstrong's axioms.

• **Reflexive rule:** if  $\beta \subseteq \alpha$ , then  $\alpha \rightarrow \beta$

• **Augmentation rule:** if  $\alpha \rightarrow \beta$ , then  $\gamma\alpha \rightarrow \gamma\beta$

• **Transitivity rule:** if  $\alpha \rightarrow \beta$ , and  $\beta \rightarrow \gamma$ , then  $\alpha \rightarrow \gamma$



## Procedure for Computing $F^+$

- To compute the closure of a set of functional dependencies  $F$ :

$$F^+ = F$$

repeat

for each functional dependency  $f$  in  $F^+$

apply reflexivity and augmentation rules on  $f$

add the resulting functional dependencies to  $F^+$

for each pair of functional dependencies  $f_1$  and  $f_2$  in  $F^+$

if  $f_1$  and  $f_2$  can be combined using transitivity

then add the resulting functional dependency to  $F^+$

until  $F^+$  does not change any further

止り、次回計算上にこまく。

止り、次回計算上にこまく。

- $2^n \times 2^n = 2^{2n}$  possible functional dependencies where  $n$  is # of attributes in  $R$

- NOTE:** We shall see an alternative procedure for this task later



# Closure of Attribute Sets

- Given a set of attributes  $\alpha$ , define the ***closure*** of  $\alpha$  under  $F$  (denoted by  $\alpha^+$ ) as the set of attributes that are functionally determined by  $\alpha$  under  $F$
- Algorithm to compute  $\alpha^+$ , the closure of  $\alpha$  under  $F$

```

result := α;
while (changes to result) do
    for each β → γ in F do
        begin
            if β ⊆ result then result := result ∪ γ
        end
    
```

is no  
↓



## Example of Attribute Set Closure

(3)

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$
- $(AG)^+$

$(A^+)$   
 $\text{result} = ABC$        $A \rightarrow B, A \rightarrow C$   
 $\text{result} = ABCH$        $B \rightarrow H$  and  $B \subseteq \text{result}$   
 $(A^+) \not\subseteq R$        $A$  is not superkey  
 $(G^+)$   
 $\text{result} = G$        $G$  is not superkey  
 $\Rightarrow AG$  is candidate key

- Is  $AG$  a candidate key?

- Is  $AG$  a super key?

- Does  $AG \rightarrow R$ ? == Is  $(AG)^+ \supseteq R$

- Is any subset of  $AG$  a superkey?

- Does  $A \rightarrow R$ ? == Is  $(A)^+ \supseteq R$

- Does  $G \rightarrow R$ ? == Is  $(G)^+ \supseteq R$

- In general: check for each subset of size  $n-1$

$B \rightarrow H$   
not.

$(A^+) = (ABC)$   $\Rightarrow (A^+)$  is not superkey

$(G^+) = G$   $\Rightarrow G$  is not superkey

$\Rightarrow (AG)^+$  is candidate key



# Uses of Attribute Closure

There are several uses of the attribute closure algorithm:

- **Testing for superkey:**
  - To test if  $\alpha$  is a superkey, we compute  $\alpha^+$ , and check if  $\alpha^+$  contains all attributes of  $R$ .
- **Testing functional dependencies**
  - To check if a functional dependency  $\alpha \rightarrow \beta$  holds (or, in other words, is in  $F^+$ ), just check if  $\beta \subseteq \alpha^+$ . *부분집합을 hold*
  - That is, we compute  $\alpha^+$  by using attribute closure, and then check if it contains  $\beta$ .
  - Is a simple and cheap test, and very useful
- **Provides an alternative way of Computing closure of  $F$** 
  - For each  $\gamma \subseteq R$ , we find the closure  $\gamma^+$ , and for each  $S \subseteq \gamma^+$ , we output a functional dependency  $\gamma \rightarrow S$ .



# Canonical Cover

- Suppose that we have a set of functional dependencies  $F$  on a relation schema. Whenever a user performs an update on the relation, the database system must ensure that the update does not violate any functional dependencies, that is, all the functional dependencies in  $F$  are satisfied in the new database state. *update 가 블루레이드 되지 않도록 함.*
- If an update violates any functional dependencies in the set  $F$ , the system must roll back the update.
- We can reduce the effort spent in checking for violations by testing a simplified set of functional dependencies that has the same closure as the given set.
- This simplified set is termed the **canonical cover**.
- To define canonical cover we must first define **extraneous attributes**.
  - An attribute of a functional dependency in  $F$  is **extraneous** if we can remove it without changing  $F^+$

How?

$$\begin{array}{l} a \rightarrow \beta \\ ta \rightarrow tb \\ \hline \end{array}$$

*ta는 b에 필요하지 않다.*



## Extraneous Attributes

불필요한.

- Removing an attribute from the **left side** of a functional dependency could make it a **stronger constraint**. *강한 예약된 dependency*
  - For example, if we have  $AB \rightarrow C$  and remove B, we get the possibly stronger result  $A \rightarrow C$ . It may be **stronger** because  $A \rightarrow C$  logically implies  $AB \rightarrow C$ , but  $AB \rightarrow C$  does not, on its own, logically imply  $A \rightarrow C$ .
- But, depending on what our set F of functional dependencies happens to be, we may be able to remove B from  $AB \rightarrow C$  safely.
  - For example, suppose that
  - $F = \{AB \rightarrow C, A \rightarrow D, D \rightarrow C\}$
  - Then we can show that F logically implies  $A \rightarrow C$ , making **B extraneous** in  $AB \rightarrow C$ .

0 5/3 .

5.8.

## Extraneous Attributes (Cont.)

- Removing an attribute from the **right side** of a functional dependency could make it a **weaker constraint**. *약한 예약된 constraint*
  - For example, if we have  $AB \rightarrow CD$  and remove C, we get the possibly weaker result  $AB \rightarrow D$ . It may be **weaker** because using just  $AB \rightarrow D$ , we can no longer infer  $AB \rightarrow C$ .
- But, depending on what our set F of functional dependencies happens to be, we may be able to remove C from  $AB \rightarrow CD$  safely.
  - For example, suppose that  $F = \{AB \rightarrow CD, A \rightarrow C\}$ . *22장 4.189 쪽*  $AB \rightarrow CD$  is **sufficient** to infer  $AB \rightarrow C$ .  $\therefore AB \rightarrow D$
  - Then we can show that even after replacing  $AB \rightarrow CD$  by  $AB \rightarrow D$ , we can still infer  $AB \rightarrow C$  and thus  $AB \rightarrow CD$ .



# Extraneous Attributes

- An attribute of a functional dependency in  $F$  is **extraneous** if we can remove it without changing  $F^+$
- Consider a set  $F$  of functional dependencies and the functional dependency  $\alpha \rightarrow \beta$  in  $F$ . 같은 결과가 되는 설명
  - **Remove from the left side:** Attribute  $A$  is **extraneous** in  $\alpha$  if
    - $A \in \alpha$  and
    - $F$  logically implies  $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$ .
  - **Remove from the right side:** Attribute  $A$  is **extraneous** in  $\beta$  if
    - $A \in \beta$  and
    - The set of functional dependencies  $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$  logically implies  $F$ .
- **Note:** Implication in the opposite direction is trivial in each of the cases above, since **a “stronger” functional dependency always implies a weaker one**.
  - **Remove from the left side:**  $F - \{\alpha \rightarrow \beta\} \cup \{(\alpha - A) \rightarrow \beta\}$  logically implies  $F$ .
  - **Remove from the right side:**  $F$  logically implies  $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ .



# Testing if an Attribute is Extraneous

- Let  $R$  be a relation schema and let  $F$  be a set of functional dependencies that hold on  $R$ . Consider an attribute in the functional dependency  $\alpha \rightarrow \beta$ .
- To test if attribute  $A \in \beta$  is extraneous in  $\beta$ 
  - Consider the set:  
$$F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\},$$
  - check that  $\alpha^+$  contains  $A$  under  $F'$ ; if it does,  $A$  is extraneous in  $\beta$
- To test if attribute  $A \in \alpha$  is extraneous in  $\alpha$ 
  - Let  $\gamma = \alpha - \{A\}$ . Check if  $\gamma \rightarrow \beta$  can be inferred from  $F$ .
    - Compute  $\gamma^+$  using the dependencies in  $F$
    - If  $\gamma^+$  includes all attributes in  $\beta$  then,  $A$  is extraneous in  $\alpha$



## Examples of Extraneous Attributes

- Let  $F = \{AB \rightarrow CD, A \rightarrow E, E \rightarrow C\}$

To check if  $C$  is extraneous in  $AB \rightarrow CD$ , we:

- Compute the attribute closure of  $AB$  under  $F'$  =  $\{AB \rightarrow D, A \rightarrow E, E \rightarrow C\}$
- The closure is  $ABCDE$ , which includes  $CD$
- This implies that  $C$  is extraneous
- $AB^+ = \{A, B, C, D, E\}$
- $F' = \{AB \rightarrow D, A \rightarrow E, E \rightarrow C\}$

$$AB \rightarrow CD \xrightarrow{\text{closure}} AB \rightarrow D$$

$C$  is extraneous

$$(AB^+) = ABCDE \Rightarrow C \text{ is extraneous}$$

$D \in B^+$

$$\begin{aligned} AB \rightarrow CD \\ \text{closure: } & (B^+) \text{ is } \{E\} \\ & (B^+) = E \\ \text{closure: } & (A^+) \text{ is } \{C\} \\ & (A^+) = AC \text{ not extraneous} \end{aligned}$$

$$\begin{aligned} AB \rightarrow CD &\Rightarrow AB \rightarrow C \\ \text{closure: } & F' = \{AB \rightarrow C, A \rightarrow E, E \rightarrow C\} \\ (AB^+) = ABCE &\Rightarrow D \in B^+ \Rightarrow D \text{ is extraneous} \end{aligned}$$



## Canonical Cover

$$F \geq F_c$$

$F_c$  is a set of dependencies such that  
functional dependency rule

A **canonical cover** for  $F$  is a set of dependencies  $F_c$  such that

- $F$  logically implies all dependencies in  $F_c$ , and
- $F_c$  logically implies all dependencies in  $F$ , and
- No functional dependency in  $F_c$  contains an extraneous attribute, and
- Each left side of functional dependency in  $F_c$  is unique. That is, there are no two dependencies in  $F_c$

?

- $\alpha_1 \rightarrow \beta_1$  and  $\alpha_2 \rightarrow \beta_2$  such that

$$\alpha_1 = \alpha_2$$

$F_c$  is simple.



# Canonical Cover

- To compute a canonical cover for  $F$ :

$$F_c = F$$

**repeat**

*not unique*

*union Ets*

- 1 Use the union rule to replace any dependencies in  $F$  of the form

$$\alpha_1 \rightarrow \beta_1 \text{ and } \alpha_1 \rightarrow \beta_2 \text{ with } \alpha_1 \rightarrow \beta_1 \beta_2$$

- 2 Find a functional dependency  $\alpha \rightarrow \beta$  in  $F_c$  with an **extraneous attribute** either in  $\alpha$  or in  $\beta$

/\* Note: test for extraneous attributes done using  $F_c$ , not  $F^*$ /

If an extraneous attribute is found, delete it from  $\alpha \rightarrow \beta$

**until** ( $F_c$  not change)

- Note: Union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied



## Example 1: Computing a Canonical Cover

- $R = (A, B, C)$   
 $F = \{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$

$$B \rightarrow C$$

- 1 Combine  $A \rightarrow BC$  and  $A \rightarrow B$  into  $A \rightarrow BC$

- Set is now  $\{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$

- 2 A is extraneous in  $AB \rightarrow C$

$$\hookrightarrow A \text{ is extraneous in } (B^+) = BC \quad A \text{ is FKE. ?}$$

- Check if the result of deleting A from  $AB \rightarrow C$  is implied by the other dependencies

- Yes: in fact,  $B \rightarrow C$  is already present!

- Set is now  $\{A \rightarrow BC, B \rightarrow C\}$

- 3 C is extraneous in  $A \rightarrow BC$

- Check if  $A \rightarrow C$  is logically implied by  $A \rightarrow B$  and the other dependencies

- Yes: using transitivity on  $A \rightarrow B$  and  $B \rightarrow C$ .

- Can use attribute closure of A in more complex cases

- The canonical cover is:  
 $A \rightarrow B$   
 $B \rightarrow C$

$$\therefore F_c = \{A \rightarrow B, B \rightarrow C\}$$



## Example 2: Computing a Canonical Cover

- $R = \{A, B, C, D\}$
- $F = \{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C, AC \rightarrow D\}$ 
  - A → BC and A → B are combined into A → BC
- $F = \{A \rightarrow BC, B \rightarrow C, AB \rightarrow C, AC \rightarrow D\}$ 
  - For  $A \rightarrow BC$ , C is an extraneous attribute
  - $F = \{A \rightarrow B, B \rightarrow C, AB \rightarrow C, AC \rightarrow D\}$ 
    - For  $AB \rightarrow C$ , C is an extraneous attribute and thus  $AB \rightarrow C$  is deleted
    - $F = \{A \rightarrow B, B \rightarrow C, AC \rightarrow D\}$   $(AB)^+ = ABCD \neq AB \rightarrow C$   $\therefore$   $C$  is extraneous
    - For  $AC \rightarrow D$ , C is an extraneous attribute
      - This is because  $A^+ = \{ABCD\}$  from  $F = \{A \rightarrow B, B \rightarrow C, AC \rightarrow D\}$
  - $F_c = \{A \rightarrow B, B \rightarrow C, A \rightarrow D\}$
  - $F_c = \{A \rightarrow BD, B \rightarrow C\}$ .

左側의 예제는  $F$ 의 캐노니컬 커버를 찾는 예제입니다.  
우선  $A \rightarrow BC$ 와  $A \rightarrow B$ 는 합쳐지며, 그 결과로  $A \rightarrow BC$ 가 생깁니다.  
 $AB \rightarrow C$ 는  $AB$ 의 전치집합인  $ABCD$ 와 같지 않기 때문에 제거됩니다.  
 $AC \rightarrow D$ 는  $C$ 가 전치집합인  $ABCD$ 와 같지 않기 때문에 제거됩니다.



의문

## Dependency Preservation

Dependence relation  $\subseteq F^+ \times F^+$

- Let  $F_i$  be the set of dependencies  $F^+$  that include only attributes in  $R_i$ .
  - A decomposition is **dependency preserving**, if
$$(F_1 \cup F_2 \cup \dots \cup F_n)^+ = F^+$$
- Using the above definition, testing for dependency preservation take exponential time.
- Note that if a decomposition is NOT dependency preserving then checking updates for violation of functional dependencies may require computing joins, which is expensive.



## Dependency Preservation (Cont.)

*F<sub>i</sub> is dependency R<sub>i</sub> + check*

- Let  $F$  be the set of dependencies on schema  $R$  and let  $R_1, R_2, \dots, R_n$  be a decomposition of  $R$ .
- The restriction of  $F$  to  $R_i$  is the set  $F_i$  of all functional dependencies in  $F^+$  that include **only** attributes of  $R_i$ .
- Since all functional dependencies in a restriction involve attributes of only one relation schema, it is possible to test such a dependency for satisfaction by checking only one relation.
- Note that the definition of restriction uses all dependencies in  $F^+$ , not just those in  $F$ .
- The set of restrictions  $F_1, F_2, \dots, F_n$  is the set of functional dependencies that can be checked efficiently.



## Testing for Dependency Preservation

*attribute closure*

- To check if a dependency  $\alpha \rightarrow \beta$  is preserved in a decomposition of  $R$  into  $R_1, R_2, \dots, R_n$ , we apply the following test (with **attribute closure done with respect to  $F$** )
  - $result = \alpha$
  - repeat**
  - for each**  $R_i$  in the decomposition
  - $t = (result \cap R_i)^+ \cap R_i$
  - $result = result \cup t$- until** ( $result$  does not change)
  - If  $result$  contains all attributes in  $\beta$ , then the functional dependency  $\alpha \rightarrow \beta$  is preserved.
- We apply the test on all dependencies in  $F$  to check if a decomposition is dependency preserving
- This procedure takes polynomial time, instead of the exponential time required to compute  $F^+$  and  $(F_1 \cup F_2 \cup \dots \cup F_n)^+$



①  $A \rightarrow B$  &  $B \rightarrow C$  R is not BCNF

## Example $R_1 =$

- $R = (A, B, C)$   
 $F = \{A \rightarrow B, B \rightarrow C\}$   $A \rightarrow C$   
Key = {A} primary
- $R$  is not in BCNF [ $\rightarrow$  because  $B$  is not superkey]

② Decomposition  $R_1 = (A, B)$ ,  $R_2 = (B, C)$

- $R_1$  and  $R_2$  in BCNF
- Lossless-join decomposition
- Dependency preserving

$F = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$   $\xrightarrow{\text{R}_1, R_2 \text{ dependency preserve}}$

$\xrightarrow{\text{Check if } A \rightarrow C \text{ is preserved by using the previous algorithm}}$

$$\textcircled{1} (B^+) = BC$$

$$\textcircled{2} t = (B^+) \cap R_2$$

$$= BC \cap R_2 = BC$$

$$\textcircled{3} \text{ result} = \text{result} \cup t$$

$$= AB \cup BC = ABC$$

$\gg R_1, R_2$

① result A

②  $(A^+) = ABC$

③  $t = (A^+) \cap R_1$

$$= (ABC) \cap (A, B)$$

$$= (A, B)$$

④ result = AB



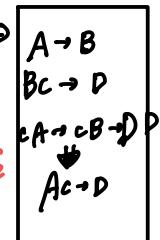
## Algorithm for Decomposition Using Functional Dependencies



# Testing for BCNF

違反

- To check if a non-trivial dependency  $\alpha \rightarrow \beta$  causes a violation of BCNF
  1. compute  $\alpha^+$  (the attribute closure of  $\alpha$ ), and
  2. verify that it includes all attributes of  $R$ , that is, it is a superkey of  $R$ .
- **Simplified test:** To check if a relation schema  $R$  is in BCNF, it suffices to check only the dependencies in the given set  $F$  for violation of BCNF, rather than checking all dependencies in  $F^+$ .
  - If none of the dependencies in  $F$  causes a violation of BCNF, then none of the dependencies in  $F^+$  will cause a violation of BCNF either.
- However, simplified test using only  $F$  is incorrect when testing a relation in a decomposition of  $R$ 
  - Consider  $R = (A, B, C, D, E)$ , with  $F = \{ A \rightarrow B, BC \rightarrow D \}$
  - Decompose  $R$  into  $R_1 = (A, B)$  and  $R_2 = (A, C, D, E)$
  - Neither of the dependencies in  $F$  contain only attributes from  $R_2$  so we might be misled into thinking  $R_2$  satisfies BCNF.
  - In fact, dependency  $AC \rightarrow D$  in  $F^+$  shows  $R_2$  is not in BCNF.
    - $A \rightarrow B \Rightarrow AC \rightarrow BC$  (by Pseudotransitivity rule)
    - $AC \rightarrow BC$  and  $BC \rightarrow D \Rightarrow AC \rightarrow D$



# Testing Decomposition for BCNF

违反 BCNF  
2F

To check if a relation  $R_i$  in a decomposition of  $R$  is in BCNF

- ① Either test  $R_i$  for BCNF with respect to the restriction of  $F^+$  to  $R_i$  (that is, all FDs in  $F^+$  that contain only attributes from  $R_i$ ) BCNF for  $R_2$
- ② Or use the original set of dependencies  $F$  that hold on  $R$ , but with the following test:
  - for every set of attributes  $\alpha \subseteq R_i$ , check that  $\alpha^+$  (the attribute closure of  $\alpha$ ) either includes no attribute of  $(R_i - \alpha)$ , or includes all attributes of  $R_i$ .  $\rightarrow R_i$  satisfies BCNF
  - If the condition is violated by some  $\alpha$  in  $R_i$ , the dependency  $\alpha \rightarrow (\alpha^+ - \alpha) \cap R_i$  can be shown to hold on  $R_i$ , and  $R_i$  violates BCNF.
  - We use above dependency to decompose  $R_i$
  - Example:  $R_2 = (A, B)$ 
    - $R_2 = (A, C, D, E)$ ,  $F = \{ A \rightarrow B, BC \rightarrow D \}$
    - $\alpha = \{A, C\} \subseteq R_2$ ,  $(AC)^+ = \{ABCD\}$  includes the attribute  $D$  from  $(R_2 - \alpha) = \{D, E\}$
    - $AC \rightarrow ((ABCD) - \{AC\}) = \{BD\} \cap R_2 = \{D\} \rightarrow AC \rightarrow D$
    - $R_2 = (A, C, D, E)$  is decomposed into  $R_3 = \{A, C, D\}$  and  $R_4 = \{A, C, E\}$ 
      - $R_3$  and  $R_4$  are in BCNF

relation  $R_2$  is not BCNF  
non-trivial FD  $AC \rightarrow D$

①

TR2 is not BCNF



# BCNF Decomposition Algorithm

①  $\text{result} := \{R\};$  ABCDE  
 $\text{done} := \text{false};$   $F = \{A \rightarrow B, BC \rightarrow D\}$   $\text{BCNF}: \text{It's 3NF 上的子集}$   
compute  $F^+$ ; ②  $A$  is Superkey  
**while** ( $\text{not done}$ ) **do**  $BC \rightarrow D \notin \text{BCNF}$   $\text{BCNF} \subset \text{3NF}$   
③ **if** (there is a schema  $R_i$  in  $\text{result}$  that is not in BCNF)  
**then begin** .  $BC \rightarrow D$  .  
let  $\alpha \rightarrow \beta$  be a nontrivial functional dependency that  
holds on  $R_i$  such that  $\alpha^+$  does not contain  $R_i$ ,  
and  $\alpha \cap \beta = \emptyset$ ;  $BC \rightarrow D$  (not BCNF)  
 $\text{result} := (\text{result} - R_i) \cup (R_i - \beta) \cup (\alpha, \beta);$   
**end** .  $R_i - D$  .  $BCD$   
**else**  $\text{done} := \text{true};$   $F_t = A \rightarrow E$

Note: each  $R_i$  is in BCNF, and decomposition is lossless-join.



## Example of BCNF Decomposition

- class (*course\_id*, title, dept\_name, credits, *sec\_id*, semester, year, building, room\_number, capacity, time\_slot\_id)
  - Functional dependencies:
 
$$F = \left\{ \begin{array}{l} \text{course\_id} \rightarrow \text{title}, \text{dept\_name}, \text{credits} \\ \text{building}, \text{room\_number} \rightarrow \text{capacity} \\ \text{course\_id}, \text{sec\_id}, \text{semester}, \text{year} \rightarrow \text{building}, \text{room\_number}, \\ \text{time\_slot\_id} \end{array} \right. \begin{array}{l} \text{Superkey} \\ \vdots \\ \vdots \end{array}$$
  - A candidate key {*course\_id*, *sec\_id*, semester, year}.
  - BCNF Decomposition:
    - $\text{course\_id} \rightarrow \text{title}, \text{dept\_name}, \text{credits}$  holds
      - but *course\_id* is not a superkey.  $\Rightarrow$  Not BCNF
    - We replace class by: *course\_id*

$R_1 = \text{course}(\text{course\_id}, \text{title}, \text{dept\_name}, \text{credits}) = (\alpha, \beta)$

$R_2 = \text{class-1}(\text{course\_id}, \text{sec\_id}, \text{semester}, \text{year}, \text{building}, \text{room\_number}, \text{capacity}, \text{time\_slot\_id}) = R - \beta$



## BCNF Decomposition

$R_1 = \text{course}(\text{course\_id}, \text{title}, \text{dept\_name}, \text{credits}) = (\alpha, \beta)$   
 $R_2 = \text{class-1}(\text{course\_id}, \text{sec\_id}, \text{semester}, \text{year}, \text{building}, \text{room\_number}, \text{capacity}, \text{time\_slot\_id}) = R - \beta$

$$R_1 \not\models F_1 = f(\text{course\_id} \rightarrow \text{dept\_name}, \text{credits})^{\text{title}} \\ \Rightarrow \text{BCNF}$$

- *course* is in BCNF
  - How do we know this?
- *building, room\_number → capacity* holds on *class-1*
  - but  $\{\text{building}, \text{room\_number}\}$  is not a superkey for *class-1*.

$$R_3 = \text{classroom}(\text{building}, \text{room\_number}, \text{capacity})$$

$$F_2 = f(\text{building}, \text{room\_number} \xrightarrow{\alpha} \text{capacity}, \text{course\_id}, \dots, \dots)$$

- *section (course\_id, sec\_id, semester, year, building, room\_number, time\_slot\_id)*

$$P_3 = \alpha\beta = bu$$

$$P_4 = P_2 - \beta =$$

- *classroom and section* are in BCNF.



## Third Normal Form

- There are some situations where
  - BCNF is not dependency preserving, and
  - efficient checking for FD violation on updates is important
- Solution: define a weaker normal form, called Third Normal Form (3NF)
  - Allows some redundancy (with resultant problems; we will see examples later)
  - But functional dependencies can be checked on individual relations without computing a join.
  - There is always a lossless-join, dependency-preserving decomposition into 3NF.



## 3NF Example -- Relation *dept\_advisor*

$$\begin{array}{ll} \text{su} & \text{sk} = s\_id, \text{dept\_name} \\ & \text{sf} = i\_id \text{ dep} \end{array}$$

- $\text{dept\_advisor}(s\_ID, i\_ID, \text{dept\_name})$   $\text{sf} = s\_id, i\_id$   
 $F = \{s\_ID, \text{dept\_name} \rightarrow i\_ID, i\_ID \rightarrow \text{dept\_name}\}$
  - Two candidate keys:  $s\_ID, \text{dept\_name}$ , and  $i\_ID, s\_ID$
  - $R$  is in 3NF  $\downarrow$ 
    - $s\_ID, \text{dept\_name} \rightarrow i\_ID$ 
      - $(s\_ID, \text{dept\_name})$  is a superkey
    - $i\_ID \rightarrow \boxed{\text{dept\_name}}$   $\cdot \beta - \alpha = \text{dept\_name}$ 
      - $\text{dept\_name}$  is contained in a candidate key
- if not dept.name ?? - ?*



## Testing for 3NF

- Need to check only FDs in  $F$ , need not check all FDs in  $F^+$ .
- Use attribute closure to check for each dependency  $\alpha \rightarrow \beta$ , if  $\alpha$  is a superkey.
- If  $\alpha$  is not a superkey, we have to verify if each attribute in  $\beta$  is contained in a candidate key of  $R$ 
  - This test is rather more expensive, since it involves finding candidate keys
  - **Testing for 3NF has been shown to be NP-hard**
  - Interestingly, decomposition into third normal form (described shortly) can be done in polynomial time



## 3NF Decomposition Algorithm

Let  $F_c$  be a canonical cover for  $F$ ;  
 $i := 0$ ;

**for each functional dependency  $\alpha \rightarrow \beta$  in  $F_c$  do**

**begin**

$i := i + 1$ ;

$R_i := \alpha \beta$

canonical cover  
relation  $1 \leq i \leq$   
 $R_1 R_2 \dots R_i$ .

**end**

**if none** of the schemas  $R_j$ ,  $1 \leq j \leq i$  contains a candidate key for  $R$

**then begin**

$i := i + 1$ ;

$R_i :=$  any candidate key for  $R$ ;

**end**

/\* Optionally, remove redundant relations \*/

**repeat**

**if** any schema  $R_j$  is contained in another schema  $R_k$

**then** /\* delete  $R_j$  \*/

$R_j = R_i$ ;

$i = i - 1$ ;

**return**  $(R_1, R_2, \dots, R_i)$



## 3NF Decomposition Algorithm (Cont.)

Above algorithm ensures

- Each relation schema  $R_i$  is in 3NF
- Decomposition is dependency preserving and lossless-join
- Proof of correctness is at end of this presentation ([click here](#))



## 3NF Decomposition: An Example

- Relation schema:  
 $\text{cust\_banker\_branch} = (\underline{\text{customer\_id}}, \underline{\text{employee\_id}}, \text{branch\_name}, \text{type})$   
PK  
↓
- The functional dependencies for this relation schema are:
  - $\text{customer\_id}, \text{employee\_id} \rightarrow \text{branch\_name}, \text{type}$  (*superkey*  $\rightarrow$  *BCNF*)
  - $\text{employee\_id} \rightarrow \text{branch\_name}$
  - $\text{customer\_id}, \text{branch\_name} \rightarrow \text{employee\_id}$
- We first compute a canonical cover
  - $\text{branch\_name}$  is extraneous in the r.h.s. of the 1<sup>st</sup> dependency
  - No other attribute is extraneous, so we get  $F_C =$   
 $\text{customer\_id}, \text{employee\_id} \rightarrow \text{type}$   
 $\text{employee\_id} \rightarrow \text{branch\_name}$   
 $\text{customer\_id}, \text{branch\_name} \rightarrow \text{employee\_id}$



## 3NF Decomposition Example (Cont.)

- The **for** loop generates following 3NF schema:
  - $(\underline{\text{customer\_id}}, \underline{\text{employee\_id}}, \text{type})$
  - $(\underline{\text{employee\_id}}, \text{branch\_name})$
  - $(\underline{\text{customer\_id}}, \text{branch\_name}, \underline{\text{employee\_id}})$
- Observe that  $(\underline{\text{customer\_id}}, \underline{\text{employee\_id}}, \text{type})$  contains a candidate key of the original schema, so no further relation schema needs be added
- At end of for loop, detect and delete schemas, such as  $(\underline{\text{employee\_id}}, \text{branch\_name})$ , which are subsets of other schemas
  - result will not depend on the order in which FDs are considered
- The resultant simplified 3NF schema is:
  - $(\underline{\text{customer\_id}}, \underline{\text{employee\_id}}, \text{type})$
  - $(\underline{\text{customer\_id}}, \text{branch\_name}, \underline{\text{employee\_id}})$



## Comparison of BCNF and 3NF

- It is always possible to decompose a relation into a set of relations that are in 3NF such that:
  - The decomposition is lossless
  - The dependencies are preserved
- It is always possible to decompose a relation into a set of relations that are in BCNF such that:
  - The decomposition is lossless
  - It may not be possible to preserve dependencies.



## Design Goals

- Goal for a relational database design is:
  - BCNF.
  - Lossless join.
  - Dependency preservation.
- If we cannot achieve this, we accept one of
  - Lack of dependency preservation
  - Redundancy due to use of 3NF
- Interestingly, SQL does not provide a direct way of specifying functional dependencies other than superkeys.  
**Can specify FDs using assertions, but they are expensive to test, (and currently not supported by any of the widely used databases!)**
- Even if we had a dependency preserving decomposition, using SQL we would not be able to efficiently test a functional dependency whose left hand side is not a key.



## Multivalued Dependencies (MVDs)

- Suppose we record names of children, and phone numbers for instructors:
  - $\text{inst\_child}(ID, \text{child\_name})$
  - $\text{inst\_phone}(ID, \text{phone\_number})$
- If we were to combine these schemas to get
  - $\text{inst\_info}(ID, \text{child\_name}, \text{phone\_number})$

Example data:

(99999, David, 512-555-1234)  
(99999, William, 512-555-4321)  
(99999, William, 512-555-1234)  
(99999, David, 512-555-4321)

- This relation is in BCNF
  - Why?



## Multivalued Dependencies

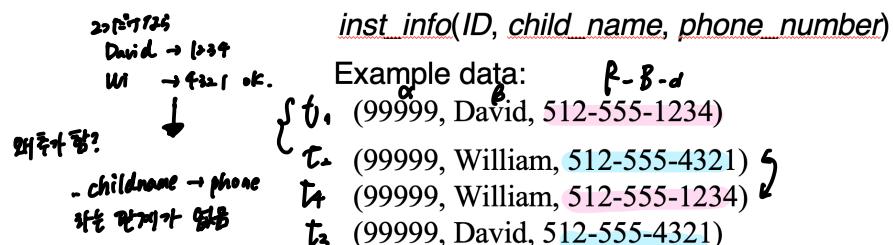
- Let  $R$  be a relation schema and let  $\alpha \subseteq R$  and  $\beta \subseteq R$ . The **multivalued dependency**

$$\alpha \rightarrow\!\!\!\rightarrow \beta$$

holds on  $R$  if in any legal relation  $r(R)$ , for all pairs for tuples  $t_1$  and  $t_2$  in  $r$  such that  $t_1[\alpha] = t_2[\alpha]$ , there exist tuples  $t_3$  and  $t_4$  in  $r$  such that:

$$\begin{aligned}t_1[\alpha] &= t_2[\alpha] = t_3[\alpha] = t_4[\alpha] \\t_3[\beta] &= t_1[\beta] \quad \text{David} \\t_3[R - \beta] &= t_2[R - \beta] \quad 4321 \\t_4[\beta] &= t_2[\beta] \quad \text{William} \\t_4[R - \beta] &= t_1[R - \beta] \quad 1234\end{aligned}$$

QH





## MVD -- Tabular representation

- Tabular representation of  $\alpha \rightarrow\!\!\!\rightarrow \beta$

≠  
→

	$\alpha$	$\beta$	$R - \alpha - \beta$
$t_1$	$a_1 \dots a_i$	$a_{i+1} \dots a_j$	$a_{j+1} \dots a_n$
$t_2$	$a_1 \dots a_i$	$b_{i+1} \dots b_j$	$b_{j+1} \dots b_n$
$t_3$	$a_1 \dots a_i$	$a_{i+1} \dots a_j$	$b_{j+1} \dots b_n$
$t_4$	$a_1 \dots a_i$	$b_{i+1} \dots b_j$	$a_{j+1} \dots a_n$

$$\begin{aligned}
 t_1[\alpha] &= t_2[\alpha] = t_3[\alpha] = t_4[\alpha] \\
 t_3[\beta] &\quad | \quad = t_1[\beta] \\
 t_3[R - \beta] &= t_2[R - \beta] \\
 t_4[\beta] &\quad = t_2[\beta] \\
 t_4[R - \beta] &= t_1[R - \beta]
 \end{aligned}$$



## MVD (Cont.)

- Let  $R$  be a relation schema with a set of attributes that are partitioned into 3 nonempty subsets.

$Y, Z, W$

- We say that  $Y \rightarrow\!\!\!\rightarrow Z$  ( $Y$  multidetermines  $Z$ ) if and only if for all possible relations  $r(R)$

$< y_1, z_1, w_1 > \in r$  and  $< y_1, z_2, w_2 > \in r$

then

~~$< y_1, z_1, w_2 > \in r$  and  $< y_1, z_2, w_1 > \in r$~~

- Note that since the behavior of  $Z$  and  $W$  are identical it follows that

$Y \rightarrow\!\!\!\rightarrow Z$  if  $Y \rightarrow\!\!\!\rightarrow W$

$ID \quad name \quad ID \quad phone$

*inst\_info*(*ID*, *child\_name*, *phone\_number*)

Example data:

(99999, David, 512-555-1234)  
(99999, David, 512-555-4321)  
(99999, William, 512-555-1234)  
(99999, William, 512-555-4321)



## Example

- In our example:

$$ID \rightarrow\!\!\! \rightarrow child\_name$$
$$ID \rightarrow\!\!\! \rightarrow phone\_number$$

- The above formal definition is supposed to formalize the notion that given a particular value of  $Y$  ( $ID$ ) it has associated with it a set of values of  $Z$  ( $child\_name$ ) and a set of values of  $W$  ( $phone\_number$ ), and these two sets are in some sense independent of each other.
- Note:
  - If  $Y \rightarrow Z$  then  $Y \rightarrow\!\!\! \rightarrow Z$
  - Indeed we have (in above notation)  $Z_1 = Z_2$   
The claim follows.



- $R(A,B,C)$
  - $\{ A \rightarrow\!\!\! \rightarrow B \}$  *인수는 가리키지 않음*
  - $\{ (a,b1,c1), (a,b2,c2), (a,b3,c3) \}$ 
    - (a,b1,c\underline{2}) (a,b1,c\underline{3}) (a,b\underline{2}, c1) (a,b2,c\underline{3}) (a,b3,c1) (a,b3,c2)
    - 추가*
- (a, b, c\underline{2})    (a, b\underline{1}, c\underline{3})    (a, b\underline{2}, c\underline{1})    (a, b\underline{2}, c\underline{3})    (a, b\underline{3}, c\underline{1})    (a, b\underline{3}, c\underline{2})*



## Use of Multivalued Dependencies

- We use multivalued dependencies in two ways:
  1. To test relations to **determine** whether they are legal under a given set of functional and multivalued dependencies
  2. To specify **constraints** on the set of legal relations. We shall concern ourselves *only* with relations that satisfy a given set of functional and multivalued dependencies.
- If a relation  $r$  fails to satisfy a given multivalued dependency, we can construct a relations  $r'$  that does satisfy the multivalued dependency by adding tuples to  $r$ .



## Theory of MVDs

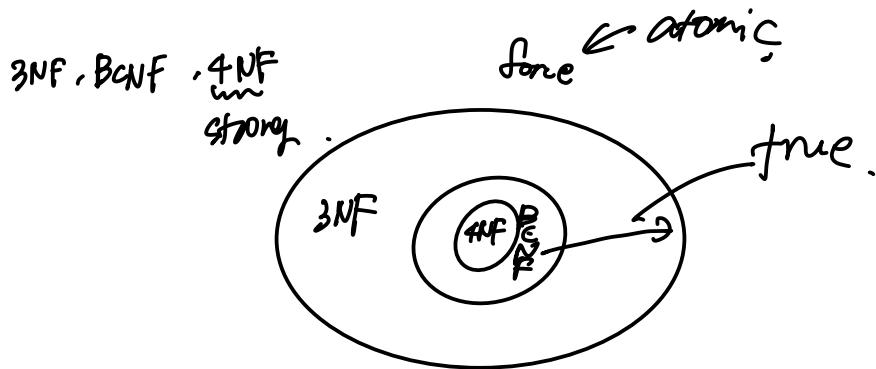
- From the definition of multivalued dependency, we can derive the following rule:
  - If  $\alpha \rightarrow \beta$ , then  $\alpha \rightarrow\rightarrow \beta$

That is, every functional dependency is also a multivalued dependency
- The **closure**  $D^+$  of  $D$  is the set of all functional and multivalued dependencies logically implied by  $D$ .
  - We can compute  $D^+$  from  $D$ , using the formal definitions of functional dependencies and multivalued dependencies.
  - We can manage with such reasoning for very simple multivalued dependencies, which seem to be most common in practice
  - For complex dependencies, it is better to reason about sets of dependencies using a system of inference rules (Appendix C).



## Fourth Normal Form

- A relation schema  $R$  is in **4NF** with respect to a set  $D$  of functional and multivalued dependencies if for all multivalued dependencies in  $D^+$  of the form  $\alpha \rightarrow\!\!\!\rightarrow \beta$ , where  $\alpha \subseteq R$  and  $\beta \subseteq R$ , at least one of the following hold:
  - $\alpha \rightarrow\!\!\!\rightarrow \beta$  is trivial (i.e.,  $\beta \subseteq \alpha$ )
  - $\alpha$  is a superkey for schema  $R$
- **If a relation is in 4NF it is in BCNF**



## Restriction of Multivalued Dependencies

- The restriction of  $D$  to  $R_i$  is the set  $D_i$  consisting of
  - All functional dependencies in  $D^+$  that include only attributes of  $R_i$
  - All multivalued dependencies of the form
$$\alpha \rightarrow\!\!\!\rightarrow (\beta \cap R_i)$$
where  $\alpha \subseteq R_i$  and  $\alpha \rightarrow\!\!\!\rightarrow \beta$  is in  $D^+$



# 4NF Decomposition Algorithm

*nontrivial*

$\text{result} := \{R\}; R = \{\text{student, course, Book}\}$   $F = \{\text{student} \rightarrow \text{course}, \text{course} \rightarrow \text{Book}\}$

$\text{done} := \text{false};$   $P_1 = (R - \emptyset) = R - \text{Book} = (\text{student, course})$

$\text{compute } D^+;$   $P_2 = (\alpha \beta) = (\text{course, book})$

Let  $D_i$  denote the restriction of  $D^+$  to  $R_i$

**while** (*not done*)

**if** (*there is a schema  $R_i$  in result that is not in 4NF*) **then**

**begin**

let  $\alpha \rightarrow \beta$  be a nontrivial multivalued dependency that holds  
on  $R_i$  such that  $\alpha \rightarrow R_i$  is not in  $D_i$ , and  $\alpha \cap \beta = \emptyset$ ;

$\text{result} := (\text{result} - R_i) \cup (R_i - \beta) \cup (\alpha, \beta);$

**end**

**else**  $\text{done} := \text{true};$

Note: each  $R_i$  is in 4NF, and decomposition is lossless-join



## Example

- $R = (A, B, C, G, H, I)$   
 $F = \{ A \xrightarrow{\alpha} B, B \xrightarrow{\beta} HI, CG \xrightarrow{\gamma} H \}$ 
  - A relation schema  $R$  is in **4NF** with respect to a set  $D$  of functional and multivalued dependencies if for all multivalued dependencies in  $D^+$  of the form  $\alpha \rightarrow \beta$ , where  $\alpha \subseteq R$  and  $\beta \subseteq R$ , at least one of the following hold:
    - ①  $\alpha \rightarrow \beta$  is trivial (i.e.,  $\beta \subseteq \alpha$  or  $\alpha \cup \beta = R$ )
    - ②  $\alpha$  is a **superkey** for schema  $R$
    - If a relation is in 4NF it is in BCNF
- $\textcircled{i}$   $R$  is not in 4NF since  $A \xrightarrow{\alpha} B$  and  $A$  is not a superkey for  $R$
- Decomposition
  - a)  $R_1 = (A, B)$   $(R_1 \text{ is in 4NF})$
  - b)  $R_2 = (A, C, G, H, I)$   $(R_2 \text{ is not in 4NF, decompose into } R_3 \text{ and } R_4)$
  - c)  $R_3 = (C, G, H)$   $(R_3 \text{ is in 4NF})$
  - d)  $R_4 = (A, C, G, I)$   $R_2 - H$   $(R_4 \text{ is not in 4NF, decompose into } R_5 \text{ and } R_6)$ 
    - $A \rightarrow B$  and  $B \rightarrow HI \Rightarrow A \xrightarrow{\alpha} HI$ , (MVD transitivity), and
    - and hence  $A \xrightarrow{\alpha} I$  (MVD restriction to  $R_4$ )
  - e)  $R_5 = (A, I)$   $(R_5 \text{ is in 4NF})$
  - f)  $R_6 = (A, C, G)$   $(R_6 \text{ is in 4NF})$



## Further Normal Forms

- **Join dependencies** generalize multivalued dependencies
  - lead to **project-join normal form (PJNF)** (also called **fifth normal form**)
- A class of even more general constraints, leads to a normal form called **domain-key normal form**.
- Problem with these generalized constraints: are hard to reason with, and no set of sound and complete set of inference rules exists.
- Hence rarely used



## Overall Database Design Process

We have assumed schema  $R$  is given

- $R$  could have been generated when converting E-R diagram to a set of tables.
- $R$  could have been a single relation containing *all* attributes that are of interest (called **universal relation**).
- Normalization breaks  $R$  into smaller relations.
- $R$  could have been the result of some ad hoc design of relations, which we then test/convert to normal form.

*Design  $\rightarrow$  PFBPBD.*



## ER Model and Normalization

- When an E-R diagram is carefully designed, identifying all entities correctly, the tables generated from the E-R diagram should not need further normalization.
- However, in a real (imperfect) design, there can be functional dependencies from non-key attributes of an entity to other attributes of the entity
  - Example: an *employee* entity with
    - attributes *department\_name* and *building*,
    - functional dependency  $\text{department\_name} \rightarrow \text{building}$
  - Good design would have made department an entity
- Functional dependencies from non-key attributes of a relationship set possible, but rare --- most relationships are binary



## Denormalization for Performance

- May want to use non-normalized schema for performance
- For example, displaying *prereqs* along with *course\_id*, and *title* requires join of *course* with *prereq*
- Alternative 1: Use denormalized relation containing attributes of *course* as well as *prereq* with all above attributes
  - faster lookup
  - extra space and extra execution time for updates
  - extra coding work for programmer and possibility of error in extra code
- Alternative 2: use a materialized view defined a  $\text{course} \bowtie \text{prereq}$ 
  - Benefits and drawbacks same as above, except no extra coding work for programmer and avoids possible errors



## Other Design Issues

- Some aspects of database design are not caught by normalization
- Examples of bad database design, to be avoided:

Instead of *earnings (company\_id, year, amount)*, use

- *earnings\_2004, earnings\_2005, earnings\_2006*, etc., all on the schema *(company\_id, earnings)*.
  - Above are in BCNF, but make querying across years difficult and needs new table each year
- *company\_year (company\_id, earnings\_2004, earnings\_2005, earnings\_2006)*
  - Also in BCNF, but also makes querying across years difficult and requires new attribute each year.
  - Is an example of a **crosstab**, where values for one attribute become column names
  - Used in spreadsheets, and in data analysis tools



## Modeling Temporal Data

- **Temporal data** have an association time interval during which the data are *valid*.
- A **snapshot** is the value of the data at a particular point in time
- Several proposals to extend ER model by adding valid time to
  - attributes, e.g., address of an instructor at different points in time
  - entities, e.g., time duration when a student entity exists
  - relationships, e.g., time during which an instructor was associated with a student as an advisor.
- But no accepted standard
- Adding a temporal component results in functional dependencies like
$$ID \rightarrow street, city$$
not holding, because the address varies over time
- A **temporal functional dependency**  $X \rightarrow Y$  holds on schema  $R$  if the functional dependency  $X \rightarrow Y$  holds on all snapshots for all legal instances  $r(R)$ .



## Modeling Temporal Data (Cont.)

- In practice, database designers may add start and end time attributes to relations
  - E.g.,  $\text{course}(\text{course\_id}, \text{course\_title})$  is replaced by  
 $\text{course}(\text{course\_id}, \text{course\_title}, \text{start}, \text{end})$
  - Constraint: no two tuples can have overlapping valid times
    - Hard to enforce efficiently
- Foreign key references may be to current version of data, or to data at a point in time
  - E.g., student transcript should refer to course information at the time the course was taken



## End of Chapter 7



## Correctness of 3NF Decomposition Algorithm

- 3NF decomposition algorithm is dependency preserving (since there is a relation for every FD in  $F_c$ )
- Decomposition is lossless
  - A candidate key ( $C$ ) is in one of the relations  $R_i$  in decomposition
  - Closure of candidate key under  $F_c$  must contain all attributes in  $R$ .
  - Follow the steps of attribute closure algorithm to show there is only one tuple in the join result for each tuple in  $R_i$



## Correctness of 3NF Decomposition Algorithm (Cont.)

- Claim: if a relation  $R_i$  is in the decomposition generated by the above algorithm, then  $R_i$  satisfies 3NF.
- Proof:
  - Let  $R_i$  be generated from the dependency  $\alpha \rightarrow \beta$
  - Let  $\gamma \rightarrow B$  be any non-trivial functional dependency on  $R_i$ . (We need only consider FDs whose right-hand side is a single attribute.)
  - Now,  $B$  can be in either  $\beta$  or  $\alpha$  but not in both. Consider each case separately.



## Correctness of 3NF Decomposition (Cont.)

- Case 1: If  $B$  in  $\beta$ :
  - If  $\gamma$  is a superkey, the 2nd condition of 3NF is satisfied
  - Otherwise  $\alpha$  must contain some attribute not in  $\gamma$
  - Since  $\gamma \rightarrow B$  is in  $F^+$  it must be derivable from  $F_c$ , by using attribute closure on  $\gamma$ .
  - Attribute closure not have used  $\alpha \rightarrow \beta$ . If it had been used,  $\alpha$  must be contained in the attribute closure of  $\gamma$ , which is not possible, since we assumed  $\gamma$  is not a superkey.
  - Now, using  $\alpha \rightarrow (\beta - \{B\})$  and  $\gamma \rightarrow B$ , we can derive  $\alpha \rightarrow B$  (since  $\gamma \subseteq \alpha \beta$ , and  $B \notin \gamma$  since  $\gamma \rightarrow B$  is non-trivial)
  - Then,  $B$  is extraneous in the right-hand side of  $\alpha \rightarrow \beta$ ; which is not possible since  $\alpha \rightarrow \beta$  is in  $F_c$ .
  - Thus, if  $B$  is in  $\beta$  then  $\gamma$  must be a superkey, and the second condition of 3NF must be satisfied.



## Correctness of 3NF Decomposition (Cont.)

- Case 2:  $B$  is in  $\alpha$ .
  - Since  $\alpha$  is a candidate key, the third alternative in the definition of 3NF is trivially satisfied.
  - In fact, we cannot show that  $\gamma$  is a superkey.
  - This shows exactly why the third alternative is present in the definition of 3NF.

Q.E.D.



## First Normal Form

- Domain is **atomic** if its elements are considered to be indivisible units
  - Examples of non-atomic domains:
    - Set of names, composite attributes
    - Identification numbers like CS101 that can be broken up into parts
- A relational schema R is in **first normal form** if the domains of all attributes of R are atomic
- Non-atomic values complicate storage and encourage redundant (repeated) storage of data
  - Example: Set of accounts stored with each customer, and set of owners stored with each account
  - We assume all relations are in first normal form (and revisit this in Chapter 22: Object Based Databases)



## First Normal Form (Cont.)

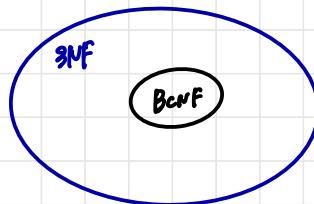
- Atomicity is actually a property of how the elements of the domain are used.
  - Example: Strings would normally be considered indivisible
  - Suppose that students are given roll numbers which are strings of the form *CS0012* or *EE1127*
  - If the first two characters are extracted to find the department, the domain of roll numbers is not atomic.
  - Doing so is a bad idea: leads to encoding of information in application program rather than in the database.

if and only if  $\kappa \rightarrow R$

$\kappa$  is superkey for relation schema  $R$

lossless decomposition

- $P_1 \cap P_2 \rightarrow P_1$
- $P_1 \cap P_2 \rightarrow P_2$



$\langle BCNF \rangle$

$\alpha \subseteq R$  and  $\beta \subseteq R$  at least one of the following holds:

- (1)  $\alpha \rightarrow \beta$  is trivial ( $\beta \subseteq \alpha$ )
- (2)  $\alpha$  is superkey for  $R$

else.  $\alpha$  is not superkey  $\Rightarrow$  NOT BCNF  
candidate key - ~~is not superkey~~

)  $2NF \rightarrow BCNF$  i.e.  
 $BCNF \neq 2NF$ .

$\langle 3NF \rangle$

$\alpha \rightarrow \beta$  inf at least one of the following holds:

- (1)  $\alpha \rightarrow \beta$  is trivial
- (2)  $\alpha$  is superkey for  $R$
- (3) Each attribute  $A$  in  $\beta - \alpha$  is contained in candidate key for  $R$ .  
↳  $\beta$  is key of  $R$ .

$\langle 4NF \rangle$

$\alpha \rightarrow \beta$  is trivial

$\alpha$  is superkey for schema  $R$

multivalued dependency

$\alpha \rightarrow \beta$

- [ 99999, David, 512-555-1234 ]
  - [ 99999, William, 512-555-4321 ]
  - [ 99999, William, 512-555-1234 ]
  - [ 99999, David, 512-555-4321 ]
- (chidname)  $\Rightarrow$  phone of ~~dog~~ dog

7.1 O

$$R = A B C D E$$

$$R_1 = (A, B, C)$$

$$R_2 = (A, D, E)$$

$$R_1 \cap R_2 = (A) \quad A \text{ is candidate key}$$

$$R_1 \cap R_2 \rightarrow R_1 \quad (A \rightarrow BC)$$

$\Rightarrow$  lossless decomposition

7.2. O

A	B	C
a <sub>1</sub>	b <sub>1</sub>	c <sub>1</sub>
a <sub>1</sub>	b <sub>1</sub>	c <sub>2</sub>
a <sub>2</sub>	b <sub>1</sub>	c <sub>1</sub>
a <sub>2</sub>	b <sub>1</sub>	c <sub>3</sub>

nontrivial

functional dependency :  $A \rightarrow B, C \rightarrow B$

$$\nmid AC \rightarrow B$$

∴ trivial  $\alpha \rightarrow \beta$  if  $\beta \subseteq \alpha$

7.3 X

$P_k(r)$  : relation r a primary key

$$P_k(\text{student}) \rightarrow P_k(\text{instructor})$$

∩

$$P_k(\text{instructor}) \rightarrow P_k(\text{student})$$

indicate a one to one relationship

student과 student는 같은 학번을 가질 때만 instructor와는 서로

같은 학번을 가질 수 있다. 즉, 같은 학생은 서로 다른 명의의 instructor를 가질 수 있다.

instructor는 서로 다른 학생과 student는 가질 수 (one to one)

- many to one student instructor  $\Rightarrow$   $P_k(\text{instructor}) \rightarrow P_k(\text{student})$  instructor is 1 to many student.
- Each instructor has 3 students. If 2 students have the same value, then they have the same instructor.
- Since any student value which is repeated will have the same instructor value but many student values may have the same instructor value.

instructor	student
a1	b1
a1	b1
a2	b1
a2	b1
.	.

7.4 O

if  $\alpha \rightarrow \beta$  and  $\alpha \rightarrow \gamma$  then  $\alpha \rightarrow \beta\gamma$   
argumentation rule

$$\left\{ \begin{array}{l} \alpha \rightarrow \beta \text{ given } \\ \alpha \rightarrow \gamma \text{ given } \end{array} \right. \quad \alpha \rightarrow \beta \wedge \alpha \rightarrow \gamma \Rightarrow \alpha \rightarrow \beta\gamma.$$

transitivity rule

$$\alpha \rightarrow (\alpha\beta \rightarrow \alpha\gamma) \rightarrow \beta\gamma$$

$$\therefore \alpha \rightarrow \beta\gamma.$$

7.5 O if  $\alpha \rightarrow \beta$  and  $\beta \rightarrow \gamma$  then  $\alpha\beta \rightarrow \gamma$

$$\alpha \rightarrow \beta \text{ given}$$

$$\alpha\beta \rightarrow \beta\gamma \text{ argumentation rule}$$

$$\alpha\beta \rightarrow \beta\gamma \text{ & } \beta\gamma \rightarrow \gamma \text{ given}$$

$$\alpha\beta \rightarrow \gamma \text{ transitivity rule } \square.$$

7.6 R = {A, B, C, D, E}



$A \rightarrow BC$

$CD \rightarrow E$

$B \rightarrow D$

$E \rightarrow A$

$A \rightarrow BC$  or  $A \rightarrow C$ ,  $A \rightarrow B$  is ~~not~~ 3.

(A<sup>†</sup>) since  $A \rightarrow B$  and  $B \rightarrow D$ ;  $A \rightarrow D$   
since  $A \rightarrow CD$  and  $CD \rightarrow E$ ,  $A \rightarrow E$ .  
since  $A \rightarrow A$ ,  $A \rightarrow ABCDE$

(E<sup>†</sup>)

result + E

result EA      ( $E \rightarrow A$ )

result AEBC    ( $A \rightarrow BC$ )

result ABCDE    ( $B \rightarrow D$ )

$\therefore (E^+) \supseteq R$

(BC)<sup>†</sup>

result BC D    ( $BC \rightarrow D$ )

result BC DE    ( $CD \rightarrow E$ )

result ABCDE    ( $E \rightarrow A$ )

$(BC) \supseteq R$

(CD)<sup>†</sup>

result CDE      ( $CD \rightarrow E$ )

result A CDE    ( $E \rightarrow A$ )

result ABCDE    ( $A \rightarrow BC$ )

$(CD) \supseteq R$

$\therefore$  candidate key is A, BC, CD, E

7.7 Canonical cover

$A \rightarrow BC$

$CD \rightarrow E$

$B \rightarrow D$

$E \rightarrow A$

In R3.  $R^{\text{BCNF}}$  unique - extraneous attr.

7.8

7.43

7.9 R (A,B,C)

B  $\rightarrow$  C or not?

B  $\rightarrow$  C  $\rightarrow$  A

B<sub>1</sub> ✓ C<sub>1</sub>

B<sub>2</sub> C<sub>1</sub>

B<sub>3</sub> C<sub>1</sub>

B<sub>4</sub> ✓ C<sub>2</sub>

B<sub>5</sub> ✓ C<sub>3</sub>

X X B<sub>1</sub> C<sub>3</sub>

a. Select B

from r

group by B

having count(distinct C) > 1

↳ true (25 Bsc Upto 25)

b. enforce the functional dependency.

Create assertion b to C check

not exist C

Select B

from r

gro

7.10

Q. Our discussion of lossless decomposition implicitly assumed that attributes on the left side of functional dependency can not take on null values

$\Rightarrow$  natural join

excludes

[ all attributes on NULL value of 2nd tuple ]  
if #F1 exclude E0f.

## 7.11 3NF

$$\alpha \rightarrow \theta$$

$r(\alpha, \beta, r)$  into

$r_1(\alpha, \beta)$  and  $r_2(\alpha, r)$

$$r_1 \cap r_2 = \alpha$$

①  $r_1$  of  $\alpha$  is primary key

$r_2$  of  $\alpha$  is foreign key

referencing  $r_1$ .

②  $r_2$  of  $r_1$  of  $\alpha$  is primary key

$\alpha \rightarrow \beta$  of  $r_1$ .  $\alpha \rightarrow \beta$  of  $r_2$  is functional dependency

~~3NF~~

primary key constraint:

3NF is achieved if schema is 1NF schema

primary key is PK.

foreign key constraint

$$R \rightarrow R_1 \quad R_2$$

In relation schema  $R$  there is functional dependency

attribute  $R_1$  and  $R_2$  is foreign key constraint of  $R$

7.14

$$F = \{X \rightarrow YZ, Y \rightarrow XZ, Z \rightarrow XY\}$$

$$X \rightarrow YZ \dots X \rightarrow Y$$

Z is redundant

$$F' = \{X \rightarrow Y, Y \rightarrow XZ, Z \rightarrow XY\}$$

$$(X^t) = XY$$

$X \rightarrow YZ$  can be  $X \rightarrow Y$  Z is canonical cover

$$F = \{X \rightarrow Y, Y \rightarrow XZ, Z \rightarrow XY\}$$

Y  $\rightarrow$  XZ remove Z

$$F' = \{X \rightarrow Y, Y \rightarrow X, Z \rightarrow XY\}$$

$$(Y^t) = XZY$$

$Y \rightarrow XZ$  can be  $Y \rightarrow X$  Z is canonical cover

$$F = \{X \rightarrow Y, Y \rightarrow X, Z \rightarrow XY\}$$

Z  $\rightarrow$  XY remove X

$$F' = \{X \rightarrow Y, Y \rightarrow X, Z \rightarrow Y\}$$

$$(Z^t) = ZYX$$

$$\therefore F = \{X \rightarrow Y, Y \rightarrow X, Z \rightarrow Y\}$$

7.15  $X \rightarrow YZ, Y \rightarrow XZ$  and  $Z \rightarrow XY$ 

what can go wrong if two attributes inferred to be extraneous  
are deleted at once.

 $X \rightarrow YZ$  は  $Y$  が  $X$  から  $Z$  へZ が  $X$  から  $Y$  へしかし同時に  $X \rightarrow YZ$  が  $X \rightarrow YZ$  へ $X \rightarrow YZ$  は  $Y$  が  $X$  から  $Z$  へ $X \rightarrow YZ$  は  $Z$  が  $X$  から  $Y$  へ $Y$  が  $X$  から  $Z$  へcanonical cover algorithm は  $X \rightarrow YZ$  と  $Z \rightarrow XY$ を同時に削除するが  $X \rightarrow YZ$  は  $Y$  が  $X$  から  $Z$  へ

である

7.17. △

$R = (A, B, C, D)$  the set of dependencies

$F' = A \rightarrow B, C \rightarrow D, B \rightarrow C$  allows three distinct

BCNF decomposition

$$R_1 = \{ (A, B), (C, D), (B, C) \}$$

$$R_2 = \{ (A, B), (C, D), (A, C) \}$$

$$R_3 = \{ (B, C), (A, D), (A, B) \}$$

$R_2$

$$A \rightarrow B$$

$$C \rightarrow D$$

$$A \rightarrow B, B \rightarrow C \text{ and } A \rightarrow C$$

BCNF is  $\alpha \rightarrow \beta$      $\alpha$  is superkey of  $R$ .