Relational Design Theory

Dongseop Kwon

Contents

- Basic Concepts
- 1NF: First Normal Form
- FD: Functional Dependencies
- BCNF: Boyce-Codd Normal Form
- 3NF: Third Normal Form
- MVD: Muti-Valued Dependencied
- 4NF: Fourth Normal Form
- Overall Database Design Process

Database Schema 디자인?

- Database Design
  - 어떤 Table 들을 만들 것인가?
  - 각 테이블은 어떤 Attribute를 가질 것인가?

- 문제점
  - 동일한 문제에 대해서 여러 가지 디자인이 가능하다
  - 어떤 것이 더 좋은 디자인인가? 왜 더 좋은 디자인인가?

First Normal Form (1NF)

- Domain is **atomic** if its elements are considered to be indivisible units
  - non-atomic domains 예:
    - Set of names, composite attributes, collections…
    - Identification numbers like CS101 that can be broken up into parts

- ***First Normal Form (1NF)*** if the domains of all attributes are **atomic**.

- We assume all relations are in first normal form.

앨범이름	가수명	곡명
ALONE	씨스타	Come Closer, 나혼자, No Mercy, Lead Me, Girls on Top, …
버스커 버스커 1집	버스커 버스커(Busker Busker)	봄바람, 첫사랑, 여수밤바다, 벚꽃엔딩, …
…	…	…
…	…	…

Database Design Goal

- 나쁜 디자인
  - Inability to represent certain information.
  - Repetition of Information.
  - Loss of information.
  - Anomaly (이상)
    - Update anomaly
    - Deletion anomaly
    - Insertion anomaly

- 좋은 디자인
  - Ensure that relationships among attributes are represented (information content).
  - Avoid **redundant** data.
  - Facilitate enforcement of database **integrity** constraints.

Example

Lending-schema = (*branch-name, branch-city, assets, customer-name, loan-number, amount)*

branch-name	*branch-city*	*assets*	*customer-name*	*loan-number*	*amount*
Downtown	Brooklyn	9000000	Jones	L-17	1000
Redwood	Palo Alto	2100000	Smith	L-23	2000
Perryridge	Horseneck	1700000	Hayes	L-15	1500
Downtown	Brooklyn	9000000	Jackson	L-14	1500

- Redundancy:
  - Data for *branch-name, branch-city,* assets are repeated for each loan that a branch makes.
  - Wastes space.
  - Complicates updating, introducing possibility of inconsistency of *assets* value.
- Null values
  - Cannot store information about a branch if no loans exist.
  - Can use null values, but they are difficult to handle.

Anomaly

- Anomalies (by Codd)
  - Insertion anomaly : loan-number 없이 branch_name등 insert불가.
  - Deletion anomaly : 어떤 branch의 마지막 account delete.
  - Update anomaly : 하나를 update했으면 다른 것도?.

- 원인
  - 정보의 중복(**Redundancy**).
  - 여러 entity가 하나의 table에 합쳐짐.

- 해결책: **Decomposition** !!

Decomposition

- Definition
  - Let R be a relation scheme
  - {R1, …, Rn} is a decomposition of R
  - if R = R1 $\cup$ … $\cup$ Rn (즉, R의 모든 attribute가 R1, … ,Rn에 존재)

- We will deal mostly with binary decomposition:
  R into {R1, R2} where R = R1 $\cup$ R2

Decomposition - Examples

- 학생(학번, 이름, 학과, 학과장, 학과전화, 학년)
  → 학생(학번, 이름, 학년, 학과)
  학과(학과, 학과장, 학과전화)

- Lending = (b_name, b_city, asset, loan#, c_name, amount)
  → Branch = (b_name, b_city, asset)
  Loan = (loan#, c_name, amount)

- ??? 좋은 Decomposition 인가???

Lossy Decomposition

Lending = (b_name, b_city, asset, loan#, c_name, amount)

: anomalies due to repetition of information

Branch = (b_name, b_city, asset), Loan = (loan#, c_name, amount)

- 문제점 : 상관 관계가 없어짐 (연결 정보의 손실).
- Called a "**connection trap**"!
- loss of information.

Branch = (b_name, b_city, asset), Loan = (loan#, c_name, amount, b_city)

- natural join시 tuple은 증가
- but information (relationship) 상실
- loss of information

A Lossy Decomposition

ID	name	street	city	salary
⋮				
57766	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

employee

ID	name
⋮	
57766	Kim
98776	Kim
⋮	

name	street	city	salary
⋮			
Kim	Main	Perryridge	75000
Kim	North	Hampton	67000
⋮			

natural join

ID	name	street	city	salary
⋮				
57766	Kim	Main	Perryridge	75000
57766	Kim	North	Hampton	67000
98776	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

Example of Lossless-Join Decomposition

- **Lossless join 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

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

MYONGJI UNIVERSITY

Lossless-join Decomposition

- Careless decomposition leads to loss of information
  - 잘못된 연결 정보 설정 (Lossy decomposition)
- For $r(R)$ and decomposition $\{R_1, R_2\}$ where $R_1 \cap R_2 \neq \Phi$

$$r \subseteq \Pi_{R1}(r) \bowtie \Pi_{R2}(r),$$

- *Definition:*

  Decomposition $\{R_1, R_2\}$ is a ***lossless-join decomposition*** of R if

$$r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r)$$

- 기준 information은 원래의 information in r.

Goal

- Devise a theory for the following:
- 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 a set of relations $\{R_1, R_2, ..., R_n\}$ such that
  - each relation is in good form.
  - the decomposition is a lossless-join decomposition.
- Our theory is based on:
  - ***functional dependencies***.
  - multivalued dependencies

Functional Dependencies

- $\alpha \rightarrow \beta$
  - α functionally determines β
  - α, β : set of attributes
  - Two tuples with same α have same β
  - **α가 정해지면 반드시 β가 정해진다.**
  - α 값이 같은데 β 값이 다를 수는 없다

- 실제 application의 규칙에 의하여 정해진다.
  - 학번 $\rightarrow$ 이름 ??
  - 이름 $\rightarrow$ 학번 ??
  - 주민등록번호 $\rightarrow$ 학번 ??

- Relation의 모든 instance에서 이 규칙이 성립하여야 함

FD example

branch-name	*branch-city*	*assets*	*customer-name*	*loan-number*	*amount*
Downtown	Brooklyn	9000000	Jones	L-17	1000
Redwood	Palo Alto	2100000	Smith	L-23	2000
Perryridge	Horseneck	1700000	Hayes	L-15	1500
Downtown	Brooklyn	9000000	Jackson	L-14	1500

- *loan-number* $\rightarrow$ *amount*
- *loan-number* $\rightarrow$ *branch-name*
- *loan-number* $\rightarrow$ *customer-name*
- *branch-city* $\rightarrow$ *branch-name*
- *branch-name* $\rightarrow$ *branch-city*
- *branch-name* $\rightarrow$ *assets*
- *loan-number* $\rightarrow$ *customer-name*

Trivial FD

- $\alpha \rightarrow \beta$ is **trivial** if $\beta \subseteq \alpha$
- **당연히!** 성립할 수 밖에 없는 FD
- 예
  - {이름, 나이} $\rightarrow$ {이름}
  - {나이} $\rightarrow$ {나이}

Closure of a Set of FDs

- **Closure**: 주어진 집합으로 유추할 수 있는 모든 원소의 집합

- ***F***$^+$: closure of *F* (**set of FDs**)
  - **주어진 FD 집합에 의해 성립하는 모든 FD의 집합**
  - 몇 가지 자동으로 성립하는 FD의 예 (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)**
    - If $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$, then $\alpha \rightarrow \beta\,\gamma$ **(union)**
    - If $\alpha \rightarrow \beta\,\gamma$, then $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$ **(decomposition)**
    - If $\alpha \rightarrow \beta$ and $\gamma\,\beta \rightarrow \delta$, then $\alpha\,\gamma \rightarrow \delta$ **(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 $CG \rightarrow I$.
  - $CG \rightarrow HI$
    - from $CG \rightarrow H$ *and* $CG \rightarrow I$: **"union rule"** can be inferred from definition of functional dependencies, or Augmentation of $CG \rightarrow I$ to infer $CG \rightarrow$ CGI, augmentation of $CG \rightarrow H$ to infer $CGI \rightarrow HI$, and then transitivity

MYONGJI UNIVERSITY

Attribute Set Closure

- α^+: closure of α (**attribute sets**) under *F*
  - the set of attributes that are functionally determined by α under *F*
  - **주어진 애트리뷰트 집합에 의해 결정되는 모든 애트리뷰트 집합**
  - 용도
    - Super Key 테스트
    - FD 테스트
    - F^+ 계산

Example of Attribute Set Closure

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$
- $(AG)^+$
  1. $result = AG$
  2. $result = ABCG$ $(A \rightarrow C$ and $A \rightarrow B)$
  3. $result = ABCGH$ $(CG \rightarrow H$ and $CG \subseteq AGBC)$
  4. $result = ABCGHI$ $(CG \rightarrow I$ and $CG \subseteq AGBCH)$
- Is AG a candidate key?
  1. Is AG a superkey?
     1. Does $AG \rightarrow R$? == Is $(AG)^+ \supseteq R$
  2. Is any subset of AG a superkey?
     1. Does $A \rightarrow R$? == Is $(A)^+ \supseteq R$
     2. Does $G \rightarrow R$? == Is $(G)^+ \supseteq R$

Cover

- **Cover**:
  - F와 동일한 Closure를 갖는 FD 집합
  - A cover of F is any F' such that $F'^{+} = F^{+}$
  - 예) {A→B, A→AC, A→BC}와 {A→B, A→C}
- **Redundant** FD:
  - A FD g ∈ F is **redundant** if $(F - \{g\})^{+} = F^{+}$ or $g \in (F - \{g\})^{+}$
  - 없어도 되는 FD
- **Minimal cover**:
  - F' is a nonredundant (minimal) cover of F if
    $F'^{+} = F^{+}$ and
    F' contains no redundant FD
  - 주어진 FD집합과 동일한 최소의 FD집합 (redundant한 FD 제거)

MYONGJI UNIVERSITY

Extraneous Attributes

- **Extraneous** Attribute: 없어져도 **무관한** 애트리뷰트
- 정의: $\alpha \rightarrow \beta$ in F.
  - Attribute A is **extraneous** in α if $A \in \alpha$ and
    F logically implies $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$.
  - Attribute A is **extraneous** in β 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
- Example: Given $F = \{A \rightarrow C, AB \rightarrow C\}$
  - B is extraneous in $AB \rightarrow C$ because $\{A \rightarrow C, AB \rightarrow C\}$ logically implies $A \rightarrow C$ (i.e. the result of dropping B from $AB \rightarrow C$).
- Example: Given $F = \{A \rightarrow C, AB \rightarrow CD\}$
  - C is extraneous in $AB \rightarrow CD$ since $AB \rightarrow CD$ can be inferred even after deleting C

Canonical Cover

- A **canonical cover** for F is a set of dependencies Fc such that
  - Fc+ = F+
  - No FD in Fc contains an extraneous attribute
  - Each left side of a FD in Fc is unique

  - 불필요한 attribute는 없고,
  - FD의 왼쪽 편이 두번 나오지 않도록 한
  - cover
- 주어진 FD 집합과 동일한 결과를 표현할 수 있는 가장 simple한 FD의 집합

Example: Canonical Cover

- $R = (A, B, C), F = \{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$
- Combine $A \rightarrow BC$ and $A \rightarrow B$ into $A \rightarrow BC$
  - Set is now $\{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$
- A is extraneous in $AB \rightarrow C$
  - 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 $\{A \rightarrow BC, B \rightarrow C\}$
- 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: $\{A \rightarrow B, B \rightarrow C\}$

Lossless-join Decomposition Revisited

- For the case of $R = (R_1, R_2)$, we require that for all possible relations r on schema R

$$r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r)$$

- A decomposition of R into R_1 and R_2 is lossless join if at least one of the following dependencies is in F^+:
  - $\boldsymbol{R_1 \cap R_2 \rightarrow R_1}$
  - $\boldsymbol{R_1 \cap R_2 \rightarrow R_2}$
- 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

Example

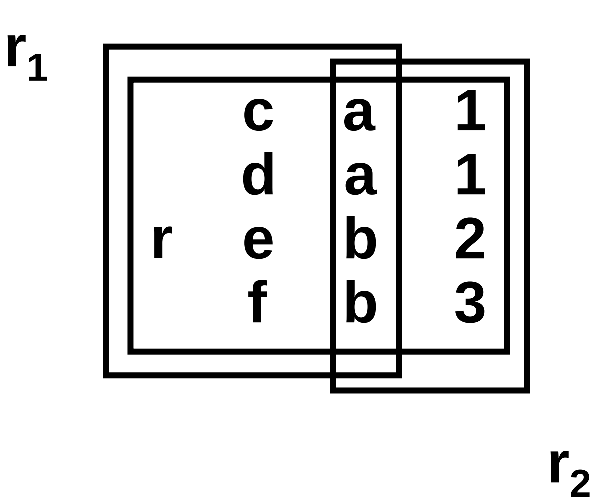
- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C\}$
  - Can be decomposed in two different ways
- $R_1 = (A, B), \quad R_2 = (B, C)$
  - Lossless-join decomposition: $\boldsymbol{R_1 \cap R_2 = \{B\}}$ **and** $\boldsymbol{B \rightarrow BC}$
  - Dependency preserving
- $R_1 = (A, B), \quad R_2 = (A, C)$
  - Lossless-join decomposition: $\boldsymbol{R_1 \cap R_2 = \{A\}}$ **and** $\boldsymbol{A \rightarrow AB}$
  - Not **dependency preserving**
    (cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

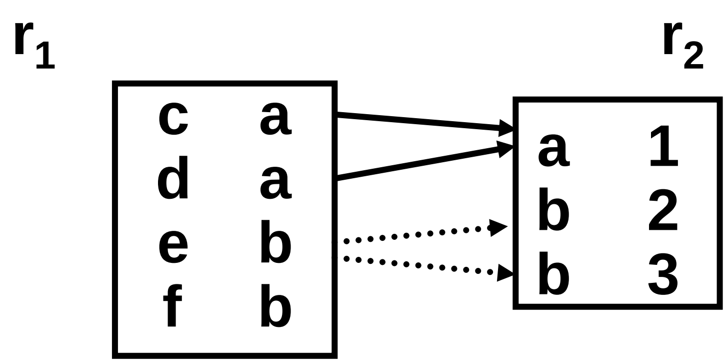
Lossless-join Decomposition Revisited (cont.)

- $\{R_1, R_2\}$ is a lossless decomposition if

  1) $R_1 \cap R_2 \rightarrow R_1$, or 2) $R_1 \cap R_2 \rightarrow R_2$

  – 즉, 분해한 두 개의 schema중 하나가 다른 하나의 superkey를 포함하면 연관 관계의 손실이 없다.





MYONGJI UNIVERSITY

Dependency Preservation

이름	시	도
홍길동	인천	경기

F = {이름 → 시, 도; 시 → 도}

이름	시
홍길동	인천

시	도
인천	경기

- 무손실 분해
- 의존성 보존됨
  - "시"이름이 같은데 "도"가 다른것이 있나? 한번에 확인 가능

이름	시
홍길동	인천

이름	도
홍길동	경기

- 무손실 분해
- **의존성이 보존되지 않음!**
  - "시"이름이 같은데 "도"가 다른것이 있나? 확인 하기 위해서는 Join이 필수!

MYONGJI UNIVERSITY

Goal for decomposition

- When we decompose a relation schema *R* with a set of functional dependencies *F* into R_1, R_2,.., R_n we want

1. **Lossless decomposition**
2. **No redundancy**
3. **Dependency preservation**

Boyce-Codd Normal Form

- A relation schema R is in BCNF (with respect to a set F of FDs) if for each FD $\alpha \rightarrow \beta$ in F+ ($\alpha \subseteq R$ and $\beta \subseteq R$), at least one of the following holds:
  - **$\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)**
  - **α is a superkey for R.**
  - **Trivial 하지 않은 모든 함수종속에서 결정자가 key인 경우 BCNF**

- Example schema not in BCNF:
  - instr_dept (ID, name, salary, dept, building, budget)
  - 이유
    - dept$\rightarrow$ building, budget 에서 dept이 key가 아님.

Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B ; \ B \rightarrow C\}$
  Key $= \{A\}$
- R is not in BCNF!
- Decompose into $R_1 = (A, B)$, $R_2 = (B, C)$
  - R_1 and R_2 in BCNF
  - Lossless-join decomposition
  - Dependency preserving

Testing for BCNF

- BCNF를 test할 때는 F만 고려해도 충분
- 분해를 할 때는 F^+를 모두 고려해야 함!

- Example

  Consider R ($\underline{A}$, B, C, D) with F = { $A \rightarrow B$, $B \rightarrow C$}
  - Decompose into $R_1(A, B)$ and $R_2(A, C, D)$
  - Neither of the dependencies in F contain only attributes from (A, C, D) so we might be mislead into thinking R_2 satisfies BCNF.
  - In fact, dependency $A \rightarrow C$ in F^+ shows R_2 is not in BCNF.

MYONGJI UNIVERSITY

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 $\alpha \cap \beta = \emptyset$;
      result := $(result - R_i) \cup (R_i - \beta) \cup (\alpha, \beta)$;
    end
    else *done* := **true;**

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

Example of BCNF Decomposition

- Let's look the same example one more time!
- $R = (A, B, C)$
  $F = \{A \rightarrow B, \ B \rightarrow C\}$
  Key = $\{A\}$
- R is not in BCNF ($B \rightarrow C$ but B is not superkey!)
- Decomposition
  - $R_1 = (B, C)$
  - $R_2 = (A, B)$

Example of BCNF Decomposition

- *class* (*c_id, title, dept, credits, sec_id, semester, year, building, room, capacity, time_slot_id*)

- FD:
  - *c_id* → *title, dept, credits*
  - *building, room* → *capacity*
  - *c_id, sec_id, semester, year* → *building, room, time_slot_id*
- A candidate key {*c_id, sec_id, semester, year*}

Example of BCNF Decomposition

- *class는 BCNF인가?*
  - *no:* ***c_id*** *→* ***title, dept, credits***
  - Decomposition
    - *course* (*c_id, title, dept, credits*)
    - *class-1 (c_id, sec_id, semester, year, building, room, capacity, time_slot_id)*
- *course는 BCNF인가?*
  - *yes (c_id가 superkey임)*
- *class-1은 BCNF인가?*
  - *no:* ***building, room→capacity***
  - Decomposition
    - *classroom* (*building, room, capacity*)
    - *section* (*c_id, sec_id, semester, year, building, room, time_slot_id*)

Example

- R = (*b-name, b-city, assets, c-name, loan-n, amount*)
  F = {*b-name* $\rightarrow$ *assets b-city* , *loan-n* $\rightarrow$ *amount b-name*}
  Key = {*loan-n, c-name*}

- Decomposition
  - R_1 = (*b-name, b-city, assets*)
  - R_2 = (*b-name, c-name, loan-n, amount*)
  - R_3 = (*b-name, loan-n, amount*)
  - R_4 = (*c-name, loan-n*)

- Final decomposition result: R_1, R_3, R_4.

BCNF and Dependency Preservation

- $R = (J, K, L)$
  $F = \{JK \rightarrow L, \quad L \rightarrow K\}$
  Two candidate keys = JK and JL
- R is not in BCNF
- Any decomposition of R will fail to preserve

$$JK \rightarrow L$$

  This implies that testing for $JK \rightarrow L$ requires a join

BCNF 분해가 의존성을 보존하지 않을 수 도 있음!

→ BCNF보다 약한 정규형이 필요함

Third Normal Form (3NF)

- A relation schema R is in third normal form (3NF) if for all $\alpha \rightarrow \beta$ 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
  - **Each attribute A in $\beta - \alpha$ is contained in a candidate key**
    (NOTE: each attribute may be in a different candidate key)
- **BCNF를 만족하지 않는 FD?**
  피결정자의 애트리뷰트는 Candidate Key의 일부분
- BCNF는 항상 3NF에 속함

3NF Example

- Relation ***dept_advisor***: (학과-지도교수)
  - *dept_advisor* (*s_ID*, *i_ID*, *dept*)
    $F = \{s\_ID, dept \rightarrow i\_ID,\ \ i\_ID \rightarrow dept\}$
    - *① 교수는 하나의 전공에만 소속된다.*
    - *② 학생은 전공별로 한 명의 지도교수를 갖는다.*
    - *③ 학생은 복수 전공이 가능함.*
  - Two candidate keys: *s_ID*, *dept*, and *i_ID*, *s_ID*
  - *R* is in 3NF
    - $s\_ID, dept \rightarrow i\_ID$ (*s_ID*, *dept* is a superkey)
    - $i\_ID \rightarrow dept$ (*dept* is contained in a candidate key)
  - BCNF는 성립하지 않음 ($i\_ID \rightarrow dept$)
  - BCNF 분해를 할 경우 ②를 검사하는 것이 불가능함

3NF Decomposition Algorithm

Let F_c be a canonical cover for F;
i := 0;
for each fd $\alpha \rightarrow \beta$ in F_c **do**
  if none of R_j, $1 \leq j \leq i$ contains $\alpha\ \beta$ **then**
    { $i := i + 1$
     $R_i := \alpha\ \beta$ }
if none of R_j, $1 \leq j \leq i$ contains a candidate key for R
  then { $i := i + 1$;
     R_i := any candidate key for R }
return $(R_1, R_2, ..., R_i)$

/* 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 $(R_1, R_2, ..., R_i)$

① Canonical Cover를 계산

② Canonical Cover의 모든 FD를 Relation 생성

③ Candidate Key를 포함한 Relation이 없으면 Candidate 키로 Relation을 하나 추가

④ 중복된 Relation 제거

MYONGJI UNIVERSITY

Example

- Banker = (branch-name, customer-name, banker-name, office-num)
  - Fc: banker-name → branch-name, office-num
    customer-name, branch-name → banker-name
  - The key: {customer-name, branch-name}
- 3NF?
  - NO: banker-name → branch-name, office-num
  - Decomposition
    - {banker-name,branch-name,office-num}
    - {<u>customer-name</u>, branch-name, <u>banker-name</u>}
  - Candidate 키가 있으므로 분해 완료!

Comparison of BCNF and 3NF

- 3NF
  - Lossless & Dependency Preserving Decomposition 가능
  - Data Redundancy가 존재 → Anomaly 가능

- BCNF
  - Lossless Decomposition 가능
  - Dependency Preserving 불가능할 수 있음
  - Data Redundancy 없음

Example: MVD(Multi-Valued Dependency)

- User(id, phone, email)
  - 사용자는 여러 개의 전화번호를 가질 수 있다.
  - 사용자는 여러 개의 이메일을 가질 수 있다.
- FD? 없음
- Key? (id, phone, email)
- BCNF? YES
- Good Design? NO
  - 전화번호 2개, 이메일 3개를 가진 사용자 1명의 Tuple만 6개

MVD(Multi-Valued Dependency)

- $\alpha \twoheadrightarrow \beta$
  - α 가 정해지면 β 의 값에 대한 나머지 부분이 항상 동일해야 함
  - β의 값이 여러 개일 수는 있지만, 각 β에 대해 동일한 rest 집합
- 예
  - id ↠ email
    - 이메일을 여러 개(A, B, C) 가질 수 있지만, (id, emailA)에 대해 나왔던 나머지 애트리뷰트 값들이 (id, emailB), (id, emailC)에 대해서도 나와야 함.
- FD도 MVD의 일종
  - $\alpha \rightarrow \beta$이면 $\alpha \twoheadrightarrow \beta$
  - 예) id에 의해 name이 결정된다는 것은 name값이 하나밖에 없으므로 rest 부분은 저절로 동일해짐.

Fourth Normal Form (4NF)

- A relation schema R is in **4NF**(with respect to a set D of MVDs) if for each FD $\alpha \twoheadrightarrow \beta$ in D+ $(\alpha \subseteq R$ and $\beta \subseteq R)$, at least one of the following holds:
  - **$\alpha \twoheadrightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)**
  - **α is a superkey for R.**
  - **Trivial 하지 않은 모든 MVD에서 결정자가 key인 경우 4NF**
- **BCNF를 MVD로 확장한 것**
- $\alpha \rightarrow \beta$이면 $\alpha \twoheadrightarrow \beta$이므로 MVD를 모두 고려하면 자동으로 FD도 고려된 것임. **그러므로 모든 4NF는 BCNF임.**
- Decomposition 방법도 BCNF와 동일

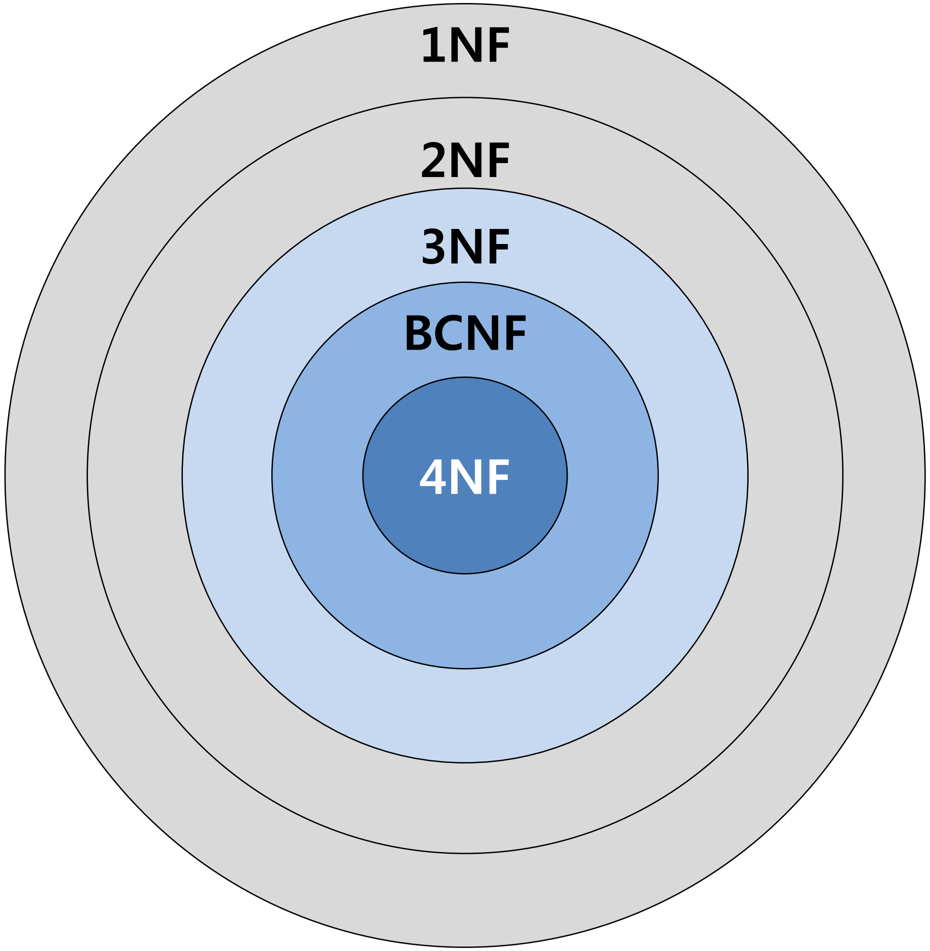
Example 1

- User(id, phone, email)
  - MVD
    - id $\twoheadrightarrow$ phone , id $\twoheadrightarrow$ email
  - Decomposition
    - (id, phone), (id, email)

Example 2

- Contact(comp, dept, manager, salary, area_code, phone)
  - 제약사항
    - 회사는 하나 이상의 부서를 갖는다.
    - 부서는 여러 개의 전화번호를 가질 수 있다.
    - 부서별로 한 명의 책임자가 존재할 수 있다.
    - 전화번호는 지역번호와 나머지번호로 구성된다.
    - 한 명의 책임자가 여러 부서를 담당할 수도 있다.
    - 책임자의 연봉은 책임자별로 정해진다.
  - MVD & FD
    - comp, dept↠ area_code, phone
    - comp, dept→manager
    - manager→salary
  - Key? 없음
  - Decomposition
    - (company, dept, area_code, phone)
    - (company, dept, manager, salary)
      → (company, dept, manager) (manager, salary)

Normal Forms



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

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.

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 FDs from non-key attributes of an entity to other attributes of the entity.

  e.g., *employee* entity with attributes *department-number* and *department-address*, and an FD *department-number* $\rightarrow$ *department-address.*
  - Good design would have made department an entity.
- FDs from non-key attributes of a relationship set (involving more than two entity sets) possible, but rare --- most relationships are binary.

Denormalization for Performance

- May want to use non-normalized schema for performance.
  - 예) displaying customer-name along with account-number and balance requires join of account with depositor.
- Alternative 1: Use **denormalized** relation containing attributes of account as well as depositor.
  - 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 as $A \bowtie D$
  - Benefits and drawbacks same as above
  - except no extra coding work for programmer (done by DBMS)
  - 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-2000, earnings-2001, earnings-2002, 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-2000, earnings-2001, earnings-2002)
    - 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

Acknowledgments

- Most of the slides and examples are provided by the authors of the book “Database Systems Concepts,“ which were modified into their current form.