Ch07 Normalization 정규화

- Binary Decomposition이 loseless 하지 않으려면, 다음 중 최소 하나 성립해야함.
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$
- loseless: $\Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) = r$ | lossy: $r \subset \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$

Normalization Theory

< Functional dependencies
Multivalued dependencies

04/16

Functional Dependencies Definition

$\alpha \subseteq R$이고 $\beta \subseteq R$일때, $\alpha \rightarrow \beta$
$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$

A	B
1	4
1	5
3	7

B→C는 성립
A→B는 성립 x

Closure of functional dependencies

F+

Keys & Functional dependencies

K는 superkey iff K→R
K는 candidate key iff K→R & for no $\alpha \subset K$, $\alpha \rightarrow R$ (minimal 해야한다)

Use of functional dependencies

1. relation의 instance가 주어진 함수 종속 집합 F를 만족하는지 검사 : r satisfies F

2. 적법한 릴레이션의 집합에 대한 제약조건 명시 : F는 r(R)을 모든(성립,유지,만족) : F holds on R

   satisfy는 특정 instance가 fd를 만족하는 것 / hold는 schema 자체가 legal

Trivial Functional Dependencies

trivial : 일반적인 $\alpha \rightarrow \beta$형태의 함수 종속에서 $\beta \subseteq \alpha$ 일때.

ex) name → name
ID,name → ID

Loseless Decomposition

→ 함수종속 사용해 loseless인지 증명
→ $R = (R_1, R_2)$ 일때
다음과 같은 dependency를 적어도 하나 F^+가 가지고 있다면 loseless
$R_1 \cap R_2 \rightarrow R_1$ 또는 $R_1 \cap R_2 \rightarrow R_2$ → 즉 $R_1 \cap R_2$가 R_1 또는 R_2에 대한 superkey이면 loseless decomposition

loseless decomposition에 대해 sufficient condition (충분조건) 이지만,
모든 constraint가 functional dependency 일때만 necessary condition (필요조건) 이 된다. → 함수적 의존성이 없어도 multivalue dependency로 loseless decomposition이 가능하다?
1) 만약 $(R_1 \cap R_2 \rightarrow R_1) \vee (R_1 \cap R_2 \rightarrow R_2)$이면 loseless decomposition이다(True).
2) R을 R_1, R_2로 나눴을 때, $(R_1 \cap R_2 \rightarrow R_1) \vee (R_1 \cap R_2 \rightarrow R_2)$이다. (False).

ex) $R = (A, B, C)$
$F = \{A \rightarrow B, B \rightarrow C\}$ $F^+ = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$

① $R_1 = (A, B)$, $R_2 = (B, C)$ loseless? → oo.
$R_1 \cap R_2 = B$ $B \rightarrow BC$에서.

*주의) $B \rightarrow BC$는 $B \rightarrow \{B, C\}$의 간소화된 표기.

② $R_1 = (A, B)$ $R_2 = (A, C)$ loseless? → oo.
$R_1 \cap R_2 = A$ $A \rightarrow AB$.

Dependency Preservation

→ dependency constraints가 지켜졌는지 아닌지를 매번 join 해야하지, 아니면 바로 알 수 있는지 여부.
→ db가 업뎃될때마다 functional dependency 테스트하려면 costly 할수도 o

ex) dept_advisor (s_ID, i_ID, dept_name)
i_ID → dept_name
s_ID, dept_name → i_ID

$R1 = \{s\_ID, i\_ID\}$
$R2 = \{i\_ID, dept\_name\}$ → loseless하지만 dependency preserving x.

어떻게 분리해도. not dependency preserving.

04/16

04/21

Normal Form (정규형)

1) BCNF

F가 주어졌을 때, F^+가 다음 중 둘 중 하나만 만족해도 BCNF 만족

$\alpha \rightarrow \beta$ 일 때
- $\alpha \rightarrow \beta$ 가 trivial ($\beta \subseteq \alpha$)
- α가 R의 superkey

ex) BCNF 가 아닌 schema

In_dep (ID, name, salary, dept_name, building, budget)

왜?

dept_name → building, budget / 근데, dept_name은 superkey가 아님.

분해해볼까?

Instructor랑 department

Instructor (ID, name, salary)

department (dept_name, building, budget)

분해 방법

$\alpha \rightarrow \beta$ 가 위반하는 FD

아까전 예시

$\alpha \Rightarrow$ dept_name

$\beta \Rightarrow$ building, budget → $\alpha \cup \beta$

$R-(\beta-\alpha)$

Instructor (ID, name, salary)

department (dept_name, building, budget)

$\alpha \cup \beta$

$R-(\beta-\alpha)$

∴ 내 생각 정리)

하나는	$\alpha+\beta$
다른건	$R-\beta+\alpha$

ex)

$R=(A,B,C)$

$F=\{A \rightarrow B, B \rightarrow C\}$, $F^+ \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$

① $R_1=(A,B)$, $R_2=(B,C)$
- Loseless? ○○
- $R_1 \cap R_2 = \{B\}$ and $B \rightarrow BC$
- Dependency preserving ○○

② $R_1=(A,B)$, $R_2=(A,C)$
- Loseless? ○○
- $R_1 \cap R_2 = \{A\}$ and $A \rightarrow AB$
- Dependency Preserving ㄴㄴ

※ BCNF와 Dependency Preservation을 동시에 성취하기 어렵다.

ex) dept_advisor (s_ID, i_ID, dept_name)

i_ID → dept_name

s_ID, dept_name → i_ID

i_ID는 superkey가 아니므로 decomposition 해야한다.

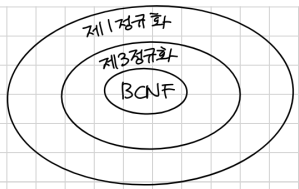
R1 (i_ID, dept_name)

R2 (s_ID, i_ID)

→ motivation for 3rd normal form

안되네..

04/30



2) Third Normal Form

BCNF : 모든 비자명한 (non-trivial) 종속은 $\alpha \rightarrow \beta$ 형태, α는 superkey 여야함.

↕

3NF : 〃 에 대하여, α가 superkey 가 아니어도 허용

F^+내 $\alpha \rightarrow \beta$ 형태 모든 함수 종속이 다음을 적어도 하나를 만족해야함.

- $\alpha \rightarrow \beta$ is trivial
- α가 R의 superkey
- $\beta-\alpha$에 있는 모든 attribute가 R의 candidate key에 포함된다. → 각 attribute가 다른 candidate key 안에 있어도 무관.

BCNF → 3NF (○) / 3NF → BCNF (×)

ex) dept_advisor (s_ID, i_ID, dept_name)

s_ID, dept_name → i_ID

i_ID → dept_name

{s_ID, dept_name}, {s_ID, i_ID} ⇒ candidate_key.

근데 3NF이다.

s_ID, dept_name은 superkey

{dept_name} - {i_ID} = {dept_name} → candidate key에 포함됨.

Redundancy in 3NF

R = (J, K, L)

F = {JK → L, L → K}

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

잘못된점) • 정보의 반복
• null value 사용

3NF 장점) dependency preservation

단점) null values

정보의 중복

Closure of a Set of Functional Dependencies

Armstrong's Axioms

① Reflexive Rule: if $\beta \subseteq \alpha$ then $\alpha \rightarrow \beta$
② Augmentation Rule: if $\alpha \rightarrow \beta$ then $\gamma\alpha \rightarrow \gamma\beta$
③ Transitivity Rule: if $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$

example of F^+

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

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

Some members of F^+
- $A \rightarrow H$ 왜? transitivity rule $A \rightarrow B, B \rightarrow H$
- $AG \rightarrow I$ 왜? $AG \rightarrow CG$, $CG \rightarrow I$ augmentation, transitivity
- $CG \rightarrow HI$ 왜? $CG \rightarrow I$, $CG \rightarrow CGI$, $CGI \rightarrow HI$

Additional Rules
- Union Rule: $\alpha \rightarrow \beta$ 이고, $\alpha \rightarrow \gamma$ 이면, $\alpha \rightarrow \beta\gamma$
  $\alpha\gamma \rightarrow \beta\gamma$, $\alpha \rightarrow \alpha\gamma$ → $\alpha \rightarrow \beta\gamma$
- Decomposition rule: $\alpha \rightarrow \beta\gamma$ 이라면 $\alpha \rightarrow \gamma$, $\alpha \rightarrow \beta$
- Pseudotransitivity rule: $\alpha \rightarrow \beta$ 이고 $\gamma\beta \rightarrow \delta$ 이면 $\alpha\gamma \rightarrow \delta$

Procedures for Computing F^+

$2^n \times 2^n = 2^{2n}$ 가능?

① 초기화: $F^+ = F$ (원래 함수 종속으로 시작)
② 반복 : Step1) F^+의 각 함수 종속에 대해,
reflexivity rule과 augmentation rule 적용해 추가
Step2) transitivity rule 적용
변하지 않을 때까지 반복.

04/30

05/07

Closure of Attribute Sets

closure of α under F를 α^+로 표시.

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

우선 자기자신을 result에 넣고
$\beta \rightarrow \gamma$ 에 대하여 β가 result의 부분집합이라면
γ을 result에 넣고 반복.

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

$(AG)^+$
1) result = AG
2) result = ABCG (왜? $A \rightarrow B$, $A \rightarrow C$)
3) result = ABCGHI (왜? $CG \rightarrow H$, $CG \rightarrow I$, $CG \subseteq ABCG$)

↓

2개로 AG가 candidate key임?
① AG가 superkey임? ㅇㅇ.
② AG의 subset 중 superkey가 존재?
$A \rightarrow R$? $A^+ = ABCH$ → superkey X
$G \rightarrow R$? $G^+ = G$.

∴ 일반화: n-1크기의 부분집합에 대해 test.

Use of Attribute Closure

① Testing for Superkey
→ α가 superkey라면 α^+는 R의 모든 attributes 가지고 있어야 함.

② Testing functional dependencies
→ $\alpha \rightarrow \beta$가 hold하는지 (다른말로 F^+에 있는지) 확인하려면
$\beta \subseteq \alpha^+$인지 확인해보면 된다.

③ Computing closure of F
→ 각 γ ($\gamma \subseteq R$)에 대하여, γ^+을 계산
$S \subseteq \gamma^+$에 대해 $\gamma \rightarrow S$를 output으로 내보낸다.

Extraneous Attribute

: 없어도 무관한 attribute → 없애서 canonical cover 도출

① 왼쪽에서 없애면 더 stronger

AB→C를 A→C로 줄이기.

왜? A→C는 logically AB→C 임서.

ex) F = { AB→C, A→D, D→C }

A→C 이기에 B 제거가능

② 오른쪽에서 제거하면 weaker

AB→CD에서 AB→C로 줄이면 약해짐.

AB가 있어도 D를 알수X.

ex) F = {AB→CD, A→C}

AB→CD를 AB→D로 바꿔해도 AB→C 추론가능.

F^+를 안 바꾸고 언제 뺄수 있는지. (F에 있는 $\alpha \to \beta$에 대하여)

1 $A \in \alpha$ 그리고

F가 $(F - \{\alpha \to \beta\}) \cup \{(\alpha - A) \to \beta\}$를 logically imply 할때.

2 $A \in \beta$ 그리고

$(F - \{\alpha \to \beta\}) \cup \{\alpha \to (\beta - A)\}$가 F를 logically imply 할때.

Testing Extraneous

① 왼쪽에서 제거.

- $\gamma = \alpha - \{A\}$로 설정하고
- $\gamma \to \beta$가 F에서 추론가능한지 확인
  - ↳ F에서 γ^+ 계산.
    γ^+가 β를 포함하면, A가 extraneous

∴ 기존 F에서 (왼쪽 attribute 제거한 γ^+)계산

β 포함 여부 확인

★

② 오른쪽에서 제거.

- $F' = (F - \{\alpha \to \beta\} \cup \{\alpha \to (\beta - A)\}$

  α^+가 F'에서 A 포함하는지 확인

∴ F'에서 α^+계산, A포함하는지.

포함하면 extraneous

↓

extraneous 속성 제거.

ex) F = {AB→CD, A→E, E→C}

C가 AB→CD에서 extraneous 인지확인.

$F' = \{AB \to D, A \to E, E \to C\}$

$AB^+ = \{AB, D, E, C\}$

C를 포함 ⇒ extraneous하다.

Canonical Cover

① F 는 F^+와 논리적으로 동치?

② F_c는 F 와 〃 ?

③ F_c는 extraneous attribute X

④ 좌변은 유일

ex) $\alpha_1 \to \beta_1$, $\alpha_2 \to \beta_2$ 일때 $\alpha_1 = \alpha_2$인 경우 X

과정

1) union rule 적용

같은 좌변이면 합치기. $\alpha_1 \to \beta_1$, $\alpha_1 \to \beta_2$ 를 $\alpha_1 \to \beta_1\beta_2$로.

2) extraneous attribute 제거.

ex 01) Computing a Canonical Cover

R = (A,B,C)

F = {A→BC, B→C, A→B, AB→C}

1) A→BC 와 A→B 합쳐.

A→BC, B→C, AB→C

2) AB→C에서 A가 extraneous? ○○

B→C. 이미 있네! ⇒ {A→BC, B→C}

3) A→BC에서 C가 extraneous?

F' = {A→B, B→C}

compute $A^+ = AC$, 어! 여기에 C가 포함되네.

extraneous !!

F_c = {A→B, B→C}

ex 02) Computing a Canonical Cover

R = {A,B,C,D}

F = {A→BC, B→C, A→B, AB→C, AC→D}

{A→BC, B→C, AB→C, AC→D}

A→BC에서 C가 extraneous?

F {A→B, B→C, AB→C, AC→D}

Yes. A→C 추론가능.

AB→C에서 C는 extraneous, 그래서 삭제.

F {A→B, B→C, AC→D}

AC→D에서 C가 extraneous?

A^+ = {A B C D}

∴ F_c = {A→B, B→C, A→D}

F_c = {A→BD, B→C}

Dependency Preservation

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

예상에

```
result = α
repeat
  for each Ri in the decomposition
      t = (result ∩ Ri)+ ∩ Ri
      result = result ∪ t
```

Algorithms for Decomposition Using Functional Dependencies

non-trivial (비자명한) dependency $\alpha \to \beta$가 BCNF 위반인지 확인하려면.
① α^+를 계산
② α^+가 R의 모든 속성을 포함하는지 확인 (그렇다면 R의 superkey임)

간단한 테스트

F에서 BCNF 위반하는지 확인, F^+의 모든 종속성 확인할 필요X
(왜? F의 종속성 중 어느것도 BCNF 위반하지 않으면, F^+에 대해서도 마찬가지일것)

그러나! R을 분해할때는 틀릴수 있다.
ex) $R = (A,B,C,D,E)$, $F = \{A \to B, BC \to D\}$ (A가 BCNF위반이라고 가정한듯)
$R_1 = (A,B)$ $\alpha \cup \beta$
$R_2 = (A,C,D,E)$ $R - \beta + \alpha$
사실
$AC \to BC \to D$. $\therefore AC \to D$ 이므로 R_2는 BCNF아님.

해결) Testing Decomposition for BCNF

R의 분해 R_i에 대해서 BCNF 위반 확인
① F^+의 R_i에만 포함된 속성들로 확인 $\alpha \to (\alpha^+ - \alpha) \cap R_i$ 가 BCNF 위반하는것.
② F에 대하여
→ R_i의 모든 속성 α의 α^+가 $R_i - \alpha$의 속성을 전혀 포함하지 않던지
R_i의 모든 속성을 포함하던지

ex) $R_2 = (A,C,D,E)$
$F = \{A \to B, BC \to D\}$ $AC \to D$.
$\alpha = \{AC\}$면 $(AC)^+ = \{ABCD\}$ 어? D를 포함하네. → BCNF위반.
$AC \to D$.
$\alpha \to \beta$.
R_3 {A,C,D} $\alpha \cup \beta$
R_4 {A,C,E} $R - \beta + \alpha$ ⟶ 이러면 BCNF 무함

05/12

05/14

BCNF Decomposition Algorithm

```
result := {R};
done := false;
~~compute F+;~~
while (not done) do
  if (there is a schema Ri in result that is not in BCNF)
    then begin
      let α → β be a nontrivial functional dependency that
      holds on Ri such that α+ does not contain Ri
      and α ∩ β = ∅;
      result := (result – Ri) ∪ (Ri – β) ∪ (α, β);
    end
  else done := true;
```

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

Third Normal Form

dept_advisor (s_ID, i_ID, dept_name)
$F = \{$ s_ID, dept_name → i_ID, i_ID → dept_name $\}$

candidate keys: (s_ID, dept_name), (i_ID, s_ID)

superkey. candidate key에 속해 있음.

Testing for 3NF

→ F에 있는 FD만 체크.
$\alpha \to \beta$ 에 대하여
α가 superkey가 아니면 β가 candidate key에 속해있는지 check.

3NF algorithm

① F_c (canonical cover) 구하기.
② 각 함수 종속 (F_c에 있는)으로 스키마 생성
③ $R_1 \sim R_i$ 중 어느것도 R의 후보키를 포괄하지 않으면, 새로운 스키마 추가 $R_{i+1} = R$의 잉의 후보키.
④ 중복스키마 삭제
R_i가 R_k에 속해 있다면 (ex $R_i \subseteq R_k$), R_i 삭제.

```
i := 0;
for each 함수 종속 α → β in Fc do
   i := i + 1;
   Ri := αβ  // α와 β를 합친 스키마
```

ex) $R = (A,B,C,D,E)$
$F = \{A \to BC, C \to D, B \to E, A \to D\}$
① F_c구하기.
$A \to BCD$, $C \to D$, $B \to E$
② 각 함수 종속으로 스키마 생성
$R_1 = \{A,B,C,D\}$
$R_2 = \{C,D\}$
$R_3 = \{B,E\}$
③ Candidate key 확인
$A^+ = \{A,B,C,D,E\}$ 다 나오니까 A가 후보키.
추가스키마 불필요.

ex) $R = (A,B,C,D)$
$F = \{B \to C, C \to D\}$
$F_c = \{B \to C, C \to D\}$
$R_1 = \{B,C\}$
$R_2 = \{C,D\}$
candidate key는 $\{A,B\}$
R_3 추가 $\{A,B\}$

dependency preserving과 lossless join을 보장!!

ex 03)
cust_banker_branch = (customer_id, employee_id, branch_name, type)

functional dependencies

customer_id, employee_id → branch_name, type
employee_id → branch_name
customer_id, branch_name → employee_id.

F_c 후보 키(선) 이거네 extraneous?
F'
α^+ = {cid, eid, type, branch_name) extraneous하다네.

customer_id, employee_id → type
employee_id → branch_name
customer_id, branch_name → employee_id.

(customer_id, employee_id, type)
(employee_id, branch_name)
(customer_id, branch_name, employee_id) → candidate key 포함!

이건 상관x.

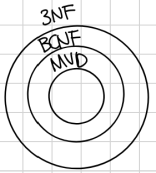
(customer_id, employee_id, type)
(customer_id, branch_name, employee_id) → 최종.

Comparison of BCNF & 3NF

3NF ⇒ loseless
dependencies preserved

BCNF ⇒ loseless
항상 dependencies preserving은 x.

Multivalued Dependencies



inst_child (ID, child_name)
inst_phone (ID, phone_number)

inst_info (ID, child_name, phone_number)
ex)
(99999, David, 512-555-1234)
(99999, William, 512-555-4321)
(99999, William, 512-555-1234)
(99999, David, 512-555-4321)

multivalued dependency $\alpha \rightarrow\rightarrow \beta$

$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$
$t_3[\beta] = t_1[\beta]$
$t_3[R-\beta] = t_2[R-\beta]$
$t_4[\beta] = t_2[\beta]$
$t_4[R-\beta] = t_1[R-\beta]$

t_1	x	y	L
t_2	x	z	ㄱ
t_3	x	y	ㄱ
t_4	x	z	L

Y, Z, W

$Y \rightarrow\rightarrow Z$ (Y multidetermines Z)

$\langle y_1, z_1, w_1 \rangle \in r$ and $\langle y_2, z_2, w_2 \rangle \in r$
$\langle y_1, z_1, w_2 \rangle \in r$ and $\langle y_2, z_2, w_1 \rangle \in r$

R = (A, B, C)

A →→ B

(a, b1, c1)	(a, b2, c1)	(a, b3, c1)
(a, b1, c2)	(a, b2, c2)	(a, b3, c2)
(a, b1, c3)	(a, b2, c3)	(a, b3, c3)

Fourth Normal Form

$\alpha \rightarrow\!\!\rightarrow \beta$ is trivial
α is a superkey for R

```
result: = {R};
done := false;
compute D+;
Let Di denote the restriction of D+ to Ri
while (not done)
  if (there is a schema Ri in result that is not in 4NF) then
    begin
      let α →→ β be a nontrivial multivalued dependency that holds
       on Ri such that α → Ri is not in Di, and α∩β=ϕ;
      result := (result - Ri) ∪ (Ri - β) ∪ (α, β);
    end
  else done:= true;
Note: each Ri is in 4NF, and decomposition is lossless-join
```

example)

R = (A, B, C, G, H, I)
F = {A →→ B, B →→ HI, CG →→ H}

R_1 = (A, B) → 4NF ○
R_2 = (A, C, G, H, I) → 4NF ×

R_3 = (C, G, H)
R_4 = (A, C, G, I)

R_5 = (A, I)
R_6 = (A, C, G)

Restriction 과정:

1. D+에서 α ⊆ Ri인 모든 MVD α →→ β 찾기
2. 각각에 대해 α →→ (β ∩ Ri) 형태로 변환
3. 결과가 non-trivial하고 Ri 속성들만 포함하는 것들만 선택