Database

Lecture 6. Relational Database Design

Spring 2024

Prof. Jik-Soo Kim, Ph.D.

E-mail: jiksoo@mju.ac.kr

Notes

- **Readings**
  - Chapter 7. Relational Database Design (Database System Concepts 7th Edition)

Basic Concepts

- **Database Design**
  - 어떤 테이블(Relation)을 만들 것인가?
  - 각 테이블은 어떤 Attribute들을 가지게 할 것인가?

- **목표**: 불필요한 중복(redundancy)없이 필요한 정보를 모두 표현(저장)할 수 있는 Schema

  => 어떤 attribute들을 갖는 어떤 table을 둘 것인가?

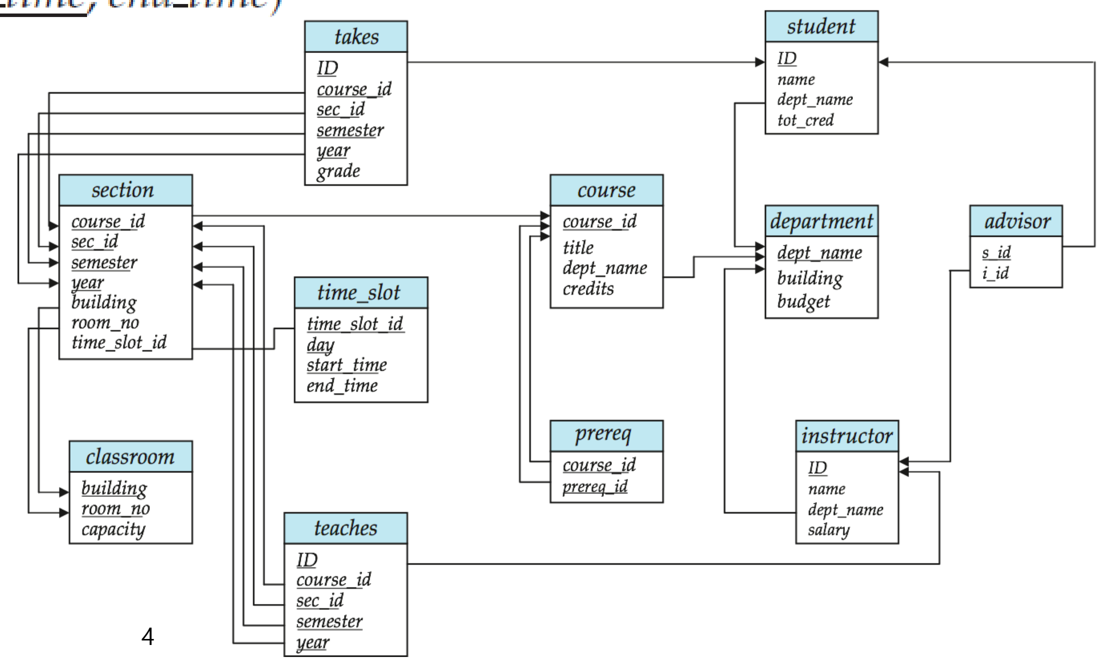
  R = (A B C D E) <---- single relation schema 정의

  DB_1 = { R_1, …… , R_n }<---- DB schema 정의(a set of relation schemas)

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

Example Schema for the University Database

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)



Combine Schemas into Larger Ones?

- combine *instructor* and *department* into *inst_dept* ?

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

repetition of information, updating budget?, creating a new department?

What about Smaller Schemas?

- How would we know to split up (**decompose**) the *inst_dept* into *instructor* and *department* ?

- Write a rule "if there were a schema (*dept_name, building, budget*), then *dept_name* would be a candidate key"
  - denote it as a **functional dependency**:

    $dept\_name \rightarrow building, budget$

    A가 b,c의 필요한 대장 느낌

  - in *inst_dept*, because *dept_name* is *not* a candidate key, the *building* and *budget* of a department may have to be *repeated* !

What about Smaller Schemas?

- **Not all decompositions are good! ☹**

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
⋮				

??

동명이인때문에 원래 테이블에 없던 튜플이 생김.

"lossy decomposition"

Database Design Goal

Find a **"good" collection of relation schemas**
for our information need

- **나쁜 디자인**
  - Inability to represent certain information
  - Repetition of Information
  - Loss of information
  - **Anomaly**
    - 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

Bad Design 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

- **Redundancy**
  - data for *branch-name, branch-city, assets* are repeated for each loan that a branch makes (wastes of 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-name* 등 insert 불가
  - **Deletion Anomaly**: 어떤 branch의 마지막 loan을 delete하면 branch 자체가 사라짐
  - **Update Anomaly**: 어떤 특정한 branch의 정보(예: Downtown의 *assets*)를 update하면 해당되는 튜플들을 모두 업데이트해야 함

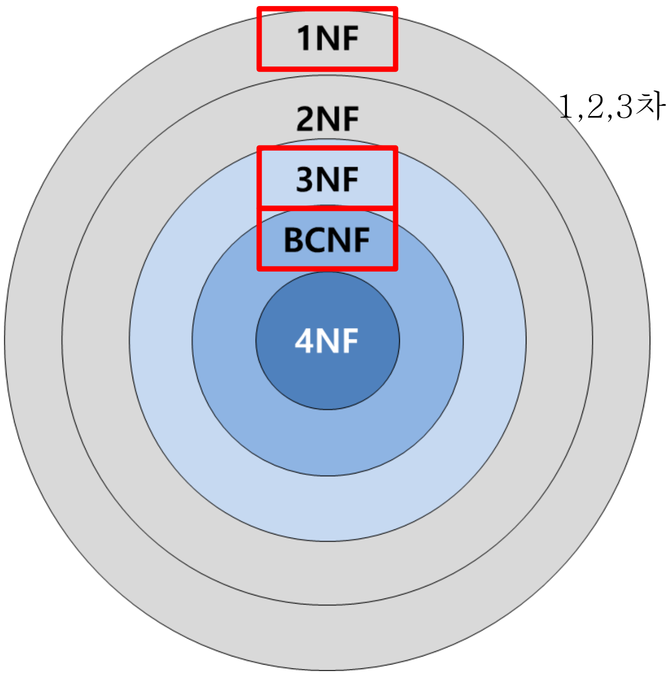
- 원인
  - 정보의 중복(**Redundancy**)
  - 여러 entity가 하나의 table에 합쳐짐

- 해결책: **Decomposition**

Normalization (정규화)

- 관계형 데이터베이스의 설계에서 **중복을 최소화하게 데이터를 구조화**하는 프로세스
  - generate a set of relation schemas that allows us to store information without unnecessary redundancy, yet also allows us to retrieve information easily

    추출하다
  - design schemas that are in an appropriate **Normal Form**



1,2,3차 정규화까지

First Normal Form

- Domain is **atomic** if its elements are considered to be *indivisible*
  - examples of non-atomic domains
    - set of names
    - composite attributes (e.g., address = street, city, state, zip)
    - identification numbers like "CS101" or "EE1127" that can be broken up into parts

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

- A relational schema is in **First Normal Form(1NF)** if the domains of all attributes are *atomic*

위 예시에선 곡 하나만 잡는게 아니라 나열된 곡 전체를 한번의 string으로 사용했야 1NF이다라고 할 수 있음.

Decomposition

- **Definition**
  - let R be a relation scheme
  - $\{R_1, \ldots, R_n\}$ is a *decomposition* of R
    if $R = R_1 \cup \ldots \cup R_n$ (즉, R의 모든 attribute가 $R_1, \ldots, R_n$에 존재)

- We will deal mostly with binary decomposition:
  R into $\{R_1, R_2\}$ where $R = R_1 \cup R_2$

우리는 한번에 두개씩만 쪼개기로

Decomposition - Examples

- 학생(학번, 이름, 학과, 학과장, 학과전화, 학년)

  => 학생(학번, 이름, 학년, 학과) 둘이 합쳤을 때 원본이 나와야함

  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)

- Is this a good decomposition?

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** 교집합 즉 연결이 사라짐

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

정보 왜곡

- natural join시 tuple 증가
- information(relationship) 상실
- 잘못된 상관 관계(**b_city**)

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

즉 잘하려면 P.K를 공통으로 쪼개자.

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

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

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

같을때만 즉 원본일때만

- 기준 information은 원래 r의 information

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 (we will not cover this!)

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 수 없다**
  - a functional dependency is a generalization of the notion of a key
- **실제** Application의 규칙에 의해 정해짐(**실 세계**의 규칙을 반영)
  - 학번 → 이름 ?? 맞음
  - 이름 → 학번 ?? 틀림 why?동명이인
  - 주민등록번호 → 학번 ?? 맞음
- Relation의 **모든** instance에서 이 규칙이 성립하여야 함

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

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

- Example
  - consider $r(A, B)$ with the following instance

A	B
1	4
1	5
3	7

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

1,4 / 1,5로 겹침

Applications of Functional Dependencies

- 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
  결정한다라고 보자
  - $K \rightarrow R$, and
  - for no $\alpha \subset K$, $\alpha \rightarrow R$

- Functional dependencies allow us to express constraints that cannot be expressed using superkeys

Loan-info-schema = (*customer-name*, *loan-number*, *branch-name*, *amount*)

We expect the following functional dependencies to hold:

loan-number $\rightarrow$ *amount*
loan-number $\rightarrow$ *branch-name*

but would not expect the following to hold:

loan-number $\rightarrow$ *customer-name* 공동대출가능해서 이건 안됨

(in general, a given loan can be made to more than one customer)

Applications of Functional Dependencies

- 주의: 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 *Loan-info-schema* may by chance, satisfy

$$loan\text{-}number \rightarrow customer\text{-}name$$

상식으로 참거짓판별하자(테이블로만 말고)

Functional Dependencies Example

- Consider the schema,

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

  We expect these functional dependencies to hold:

  $dept\_name \rightarrow building$ and $ID \rightarrow building$

  but would not expect the following to hold:

  $dept\_name \rightarrow salary$

Trivial Functional Dependencies

- A functional dependency is **trivial** if it is satisfied by all instances of a relation
  - example
    - *customer-name, loan-number* $\rightarrow$ *customer-name*
    - *customer-name* $\rightarrow$ *customer-name*

- **당연히!** 성립할 수 밖에 없는 FD

- 예)
  - {이름, 나이} $\rightarrow$ {이름}
  - {나이} $\rightarrow$ {나이}
  - {나이} $\rightarrow$ ϕ

- $\alpha \rightarrow \beta$ is **trivial** if $\beta \subseteq \alpha$ 부분집합이면 생각할 필요없이 맞는거

MYONGJI UNIVERSITY

Closure of a Set of Functional Dependencies

- Given a set F of FDs, there are certain other FDs that are logically *implied* by F
  - e.g., if $A \to B$ and $B \to C$, then we can infer that $A \to C$ 삼단논법
- The set of all functional dependencies logically implied by F is the **closure** of F and denote the closure of F by F^+

위 조건중 하나만이라도 만족하면 됨.

- We can find all of F^+ by applying **Armstrong's Axioms:**
  - if $\beta \subseteq \alpha$, then $\alpha \to \beta$ **(Reflexivity)**
  - if $\alpha \to \beta$, then $\gamma\,\alpha \to \gamma\,\beta$ 여기서 r은 name같은 어떠한 조건 **(Augmentation)**
  - if $\alpha \to \beta$, and $\beta \to \gamma$, then $\alpha \to \gamma$ **(Transitivity)**
- These rules are F+를 적용하면 무조건 유효한 것
  - **sound** (generate only functional dependencies that actually hold) and **complete** (generate all functional dependencies that hold)

MYONGJI UNIVERSITY

Closure of a Set of Functional Dependencies

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

- Some members of F^+
  - $A \rightarrow H$ 3번째 법칙
    - by transitivity from $A \rightarrow B$ *and* $B \rightarrow H$
  - $AG \rightarrow I$ 2,3번째 법칙
    - by augmenting $A \rightarrow C$ with G, to get $AG \rightarrow CG$ and then transitivity with $CG \rightarrow I$
  - $CG \rightarrow HI$ cg−>h, cg−>i를 합하면 cgcg−>hi 이는 부분집합개념이라 cg−>cgi 즉 cgi−>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

Union, Decomposition, Pseudotransitivity

- **Additional rules for finding the closure**
  - 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)** 증명은 ar->br = -> d

→ above rules can be inferred from Armstrong's axioms!

Attribute Set Closure

- Given a set of attributes α, define the **closure** of α **under** F (denoted by α^+) as 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

- Algorithm to compute α^+, the closure of α under F

result := α;
while (changes to *result*) **do**
    for each $\beta \rightarrow \gamma$ **in** F **do**
      begin
        if $\beta \subseteq$ *result* **then** *result* := *result* $\cup\ \gamma$
      end

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. $result = AG$ 스스로 결정되니
  2. $result = ABCG$ $(A \rightarrow C$ and $A \rightarrow B)$ a가 b,c둘다 경하므로 가능
  3. $result = ABCGH$ $(CG \rightarrow H$ and $CG \subseteq AGBC)$
  4. $result = ABCGHI$ $(CG \rightarrow I$ and $CG \subseteq AGBCH)$

  (AG)+ = R이면 종료됨
  즉 AG->R == AG가 superkey
- Is AG a candidate key?
  1) Is AG a superkey?
     ① Does $AG \rightarrow R$? == Is $(AG)^+ \supseteq R$ 예시의경우 O
  2) Is any subset of AG a superkey?
     ① Does $A \rightarrow R$? == Is $(A)^+ \supseteq R$ 예시의 경우 X,X
     ② Does $G \rightarrow R$? == Is $(G)^+ \supseteq R$

Use Cases of Attribute Set Closure

- Testing for superkey
  - to test if α is a superkey, we compute α^+ and check if α^+ 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 $\beta \subseteq \alpha^+$
    - compute α^+ by using attribute set closure, and then check if it contains β
  - a simple and cheap test, and very useful

- Computing closure of F
  - for each $\gamma \subseteq R$, we find the closure γ^+, and for each $S \subseteq \gamma^+$, we output a functional dependency $\gamma \rightarrow S$

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^+:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$

→ In other words, if $R_1 \cap R_2$ forms a **superkey** of either R_1 or R_2, the decomposition of R is a lossless decomposition

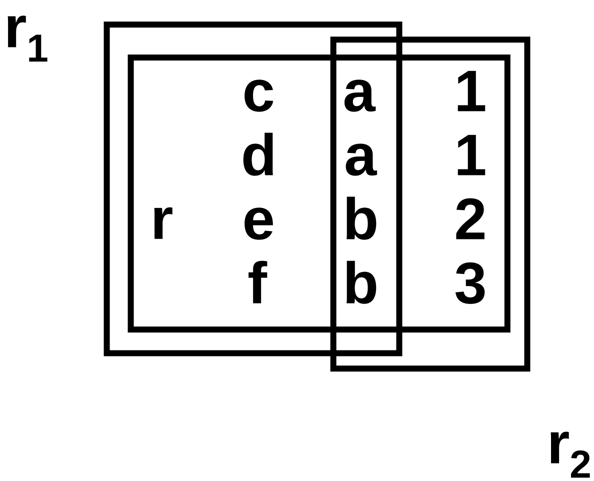
즉 한쪽이 슈퍼키면 합쳐도 늘어날일이 없다.

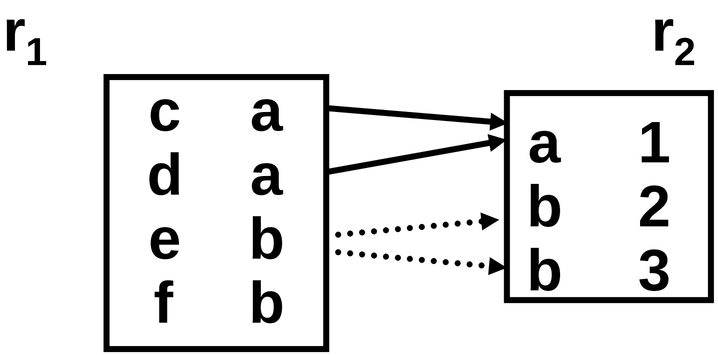
Lossless-join Decomposition Revisited

- $\{R_1, R_2\}$ 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





둘이 natural join시 e b b 3, f b b 2 총 두개가 더 생김. 즉 superkey가 없음.

Lossless-join Decomposition Revisited

- $R = (A, B, C)$
  $F = (A \rightarrow B, B \rightarrow C)$
  - can be decomposed in two different ways

- $R_1 = (A, \boldsymbol{B}), R_2 = (\boldsymbol{B}, C)$
  - Lossless-join decomposition: $R_1 \cap R_2 = \{B\}$ and $B \rightarrow BC$
  - **Dependency preserving**

즉b가 r2에서 슈퍼키이므로 가능

- $R_1 = (\boldsymbol{A}, B), R_2 = (\boldsymbol{A}, C)$
  - Lossless-join decomposition: $R_1 \cap R_2 = \{A\}$ and $A \rightarrow AB$
  - **Not dependency preserving** 의존성 보존 실패
    (cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

무손실 분해는 맞음(

r1과 r2를 합쳐야 b,c판별가능

Dependency Preservation

이름	시	도
홍길동	인천	경기

F = {이름 → 시, 도; 시 → 도}

3법칙에 따라 이름 -> 도 가능

이름	시
홍길동	인천

시	도
인천	경기

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

이름	시
홍길동	인천

이름	도
홍길동	경기

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

무조건 join을 해야 시->도를 알기 때문에 이건 안쓴다.

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** 정보의 무손길 분해
2. **No redundancy** 중복 최소화
3. **Dependency preservation** 의존성 보존

MYONGJI UNIVERSITY

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:

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

  because *dept_name* $\rightarrow$ *building, budget* holds on *inst_dept* but *dept_name* is not a superkey!

  이거때문에 BCNF실패

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:
  - **(α U β)** R_1
  - **(*R* - (β - α))** R_2

- In our example,

  inst_dept (ID, name, salary, dept_name, building, budget)
  - α = *dept_name*
  - β = *building, budget*

  and *inst_dept* is replaced by
  - (α U β) = (*dept_name, building, budget*)
  - (*R* - (β - α)) = (*ID, name, salary, dept_name*)

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*** 하나론 superkey가 안됨
  - decomposition
    - *course* (*c_id, title, dept, credits*)
    - *class-1 (c_id, sec_id, semester, year, building, room, capacity, time_slot_id)* aUb 와 r −(b−a)로 쪼갠다

- course는 BCNF인가?
  - Yes! (*c_id*가 superkey임)

- class-1은 BCNF인가?
  - No! ***building, room*→*capacity*** 안됨. 찢어져야됨.
  - decomposition
    - *classroom* (*building, room, capacity*)
    - *section* (*c_id, sec_id, semester, year, building, room, time_slot_id*)
      이렇게 찢어야 결과적으로 BCNF전체 다 만족

MYONGJI UNIVERSITY

Example of BCNF Decomposition

- 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
  ① R → R_1 = (b-name, b-city, assets),
  R_2 = (b-name, c-name, loan-n, amount)
  ② R_2 → R_3 = (b-name, loan-n, amount),
  R_4 = (c-name, loan-n)

- Final decomposition result: R_1, R_3, R_4

MYONGJI UNIVERSITY

BCNF and Dependency Preservation

- $R = (J, K, L)$
  $F = \{JK \rightarrow L, L \rightarrow K\}$
  Two candidate keys = JK and JL

  1은 되는데 2가 문제
  즉 (l,k) (j,l)로 나눔
  어케해도 만족안됨
- 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

It is not always possible to get a BCNF decomposition that is dependency preserving!

BCNF 분해가 의존성을 보존하지 않을 수도 있음!

→ BCNF보다 약한 정규형이 필요함

Third Normal Form: Motivation

- There are some situations where
  – BCNF is not dependency preserving, and
  – efficient *checking* for FD violation on updates is important
    (3NF makes testing of updates cheaper)
  → 데이터베이스의 일관성 유지에 중요할 수 있음!

- Solution: define a weaker normal form, called **Third Normal Form**
  – allows some *redundancy* (with resultant problems) 정보중복이 필연적
  – but FDs can be checked on individual relations without computing a join
  – 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 $\alpha \rightarrow \beta$ 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

    BCNF조건
  - **Each attribute A in $\beta - \alpha$ is contained in a candidate key**
    (NOTE: each attribute may be in a different candidate key)

- BCNF는 항상 3NF에 속함

MYONGJI UNIVERSITY

3NF Example

- Relation ***dept_advisor*** (학과-지도교수)
  - *dept_advisor* (*s_ID*, *i_ID*, *dept*)

    F = {(*s_ID*, *dept* → *i_ID*), (*i_ID* → *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* → *i_ID* (s_ID, dept is a superkey)
    - *i_ID* → *dept* (dept is contained in a candidate key)
  - BCNF는 성립하지 않음 (*i_ID* → *dept*)
  - BCNF 분해를 할 경우 ②를 검사하는 것이 불가능함

dept_advisor

(i_id, dept)

(s_id,i_id)

Redundancy in 3NF

- Example of problems due to redundancy in 3NF
  - $R = (J, K, L)$
    $F = \{JK \rightarrow L, L \rightarrow K\}$

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

- repetition of information (e.g., the relationship l_1, k_1)
  - e.g., (*i_ID, dept_name*) qksqhr rksmd

- need to use *null* values (e.g., to represent the relationship l_2, k_2 where there is no corresponding value for J) ㅜㅕㅣㅣrkqtdl skdhf tn dlTek.
  - e.g., (*s_ID, i_ID, dept_name)* if there is no separate relation mapping instructors to departments

Comparison of BCNF and 3NF

- **3NF**
  - Lossless & Dependency Preserving Decomposition 가능
  - Data Redundancy가 존재 → **Anomaly** 존재할 수 있음

- **BCNF**
  - Lossless Decomposition 가능
  - Dependency Preserving **불가능**할 수 있음
  - Data Redundancy 없음

Another Definition of 3NF

- A database is in **third normal form (3NF)** if it satisfies the following conditions:
  - it is in second normal form
  - it contains only columns that are **non-transitively dependent** on the primary key
    삼단논법이 불가능해야됨

TABLE_BOOK_DETAIL

Book ID	Genre ID	Genre Type	Price
1	1	Gardening	25.99
2	2	Sports	14.99
3	1	Gardening	10.00
4	3	Travel	12.99
5	2	Sports	17.99

Primary Key: [Book ID]
[Book ID] → [Genre ID]
[Genre ID] → [Genre Type]
⇒ [Book ID] → [Genre ID] → [Genre Type]
(transitive functional dependency!)

출처: https://www.1keydata.com/database-normalization/third-normal-form-3nf.php

Another Definition of 3NF

- A database is in **third normal form (3NF)** if it satisfies the following conditions:
  - it is in second normal form
  - it contains only columns that are **non-transitively dependent** on the primary key

TABLE_BOOK

Book ID	Genre ID	Price
1	1	25.99
2	2	14.99
3	1	10.00
4	3	12.99
5	2	17.99

TABLE_GENRE

Genre ID	Genre Type
1	Gardening
2	Sports
3	Travel

3NF Decomposition

출처: https://www.1keydata.com/database-normalization/third-normal-form-3nf.php

Second Normal Form (2NF)

- A database is in **second normal form (2NF)** if it satisfies the following conditions
  - it is in first normal form 모든 어트리뷰트가 atomic해야된다.(하나의 단일 밸류로만 취급되야된다.)
  - all non-key attributes are **fully functional dependent** on the primary key p.k가 두개이상일 때 의심해보기

TABLE_PURCHASE_DETAIL

Customer ID	Store ID	Purchase Location
1	1	Los Angeles
1	3	San Francisco
2	1	Los Angeles
3	2	New York
4	3	San Francisco

Primary Key: [Customer ID, Store ID]

[Purchase Location] only depends on [Store ID]!

출처: https://www.1keydata.com/database-normalization/second-normal-form-2nf.php

Second Normal Form (2NF)

- A database is in **second normal form (2NF)** if it satisfies the following conditions
  - it is in first normal form
  - all non-key attributes are **fully functional dependent** on the primary key

TABLE_PURCHASE

Customer ID	Store ID
1	1
1	3
2	1
3	2
4	3

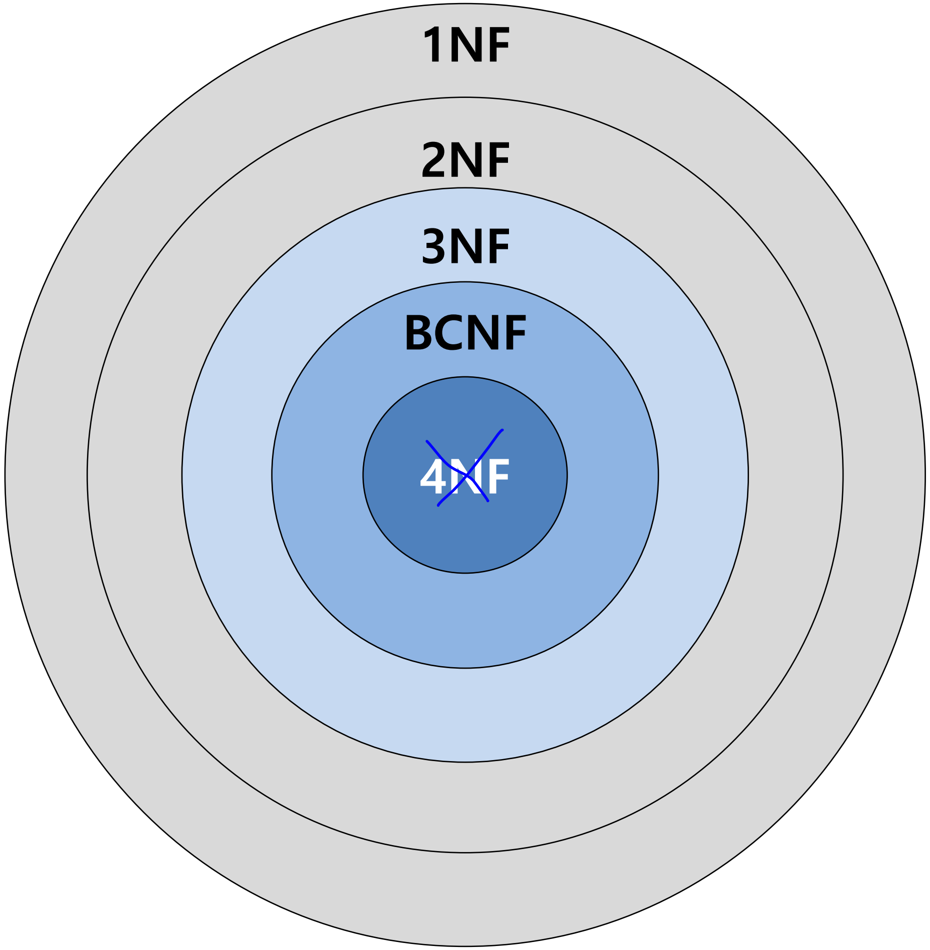
TABLE_STORE

Store ID	Purchase Location
1	Los Angeles
2	New York
3	San Francisco

2NF Decomposition

출처: https://www.1keydata.com/database-normalization/second-normal-form-2nf.php

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 의존성을 버리거나
  - Redundancy due to use of 3NF 정보중복허용 하거나

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

검토 수단으로 정규화이론을 쓰자.

Denormalization for Performance 반정규화

- May want to use non-normalized schema for performance 성능을 위해
  - e.g., 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
  account $\bowtie$ depositor
  - 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
    - an example of a **crosstab**, where values for one attribute become column names (used in spreadsheets, and in data analysis tools)

THE END