Database Systems
Lecture10 – Ch 7. Normalization

Beomseok Nam (남범석)
bnam@skku.edu

Combine Schemas?

- Suppose we combine *instructor* and *department* into *inst_dept*
- Result is possible repetition of information

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

Functional Dependency

- Identify functional dependencies of attributes.
  - **functional dependency**:

    dept_name → *building, budget*
  - In *inst_dept*, *dept_name* is not a candidate key.
  - But building, budget depends on dept_name.
  - The building and budget of a department need to be repeated

    →This indicates the need to ***decompose*** *inst_dept*

Decomposition

- Suppose we decompose

  employee(ID, name, street, city, salary)

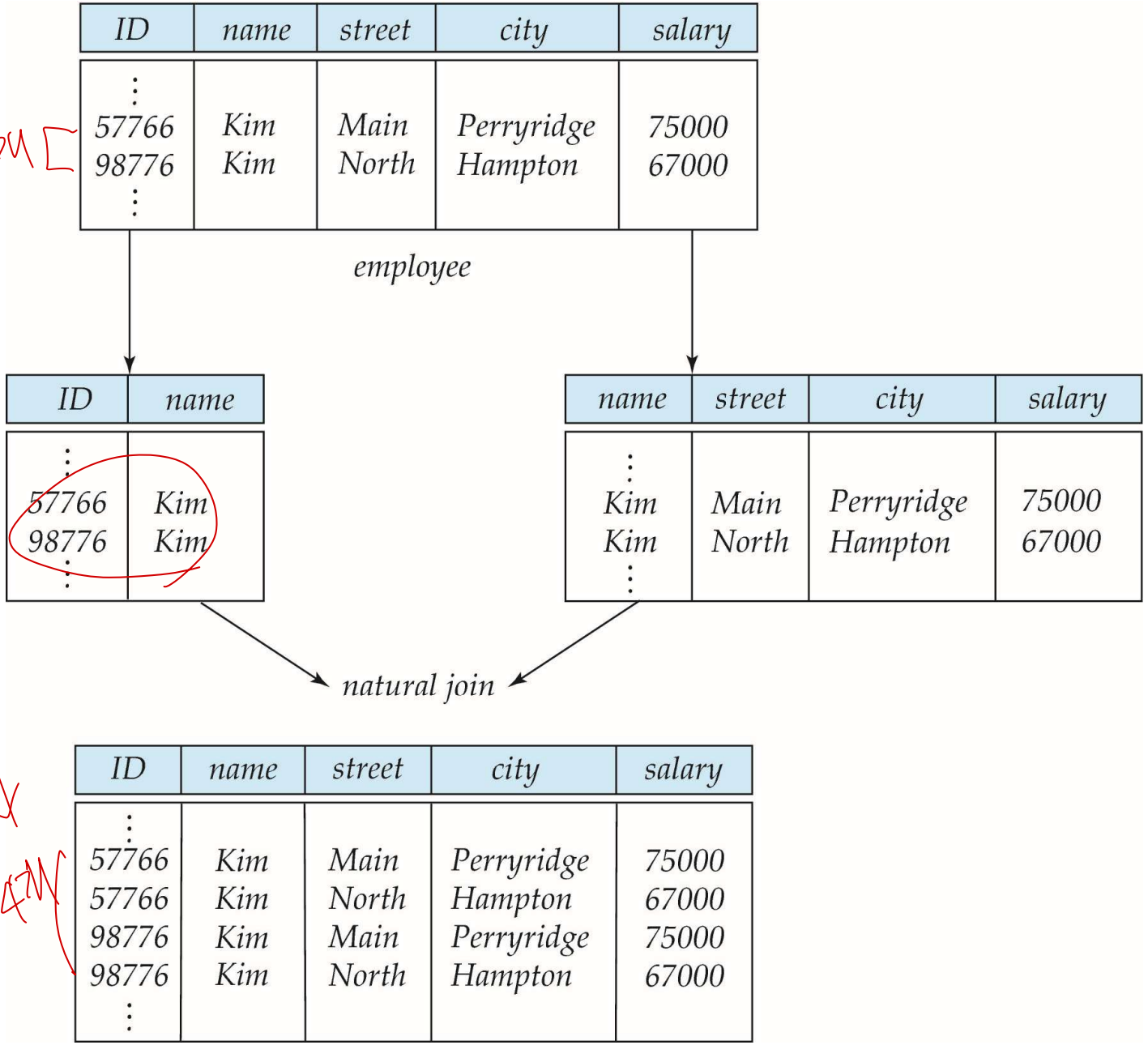
  into

  employee1 (*ID*, *name*)

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

- The problem arises when we have two employees with the same name
- We cannot reconstruct the original *employee* relation -- and so, this is a **lossy decomposition**.

Lossy Decomposition



Lossless Decomposition

- **Lossless join decomposition**
  - 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 U R_2
  - Formally,

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

  - And, conversely a decomposition is lossy if

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

Example of Lossless Decomposition

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

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

A	B	C
α	1	A
β	2	B

r

original

A	B
α	1
β	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
α	1	A
β	2	B

Normalization Theory

- Decide whether a particular relation *R* is in "good" form.
- If a relation *R* is **not** in "**good**" form, decompose it into $\{R_1, R_2, ..., R_n\}$ such that
  - each relation is in good form
  - the decomposition is a lossless-join decomposition

이 방법이어야한다.

Functional Dependencies

- 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 α, they also agree on the attributes β. 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.*

	A	B
t_1	1	4
t_2	1	5
	3	7

$t_1\alpha = t_2\alpha \rightarrow t_1\beta \neq t_2\beta$

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

Functional Dependencies (Cont.)

- *Functional dependency is a generalization of the notion of key.*
- *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 any $\alpha \subset K$, *no* $(K - \alpha) \rightarrow R$ */* K is minimal */*

- Consider the schema:

  inst_dept (*ID*, ~~*name*~~, *salary*, *dept_name*, *building*, ~~*budget*~~)

  We expect the following functional dependencies to hold:

  $dept\_name \rightarrow building$

  $ID \rightarrow building$

  but would not expect the following to hold:

  $dept\_name \rightarrow salary$

Functional Dependencies (Cont.)

- *A* functional dependency $\alpha \rightarrow \beta$ is **trivial** if $\alpha \supseteq \beta$
  - Example*:*
    - *name* $\rightarrow$ *name*
    - *ID, name* $\rightarrow$ *ID*
    - *ID, name, salary* $\rightarrow$ *salary*

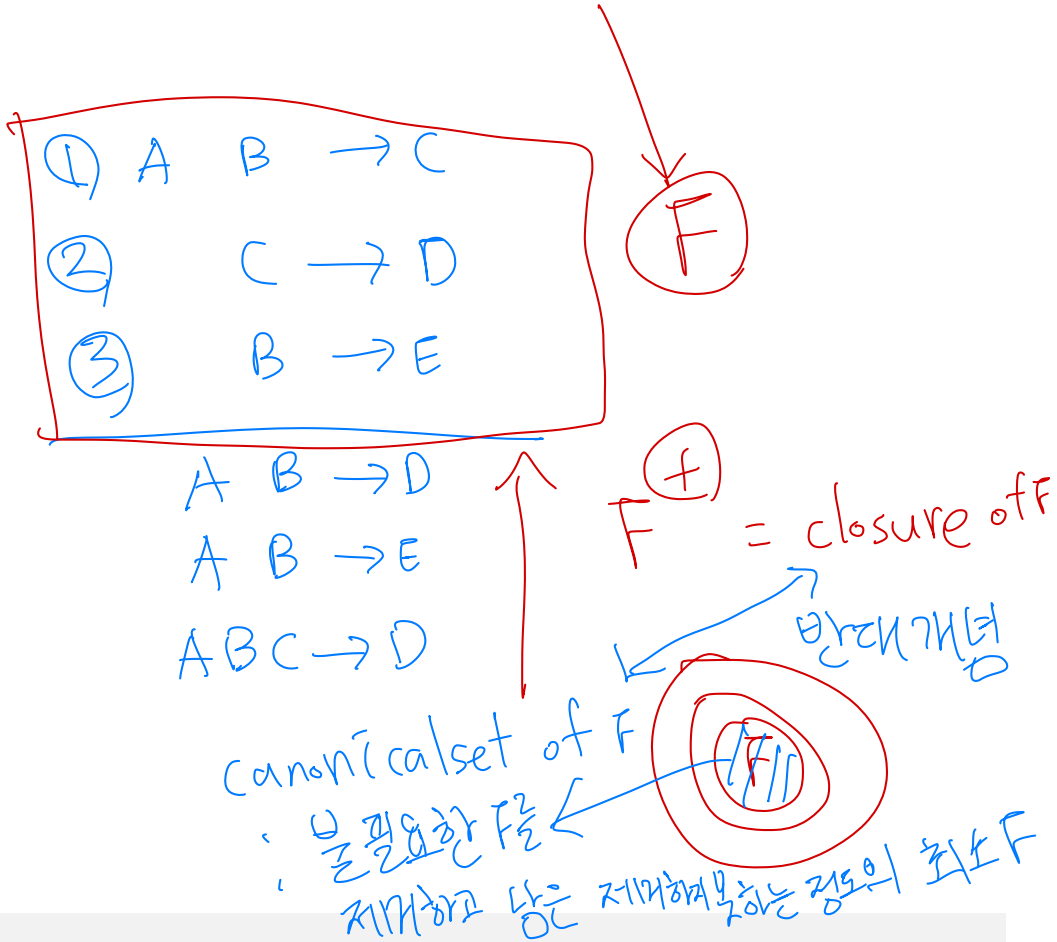
Closure of a Set of Functional Dependencies

- Functional dependencies can be derived using inference rules.
  - E.g.: If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$

ID → d_n d_n → Building

- The set of **all** functional dependencies logically implied by *F* is the **closure** of *F*.
- We denote the *closure* of *F* by **F^+**.
- F^+ is a superset of *F*.

부분집합 포함 여기서



Closure of a Set of Functional Dependencies

- We find F^+, the closure of F, by repeatedly applying **Armstrong's Axioms:**
  - if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ **(reflexivity)**
  - if $\alpha \rightarrow \beta$, then $\gamma\ \alpha \rightarrow \gamma\beta$ **(augmentation)**
  - if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ **(transitivity)**
- These rules are
  - **sound** (generate only functional dependencies that actually hold), and
  - **complete** (generate all functional dependencies that hold).

Closure of a Set of FD: Example

- $R = (A, B, C, G, H, I)$

$F = \{ A \rightarrow B$
$A \rightarrow C$
$CG \rightarrow H$
$CG \rightarrow I$
$B \rightarrow H\}$

candidate key : A, G

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

Closure of a Set of Functional Dependencies

- Additional rules:

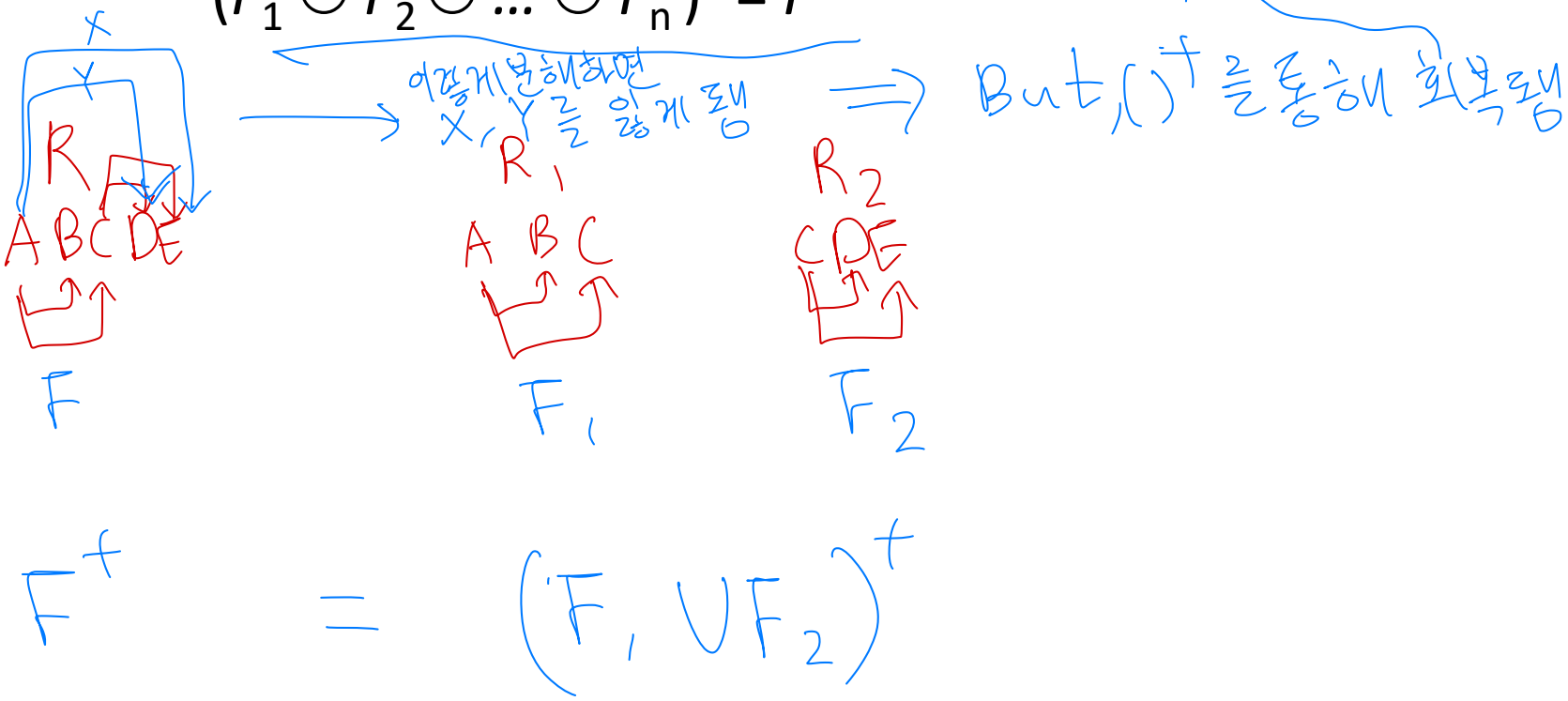
  - If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds **(union)**
  - If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds **(decomposition)**
  - If $\alpha \rightarrow \beta$ holds and $\gamma\ \beta \rightarrow \delta$ holds, then $\alpha\ \gamma \rightarrow \delta$ holds **(pseudotransitivity)**

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

Dependency Preservation

- Let F_i be the set of dependencies in F^+ for a relation R_i.
  - A decomposition is **dependency preserving**, if

$$(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$$



Boyce-Codd Normal Form (BCNF)

- Relation schema R is in BCNF

  if for all functional dependencies in F+ of the form $\alpha \rightarrow \beta$

  at least one of the following holds:

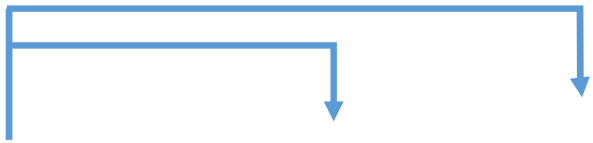
- $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
- α is a superkey for R

trivial하지 않으면 check

is contained 관계

Boyce-Codd Normal Form (BCNF)

- Example schema not in BCNF:



- *inst_dept* (*ID, name, salary, dept_name, building, budget*).

- because dept_name→ building, budget holds on instr_dept, but dept_name is not a superkey

① trivial? X
② superkey? X
⇒ BCNF X

R = (A, B, C, D)

F = { ① A → B
② BC → D }

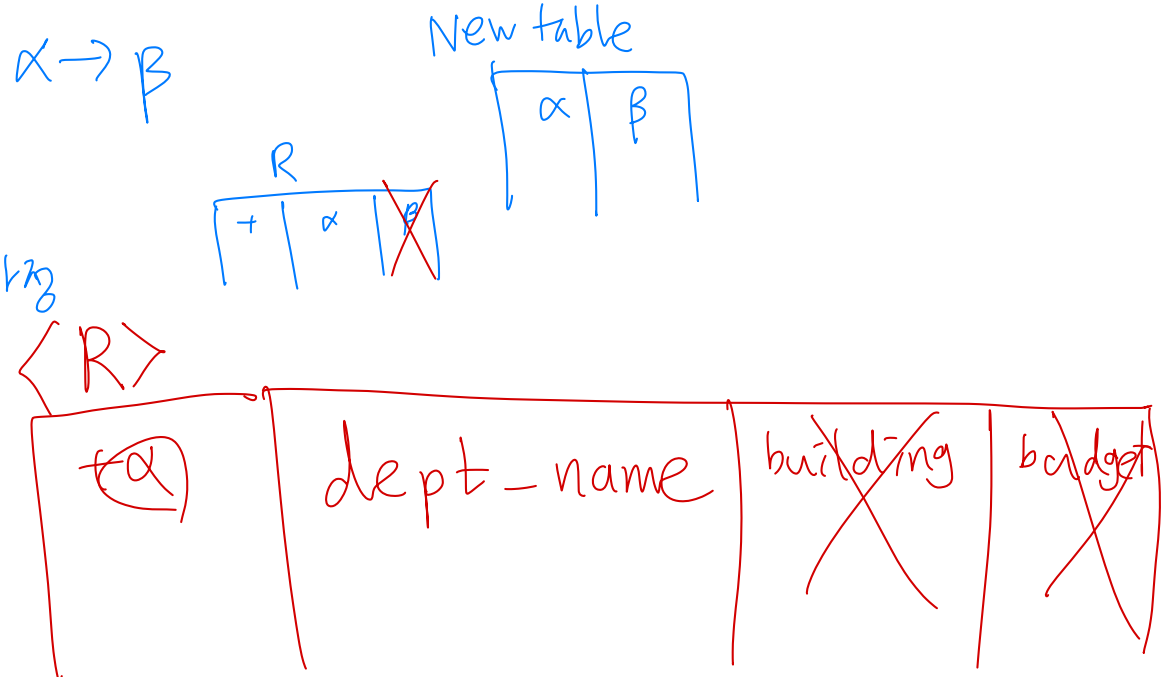
AC → D

step
① candidate key 찾기
∴ A, C
② ① BCNF 조건 체크 → violate
⇒ R_1 = (A, B) R_2 = (A, C, D)
B remove

R_1 BCNF? yes R_2 BCNF? yes
Candidate key = C.K(R_1) = {A}
C.K(R_2) = {A, C}

Decomposing a Schema into BCNF

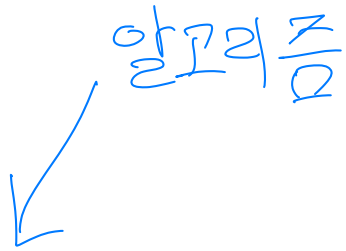
- Suppose we have a schema R and a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF.
- We decompose R into:
  - $(\alpha \cup \beta) = R_1$
  - $(R - (\beta - \alpha)) = R_2$
- In the previous example,
  - α = dept_name
  - β = building, budget



Then, inst_dept needs to be decomposed into

- $(\alpha \cup \beta)$ = (dept_name, building, budget)
- (R - (β - α)) = (ID, name, dept_name, salary)

BCNF Decomposition Algorithm



result := {*R* };
done := false;
compute F^+;
while (not *done)* **do**
  if (there is a schema R_i in *result* that is not in BCNF)
    then begin
      let $\alpha \rightarrow \beta$ be a nontrivial functional dependency that
        holds on R_i such that $\alpha \rightarrow R_i$ is not in F^+,
        and $\alpha \cap \beta = \varnothing$;
      result := (*result* – R_i) $\cup$ (R_i – β) $\cup$ (α, β);
      end
    else *done* := **true;**

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

BCNF may decompose "too much"

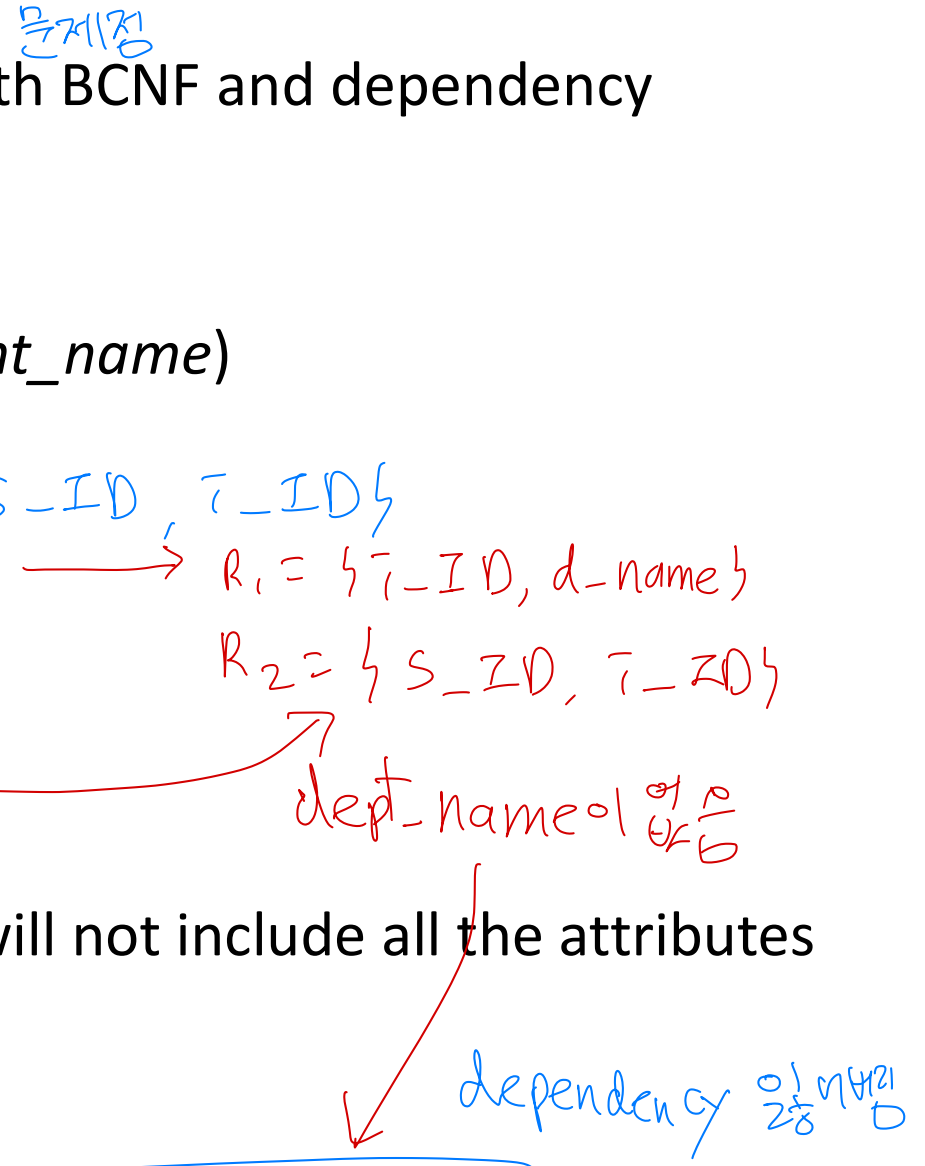
- It is not always possible to achieve both BCNF and dependency preservation
- Consider a schema:

  dept_advisor(s_ID, i_ID, department_name)
- With function dependencies:

  $i\_ID \rightarrow dept\_name$

  $s\_ID, dept\_name \rightarrow i\_ID$
- *dept_advisor* is not in BCNF
  - *i_ID* is not a superkey.
- Any decomposition of *dept_advisor* will not include all the attributes in

  $s\_ID, dept\_name \rightarrow i\_ID$
- Thus, the composition is NOT be dependency preserving
- So, we need a weaker normal form - *Third Normal Form (3NF).*



Third Normal Form (3NF)

- Third Normal Form (3NF)
  - Allows some redundancy 중복성 허용
  - There is always a lossless-join, dependency-preserving decomposition into 3NF.

Third Normal Form (3NF)

- 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., $\beta \in \alpha$)
- α is a superkey for R

} BCNF condition

- Each attribute A in $\beta - \alpha$ is contained in a candidate key.
- (**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.

Third Normal Form (3NF)

- Consider a schema:

  dept_advisor(s_ID, i_ID, dept_name)

- With function dependencies:

  i_ID → *dept_name*

  s_ID, dept_name → *i_ID*

<third condition>
α → β 이때 β = dept_name 중 각각 Attribute
C.K에 존재하면 3NF!
→ super key!

- Two candidate keys = {*s_ID, dept_name*}, {*s_ID, i_ID* }
- We have seen before that *dept_advisor* is not in BCNF
- *R*, however, is in 3NF
  - *s_ID, dept_name* is a superkey
  - *i_ID* → *dept_name and i_ID* is NOT a superkey, but:
    - { *dept_name*} – {*i_ID* } = {*dept_name* }
    - *dept_name* is contained in a candidate key

Redundancy in 3NF

- Consider the schema R below, which is in 3NF
  - *R* = (*name, phone, address*)
  - *F* = {*name phone* → *address*, *address* → *phone* }
  - candidate keys: *name phone*, *name address*
  - And an instance table:

name	phone	address
Alice	123-4567	10 Main St.
Bob	987-6543	20 Oak St.
Alice	555-7890	30 Pine St.
Carol	123-4567	10 Main St.
null	310-3003	40 1st St.

- *name phone* → *address* : *name phone* is a *super key*
- *address* → *phone* : {*phone*} - {*address*} = {*phone*} is contained in a candidate key
- *So, this is in 3NF*

- What is wrong with the table?
  - Repetition of information
  - *name* can be null
  - i.e., *40 1st St.* alone determines *310-3003* while there is no corresponding value for *name*

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**
  if none of the schemas R_j, $1 \le j \le i$ contains $\alpha\ \beta$
    then begin
      $i := i + 1$;
      $R_i := \alpha\ \beta$
    end
if none of the schemas R_j, $1 \le j \le i$ 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 */
    $R_j = R$;;
    $i=i-1$;
return $(R_1, R_2, ..., R_i)$

Read!

Comparison of BCNF and 3NF

- Advantages to 3NF over BCNF.
  - It is always possible to obtain a 3NF design without sacrificing loseless join guarantee 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.

Goals of Normalization

- Benefits of Normalization BCNF, 3NF
  - Less storage space
  - Quicker updates
  - Less data inconsistency
  - Clearer data relationships
  - Easier to add data
  - Flexible Structure

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
- SQL does not provide a way of specifying functional dependencies other than superkeys.
- In SQL, we can not efficiently test a functional dependency whose left hand side is not a key.

보존안될경우 → 3NF 사용

functional dependencies는 디자인페이즈에서 신경쓰자! (SQL에선불가능)

Overall Database Design Process

- We have schema *R*
  - *R* could be a single relation containing *all* attributes
  - Normalization breaks *R* into smaller relations

- When an E-R diagram is carefully designed, the tables should not need further normalization.
- However, in a real design, there can be functional dependencies from non-key attributes to other attributes
  - Example: an *employee* entity with attributes
    department_name and *building*,
    and a functional dependency
    department_name $\rightarrow$ *building*
  - Good design would have made department a separate entity set

Denormalization for Performance

- May want to use non-normalized schema for performance
- Join is the most expensive operation
- 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* (a physical table containing all the tuples in the result of the query) defined as
  $course \bowtie prereq$
  - Benefits and drawbacks same as above, except no extra coding work for programmer and avoids possible errors