11. Normalization

▼ Combine Schemas

연관 있는 relation끼리 table 합치기 가능 → 한 번에 확인 가능

- but, 불필요하게 중복되는 내용 발생 여부 체크 필요

▼ example ⇒ repetition 발생

- *instructor* , *department* ⇒ *inst_dept*로 combine

dept_name : building, budget 내용결정 가능
But, ID, name은 불가
⇒ not candidate
→ decompose 필요

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

inst_dept table

→ 불필요하게 내용 중복
⇒ 발생여부 check해야함.
+) 내부에서 저장하는게 공간도덜 차지할것임.

- **dept_name** : 내용 구분이 어려울 수 있음 cuz not candidate ⇒ decompose

→ how?

▼ example ⇒ repetition 없음

```
sec_class(sec_id, building, room_number) and
section(course_id, sec_id, semester, year)

//combine
section(course_id, sec_id, semester, year,
building, room_number)
```

하나의 relation

referential integrity
: 서로 관계 지어진 것.
functional dependency.
: 결정여부.

▼ Smaller Schemas ⇒ decompose

- **functional dependency** : 특정 attribute가 다른 attribute의 값을 결정 or not

▼ example ⇒ inst_dept table를 decompose 가능?

- dept_name → building, budget 결정 → candidate key여야 함
  - dept_name 하나만 가지고 row 구분 가능
- inst_dept (table) dept_name ⇒ not candidate key (결정 x)
- building, budget ⇒ dept_name에 의존→ repeated 필요

⇒ inst_dept decompose 필요 + dept_name은 반복되어야 함. (not candidate key)

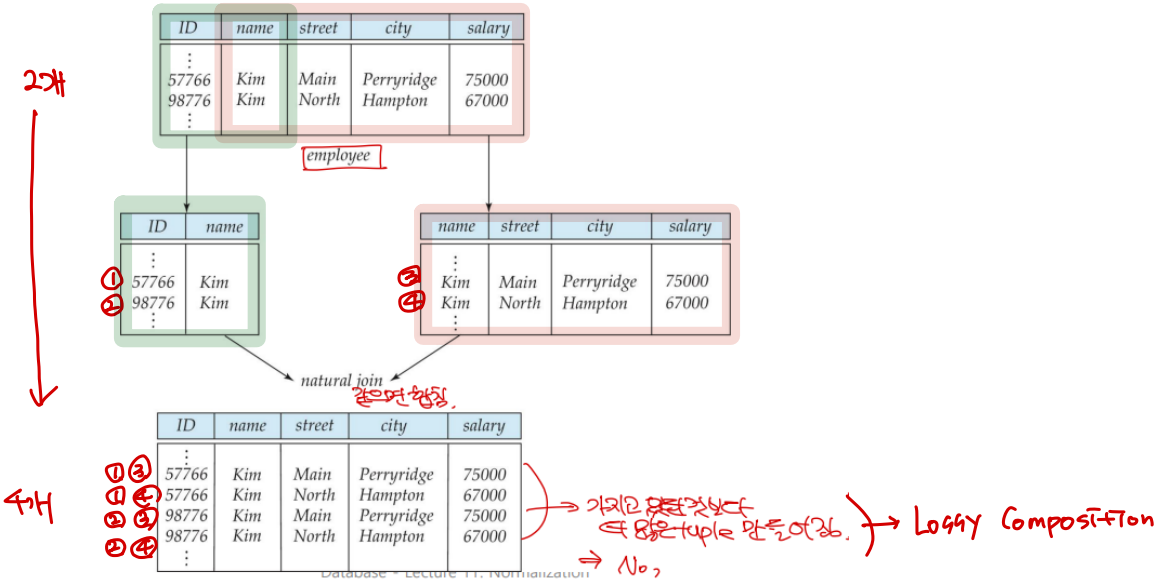
But

- 모든 decomposition이 좋은 것은 아님

```
employee(ID, name, street, city, salary) into
employee1 (ID, name)
employee2 (name, street, city, salary)
```

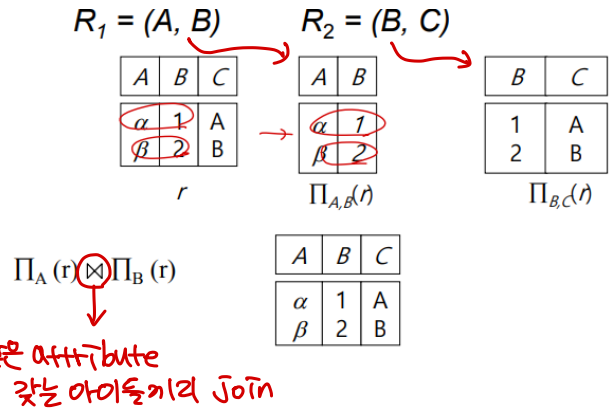
▼ lossy decomposition

- reconstruct 불가 ⇒ 원래 *employee relation* 만들 수 없음



▼ lossless → join decomposition

- decomosition of R = (A, B, C)



▼ Database Normalization

데이터를 일정한 형태와 규칙에 맞게 구조화 → 위와 같은 실수 방지

- Normal Form type

- 1NF : atomic
- 2NF
- 3NF
- BCNF : trivial + superkey

- goal

  - Less storage space : 저장공간↓
  - Quicker updates : update 속도↑
  - Less data inconsistency : 일관성 유지
  - Clearer data relationships : 깔끔한 relationship.
  - Easier to add data : data 추가 쉬움
  - Flexible Structure : 유연한 구조

- design goals

  1. BCNF
  2. lossless join : join 연산 시에 잃는 data x
  3. dependency preservation : 같이 고려해야 함

  ⇒ 3가지 다 고려하고 싶어함.

  → 만약 3가지 다 이루지 못 하면 둘 중 하나 선택

  1. lack of dependenvy preservation
  2. redundancy due to use of 3NF

  ⇒ 잘 design → 분석화 ⇒ database 생성

  - SQL : superkey보다 functional dependency 보장을 제공하지 않음
    - functional dependency 효율적으로 text 어려울 수 있음

▼ functional dependency

$\alpha \subseteq R \text{ and } \beta \subseteq R$ (R이라는 같은 relation 내의 특성 attribute들)

$\alpha \rightarrow \beta$

$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$ (holds on) ★ ⇒ R ↔ r(R)

- example
  - consider r(A,B) → instance of r

A	B
1	4
1	5
3	7

A의 값 → B 결정 x (ex. A=1 ⇒ B=4 or 5)

B의 값 → A 결정 (ex. B=4 ⇒ A=1, B=5 ⇒ A=1)

  - A → B : NOT hold
  - B → A : hold ★ → A의 값이 B를 결정 ⇒ hold한다고 함.

▼ key의 generalization

⇒ fd : key의 notion이 될 수 있음.

super candidate • primary

example
ID, name ⇒ super key
(ID-name) ↛ R ⇒ candidate X

- { K : superkey of R } iff { K → R } ⇒ hold K : tuple 구분가능
- { K : candidate key of R } iff
  - { K → R }
  - { a ⊂ K, no (K - a) → R } → hold X
- example

```
inst_dept(ID, name, salary, dept_name, building, budget)
```

  - dept_name → building (candidate key)
  - ID → building (super key)
  - dept_name ↛ salary

▼ trivial functional dependency

- relation의 모든 instance가 funcitional dependency 만족
- example
  - (ID, name) → ID (α → β, 자기 자신이 바뀔 필요 X)
  - name → name
- a → β is trival ⇒ β ⊆ a → subset (부분집합일 때)

▼ closure of a set of functional dependency

functional dependency : inference rule 사용하는 것으로 구현 가능

- example
  - A → B and B → C ⇒ A → C
  - ID → dept_name
    dept_name → building
    ⇒ ID → building (추론 가능)
- functional dependency F에 의해 명시된 모든 functional dependency 집합 → F의 closure (=F+) ☆
  - closure of F ⇒ F+ (F도 포함)
  - F+ : superset of F
  - F+ → armstrong's axioms

fd theory →

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) ☆

  - sound (실제로 hold하고 있는 functionall dependency만 생성하기) → 하고 있는 것만
  - complete (hold하고 있는 모든 functional dependency 생성하기) → 모두 다

↳ 튜플이 가지는 성질

- 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**)

- example

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

- dependency preservtation : F^+ = F_i들의 set (R$_i$가 득한)
  - fd decomposing ⇒ dependency preserving

$$(F_1 \cup F_2 \cup F_3 \cup \ldots \cup F_n)^+ = F^+ \quad \rightsquigarrow \text{decomposition}$$

  - FD 분해 후 지켜지는지 검사할 때 하나의 relation으로만 판별 가능 (다시복구가능)
  - lossless-join ⇒ dependency-preserving decomposition ⇒ 3NF 충족
    - BCNF, dependency preserving 동시에 얻어내는 것은 항상 가능 x → 3NF 사용 ★

▼ Properties of Functional dependency

1. subset property (Trival) : $Y \subseteq X$, $X \rightarrow Y$ (모든 instance가 만족)
2. augmentation : $X \rightarrow Y$, $XZ \rightarrow YZ$
3. transitivity : $X \rightarrow Y$ and $Y \rightarrow Z$, $X \rightarrow Z$
4. union : $X \rightarrow Y$ and $X \rightarrow Z$, $X \rightarrow YZ$
5. decomposition : $X \rightarrow YZ$, $X \rightarrow Y$ and $X \rightarrow Z$
6. pseudo-transitivity $X \rightarrow Y$ and $WY \rightarrow Z$, $WX \rightarrow Z$

▼ First Normal Form (1NF)

relation schema R ⇒ 모든 attribute에 대한 domain이 atomic이라면 first normal form임 (나눠지지 X)

- 1NF 규칙 : non-atomic values → redundant data(중복 data 생성) ⇒ atomic으로 만들자!

ex) non-atomic domains : Set of names ⇒ composite attributes
(domain은 더 이상 나눠지지 X)

- 모든 relation이 first normal form이라고 가정
  - relation R → good form 만들기
    - R is not 'good' form → decompose {R1, R2, ... , Rn}
    - decomposition → lossless-join decomposition
  - ⇒ funcitional dependency에 따라서 진행

▼ Boyce-Codd Normal Form(BCNF)

F+ → F 집합에 대해 BCNF

$\alpha \subseteq R$ and $\beta \subseteq R$

$\alpha \rightarrow \beta$

1. a → b : trival (모든 function dependency가 superkey)
2. a is a superkey for R

⇒ 두 가지 조건 충족하는 R : BCNF

- example ⇒ 모든 functional dependency가 superkey여야 함

```
instr_dept(ID, name, salary, dept_name, building, budget)
```

  - ID → name
  - name → dept_name
  - ID → salary
  - dept_name → buildng, budget
    - dept_name ≠ instr_dept의 superkey

⇒ BCNF 충족 x → decompose 필요함

▼ decomposing a schema

- example 1
  - 가정 : schema R, non-trival dependency a → b ⇒ BCNF 충족 x

We decompose R into:

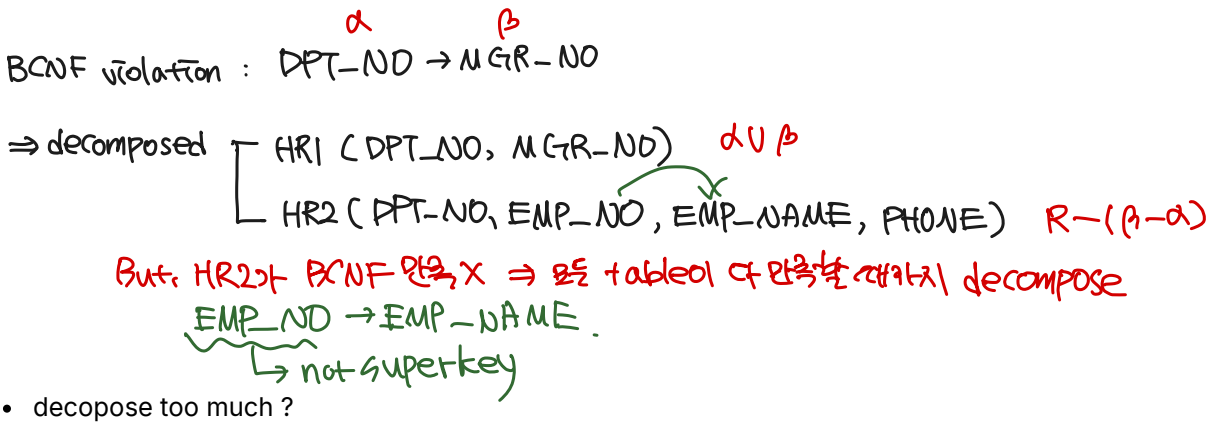
- $(\alpha \cup \beta)$
- $(R - (\beta - \alpha))$

In our example,

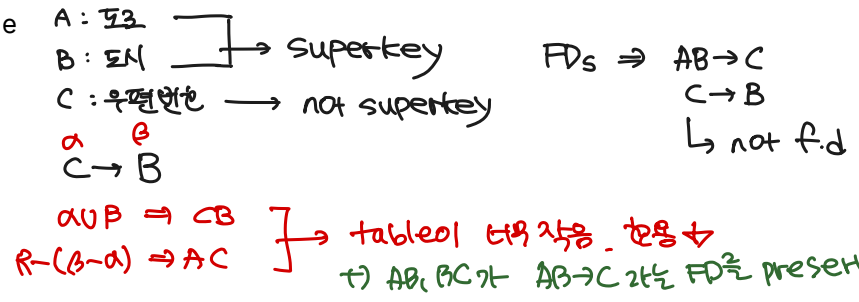
- α = dept_name
- β = building, budget

- example2 : HR(DPT_NO, MGR_NO, EMP_NO, EMP_NAME, PHONE)

  F = DPT_NO → MGR_NO
  DPT_NO → PHONE
  EMP_NO → EMP_NAME



- decopose too much ?
  - 너무 많이 decompose하면 너무 작은 table → BCNF 만족 x
  - example



  - unenforceable FD
    - BCNF, dependency preservation → 모두 만족하는게 항상 가능 x

⇒ 더 널럴한 정규화 형식인 third normal form!

⇒ BCNF의 dependency 두 마리 토끼 다 잡기 어려움
⇒ weaker form (third)

▼ Third Normal Form (3NF)

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

1. a → b is trival
2. a is superkey for R
3. each attribute A in b-a : candidate key of R 포함

약간의 중복 허용

- BCNF 만족 → 3NF 만족
  - BCNF 조건 두 가지 만족함
    - a→b : trivial
    - a : superkey.

☆3번째 조건 : BCNF의 minimal relaxation ⇒ dependcency preservation

▼ motivation

- BCNF : dependency preserving 항상 보장은 x
- update 시에 효율적인 FD violation 확인이 필요함

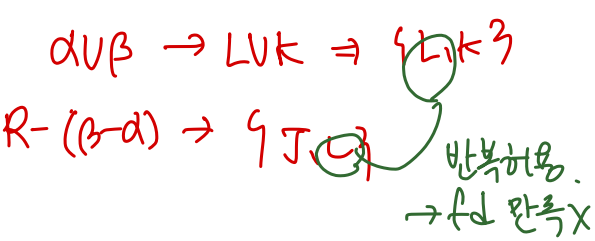
⇒ 보장하는 것에서 테이블 결정

⇒ weak normal form

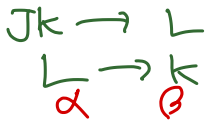
- third normal form
  - 어느 정도 반복 허용 (resultant problem 발생)

        - 작게 나누는 것보다 중복제거 하는 것보단 성능이 떨어짐
    - FD: join 연산 없이 각 relation 확인 가능
    - 3NF의 dependency-preserving decomposition으로 항상 lossless join 가능

▼ Redundancy in 3NF



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



$$R = (J, K, L)$$
$$F = \{JK \rightarrow L, L \rightarrow K\}$$

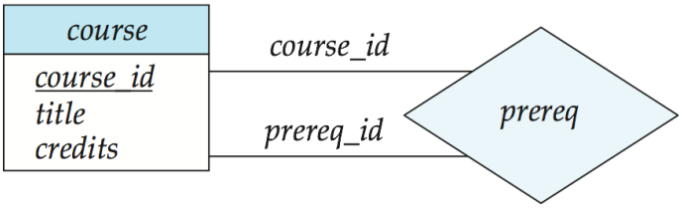
- table의 잘못된 점?
    - information 반복
        - JK : superkey, K : candidate key ⇒ 3NF 보장 → 더이상 쪼개면 x)
    - null value가 필요함 → relationship 나타낼 때 J에 대해 상응하는 값이 없음)

▼ Comparison of BCNF and 3NF

- advantage
    - losslessness sacrificing, dependency preservation 없이 3NF 디자인 항상 가능
- disadvantage
    - data item 사이에 가능한 의미 있는 relationship 중 null value가 필요할 수 있음
    → repetition of information 발생

▼ Overall Database Design process

- schema R ⇒ E-R diagram
    - R : 모든 attribute 포함하는 single relation이 될 수 있음
    - normalization : R을 smaller relation으로 쪼개는 것을 멈추게 함
- E-R diagram이 good design이라면 table은 더 이상 normalization 하지 않아도 됨
- 하지만, 실제 design에서는 non-key attribute에서 fd 충족될 수 있음
    - exmaple
        - employee(department_name, building)
        - fd : department_name → building
        - good design → department를 separate
- denormalization
    - 성능을 위해 non-normalized schema 사용을 원할 수도 있음
    - example



- 대안 1 : attribute를 포함하는 정규화되지 않은 relation과 모든 attribute 포함하는 prereq 사용
  - faster
  - extra space, extra execution time → update 위해 필요함
  - extra coding work, powwibility of error
- 대안 2 : course, prereq로 정의된 materialized view 사용
  - extra coding work와 error 가능성 없음