11. Database Design theory

11.1 Good Schema and Bad Schema

나쁜 스키마 - 이미 존재하는 테이블을 합쳐놓은 형태..

anomalies

- Update
- Delete
- Insert

예시

cID	title	deptName	credit	chairman	building	budget
202	Java	CS	2	Lee	IT building	67000
203	Data Structure	CS	3	Lee	IT building	67000
301	Databases	CS	3	Lee	IT building	67000
241	Logic Circuit	EE	2	Han	HN Eng	82500
341	LAN	EE	3	Han	HN Eng	82500
342	Electronics	EE	3	Han	HN Eng	82500
222	Computer Art	Media	3	Paik	IT building	55500
223	Game Theory	Media	3	Paik	IT building	55500

- Update 학과장 바뀌면 3개의 튜플을 업데이트 해야함
- Delete CS의 전체 과목이 없어지면 CS과에 대한 학과장, 예산 정보가 다 날라감
  (CS과는 존재함에도 불구하고..)
- Insert 신생 과 AI 생겼을 때, 그 과의 과목을 개설하여 cID를 얻을 때까지 테이블에 insert하지 못한다.
  -> primary key가 없기 때문이다.

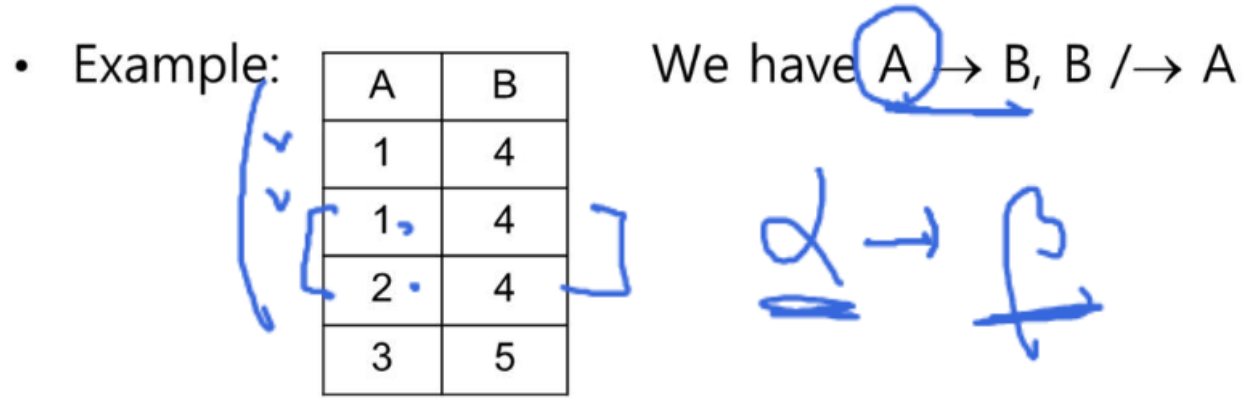
Design Goal

스키마 주어지면 좋은 형태인지 아닌지 판단한다.
만약 나쁜 형태면 스키마 분해해서 각각 좋은 형태로 다시 테이블 생성

- functional dependency
- multivalued dependency 로 하기

functuinal dependency

함수 종속성
$\alpha -> \beta$ 알파가 동일하면 베타도 동일해야 한다.



1-2 성립 2-3 성립 (알파가 다르니까 베타는 뭐가 나와도 상관없음) 3-4 성립

하지만 $\beta -> \alpha$ 성립하지 않는다.

- FD and keys 알파가 모든 컬럼을 결정하는 경우에 알파는 슈퍼키이다. (알파->R)

만약 베타가 알파에 포함되지 않고, 베타 -> R이면 알파는 후보키이다

11.2 Functional Dependency Theory

주어지는 함수 종속성으로부터 새로운 함수 종속성을 imply하도록 추론할 수 있다.

Trivial Functional Dependency

함수 종속성이 테이블의 모든 instatnce에 만족하는 것.
$\alpha -> \beta\qquad \beta\in\alpha$ 알파가 베타를 결정하고, 알파가 베타를 포함한다

Closure of a Set of Functional Dependency

주어진 함수 종속성으로부터 유추해서 다른거 만들어내기
기존 것에서 새로운 것 derive하기

- Armstrong's Axioms

1. reflexivity 알파가 베타보다 크면 알파가 베타를 결정한다
2. augmentation 알파가 베타를 결정하면 알파/베타에 속성 첨가해도 동일하다.
3. transitivity 알파가 베타를 결정하고, 베타가 감마를 결정하면 알파도 감마를 결정한다.

예시

- R = (A, B, C, G, H, I)
  F = {A → B A → C
  CG → H CG → I B → H}
- some FDs that can be derived from given F
  - A → H
  - AG → I
  - CG → HI
    - by augmenting CG → I to infer CG → CGI, and augmenting of CG → H to infer CGI → HI, and then transitivity
  - A → BC

- a-> h (transitivity)
- ag -> i (augemtation, transitivity)
- cg -> hi (cg->h, cg->i)
  cg -> i (augemting) cgcg(cg) -> cgi
  cg -> h (augemting) cgi -> hi cg -> hi (transitivity)
- a -> bc (동일한 방식)

이러한 걸로 모든 함수종속성 만들 수 있다!!

additional rule

- Additional rules
  - 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)
- The above rules should be inferred from Armstrong's axioms (Why???)

이것도 암스트롱으로 할 수 있다.
--> valid한 모든 종속성을 만들 수 있기 때문에.

지금부터는 set 형식

Attribute Set Closure

- $R = (A, B, C, G, H, I)$
  - $F = \{A \rightarrow B \quad A \rightarrow C \quad CG \rightarrow H$
    $CG \rightarrow I \quad B \rightarrow H\}$
- $(AG)^+$
  - closure = AG
  - closure = ABCG $(A \rightarrow C$ and $A \rightarrow B)$
  - closure = ABCGH $(CG \rightarrow H$ and $CG \subseteq AGBC)$
  - closure = ABCGHI $(CG \rightarrow I$ and $CG \subseteq AGBCH)$
- Is AG a candidate key?
  - Is AG a super key? Yes, since $AG \rightarrow ABCGHI$ (all attributes)
  - Is any subset of AG a super key? No
    - Since $A \rightarrow ABCH$ (which is not all attributes) and $G \rightarrow G$
  - So AG is a candidate key

- ag a->b, a-> c
- abcg cg->h, cg->i
- abcgh
- abcgi

슈퍼키인지 확인

ag -> 모든 속성을 포함하기에 슈퍼키이다.

- a, g가 각각 후보키가 아닌 것을 확인해야 minimal한 ag superkey 생성 가능
- A? a->b, a->c
  b-> h
  모든 속성 불가
- G? 모든 속성 불가

Attribute closure의 쓰임

- 슈퍼키 확인
- 함수 종속성 확인
- F의 closure 구하기

Canonical Cover

필요없는 속성 제외하기
a->b, b->c일 때 a->c를 명시할 필요 없음 (유추 가능하기 때문에)

알고리즘을 통해 설계된다..

Example

a->bc, a->b, b->c, ab->c

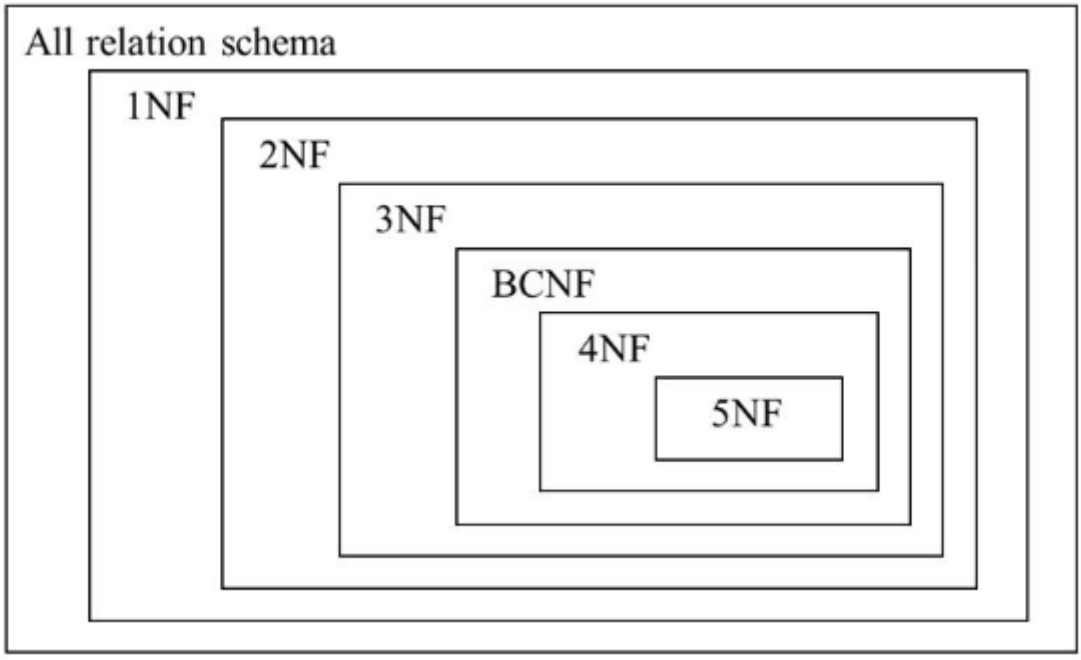
- a->bc로 a->b, a->c 유추가능 a->b 버리기
- ab->c에서 a를 제외했을 때 b->c는 이미 있으니 버리기
- a->bc, b->c에서 a->c로 바꾸기

Normal Forms

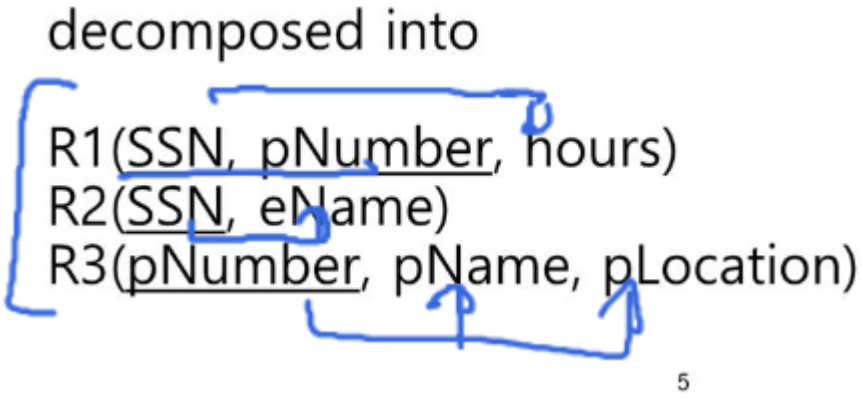
함수 종속성을 가지고 정의하는 정규형은 BCNF까지.

용어

- prime 후보키중에 하나라도 들어가면 prime attribute
- full func dependency a-> cde, b->cde로 결정하지 못하고
  ab->cde로만 결정 가능 하면 full func dependency
- partial func dependency a->cde 할 수 있다면 ab->cde는 partial func dependency
- transitively func dependency 종속적인 attribute로부터 통해서 종속적이게 된다.

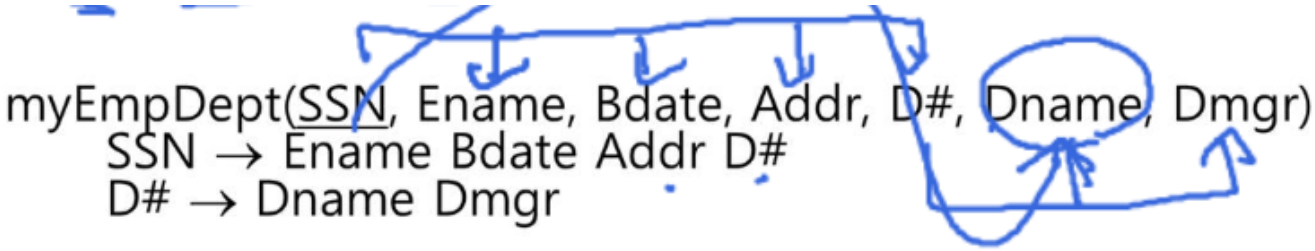


1. 1NF 관계형 db의 모든 값은 atomic한 값을 가져야한다.
2. 2NF 모든 nonprime attrbute가 후보키에 fully dependency하다



모든 후보키에 다른 속성이 의존적이어야한다.
a->n 아니고 ab->n이어야한다.

3. 3NF 2NF해야하며, no nonprime이 transitively dependent하면 안된다.



SSN -> D#
D# -> Dname, Dmgr
--> 이런 형식이면 안된다.

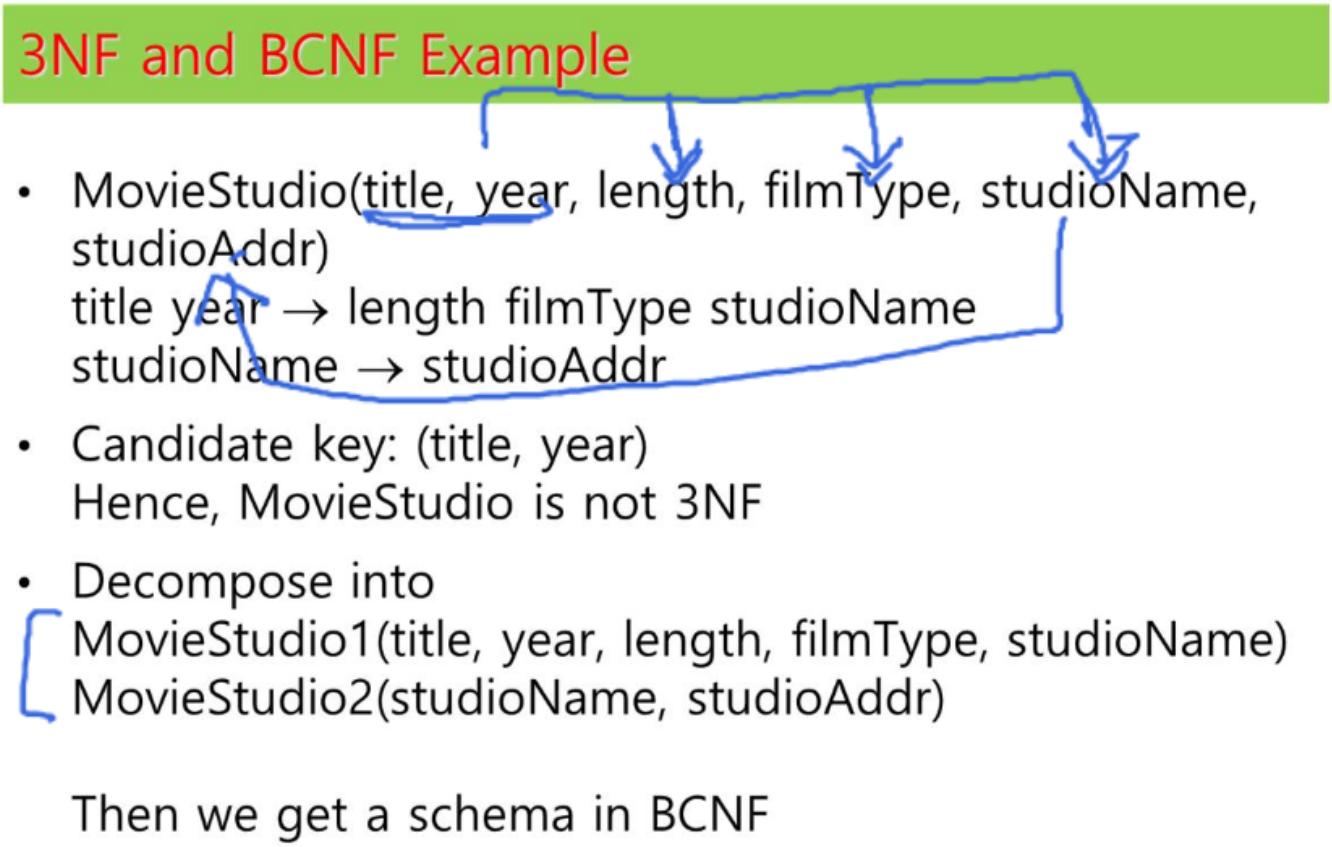
2NF이면서 3NF인것도 있지만 후보키가 아닌 속성에 대해 종속적인 컬럼 있다면 3NF 아님

BCNF

> 3NF에서 알파가 슈퍼키거나 그렇지 않으면 베타가 prime attribute면 된다.

이 조건을 제외한 것이 BNCF

3NF vs BCNF



첫번째는 2NF에 해당 (transitively하므로)

분해하면 BCNF가 된다.

- 테이블의 컬럼이 2개면 무조건 BCNF이다.

Exercise

1. a -> cd, b -> ef
   후보: ab prime: a,b non-prime: c, d, e, f type: 1NF //후보키에 일부 의존적이다 (a -> e,f 하지 못함)
2. ab -> cde, c -> d 후보: ab prime: a, b non-prime: c, d, e type: 2nf
3. a -> bc, c -> de 후보: a prime: a non-prime: b, c, d, e type: 2nf
4. ab -> cde, d -> b 후보: ab, ad prime: a, b, d non-prime: c, e type: 3nf //후보키가 아닌 d가 b를 결정하기 때문에 bcnf가 아님
5. a -> bcde, d -> a 후보: a, d prime: a, d non-prime: b, c, e type: bcnf //모든 함수 종속성에서 왼쪽 부분이 후보키가 된다.
6. a -> bcde 후보: a prime: a non-prime: b,c,d,e type: bcnf
7. a -> b, b -> c, c -> d, d -> a, d -> e 후보: a,b,c,d prime: a,b,c,d non-prime: e type: bcnf
   - 예시

   info(주민번호, 여권번호, 운전면허번호, 군번, 이름)