**- Chapter 08: Relational DB Design**

**Pitfalls(위험) of Relational Database Design**

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

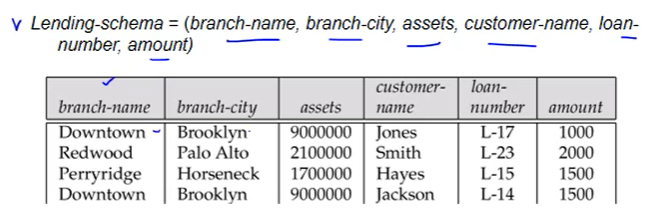
DB = { R1, …, Rn } <- DB schema (relation schema의 집합)

목표:

1. attribute들 간의 relationship 이 잘 표현됨.
2. redundant를 피함
3. integrity constraint를 enforce함.

잘못된 디자인 설계는 정보 손실, 정보 중복, 정보 표현 불능 등의 문제가 생길 수도 있음.

**Example**



문제: 중복이 있음. Loan-number마다 branch-name, branch-city, assets가 반복됨.

공간 낭비, 업데이트 할 때마다 복잡하게 함. Inconsistency 유발

여기서 관계는 association이지 값만 같다고 중복이 아님. Value의 association이 중복이여야 중복.

**Redundancy creates problems**

**Insertion anomaly**: loan이 없으면 branch에 대한 정보 저장 불가

**Deletion anomaly**: branch에 대한 마지막 account가 삭제되면 branch에 대한 정보가 삭제됨

**Update anomaly**: branch에 대한 asset을 모든 branch에 대해 변경해줘야 함.

이런 문제들은 redundancy 때문에 발생함

해답: 각각의 정보는 오직 한 번만 표현되도록 schema를 decompose 하는 것.

Information content: attribute 사이의 relationship.

**First Normal Form**

Domain이 atomic (요소들이 더 이상 나눌 수 없는 unit)이어야 함. 도메인이란? Attribute가 가질 수 있는 값의 집합

이 때 모든 domain이 1NF일 때 Relational schema가 1NF 라고 하고, DB schema의 모든 relational schema가 1NF여야 DB가 1NF라고 함.

모든 relational이 1NF라고 가정.

**Relational Theory**

특정한 relation R이 좋은 form인 지, 아닌 지 판단하는 것.

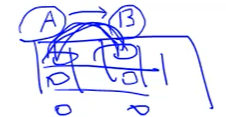
Relation의 집합을 decompose 함. 그래서 각각의 relation이 good form이고, lossless해야 함.

이 theory는 functional dependency, multivalued dependency에 의함.

**Functional Dependencies**

Attribute들 간의 legal relation의 집합에 대한 제약 조건.

특정한 attribute의 집합에 대한 값들이 유일하게 다른 attribute의 집합에 대한 값들을 결정함.

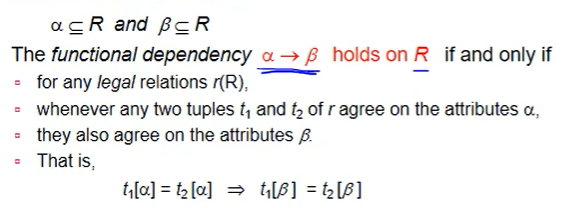


Functional dependency는 key라는 notion의 일반화임.

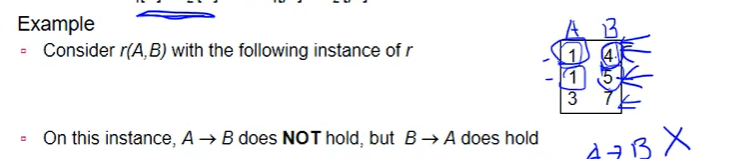


ID -> Name / Dept -> Dept\_office / Dept -> College / College -> Dean / Advisor -> Adv\_phone.

모든 attribute는 학번에 dependent하다고 할 수 있음.

R이 relation schema라고 하면, 

a -> b가 모든 legal relation에 대해 적용되므로 functional dependency를 constraint라고 함.



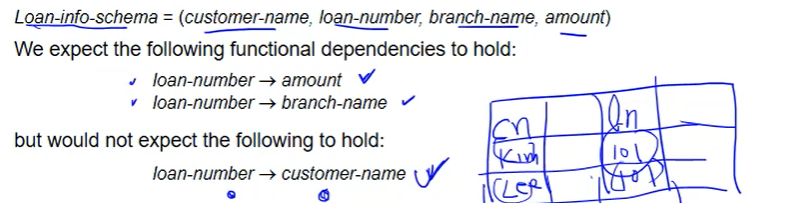
B->A는 같은 B에 대해 다른 A가 없으므로 B -> A는 만족함.

**Applications of FD**

Relation schema R에 대해 K -> R이 성립하면 K는 superkey임.

K는 R에 대해 K -> R이 성립하고 K의 진부분집합 a에 대해 a -> R이 성립하지 않으면 K는 candidate key임.

Functional dependency는 superkey를 사용해 표현하지 못하는 제약 조건들을 표현할 수 있게 해줌.



Functional dependency는 legal relation들의 집합에 대해 제약 조건을 정의함.

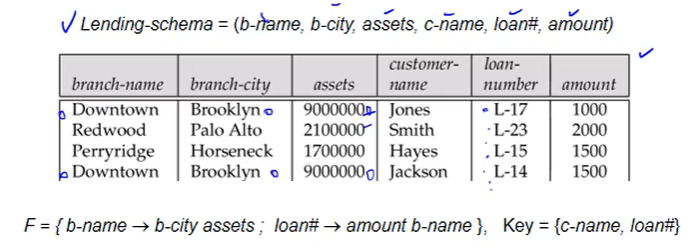
- 만약R의 모든 legal relation들이 FD의 집합 F를 만족할 때 F holds on R이라고 함.

relation들이 주어진 FD의 집합에 대해 legal한지 테스트함.

- 만약 relation r이 FD의 집합 f에 대해 legal하면, r satisfies F라고 함.

Note: relation schema의 특정한 instance는 FD가 모든 legal instance에 대해 성립하지 않아도 FD를 만족할 수도 있음. 그러니까 instance에 대해 FD를 보는 게 아니라 schema에 대해 보는 것.

**What causes redundancy?**



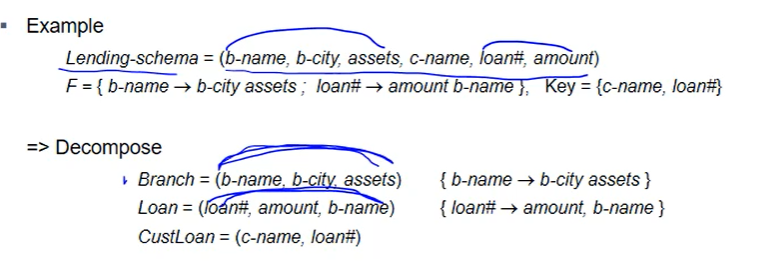
Key가 아닌 attribute에 functionally dependent하면 repeat 된다.

**Boyce-Codd Normal Form (BCNF) - informally**

Relation R의 attribute들이 key에만 dependent하면 좋은 형태라고 한다.

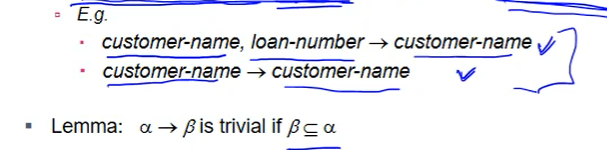
Non-key FD가 없어야 함.

Solution: tightly related attribute들로 R을 더 작은 relation들로 나눔.



**Trivial FD**

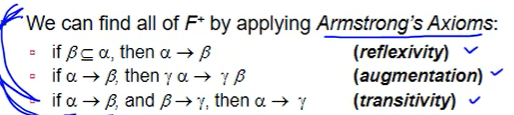
모든 instance에 대해 만족하면 FD가 trivial하다고 함.



**Closure of a Set of FDs**

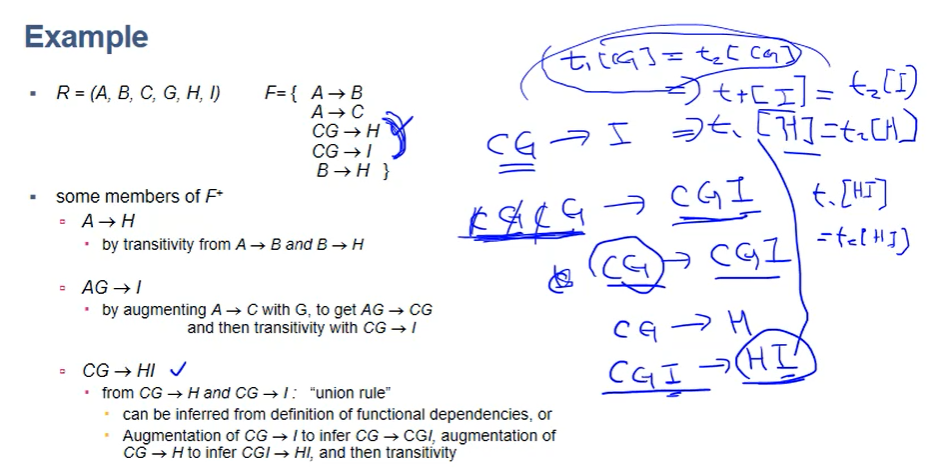
3단논법이 적용됨.

F에 의해 논리적으로 적용되는 모든 FD의 set은 F의 **closure**라고 함. (F^+로 나타냄)



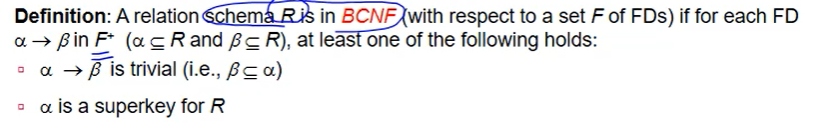
이러한 rule들은 **sound (올바른 FD만 generate) and complete (모든 FD를 generate)**.

**Example**

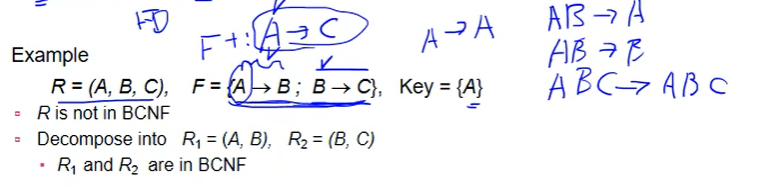


**Boyce-Codd Normal Form – formally**

R이 “good” form인지 판단하는 걸 원함.



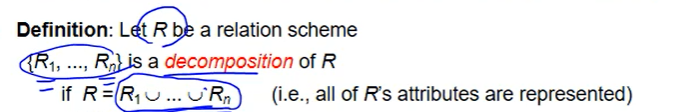
즉 a->b가 자명하거나, 아닌 경우는 a가 R의 superkey여야 함.



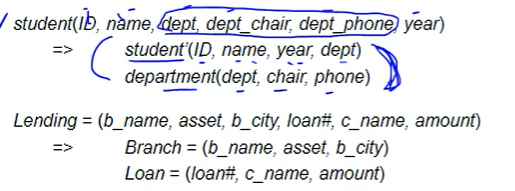
위 예시는 B가 key가 아니기 때문에 BCNF가 아님.

R1과 R2로 나누면 BCNF.

**Decomposition**



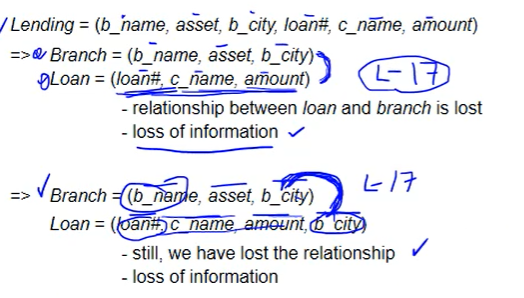
우리는 대부분 R을 R1과 R2로 나누는 decomposition을 다룸.

예시: 

**Lossy Decomposition**

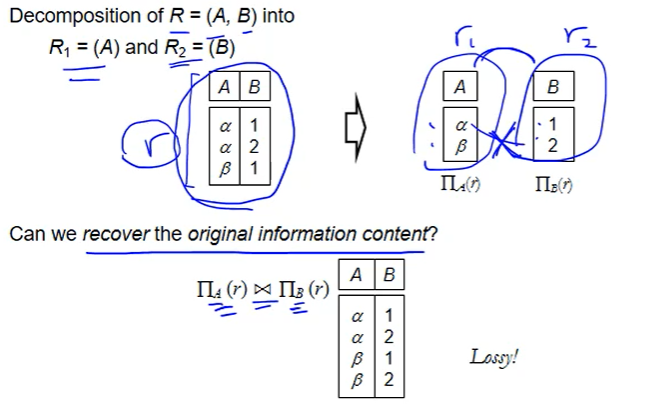
부주의한 decomposition은 정보의 손실을 유발할 수 있음 -> **Lossy Decomposition**

모든 정보가 오직 한 번만 표현되도록 schema를 decompose 해야 함.

예시: 

**Lossy Decomposition (cont.)**

Decompose한 Relation들을 join해서 원래의 정보를 찾아낼 수 있느냐?

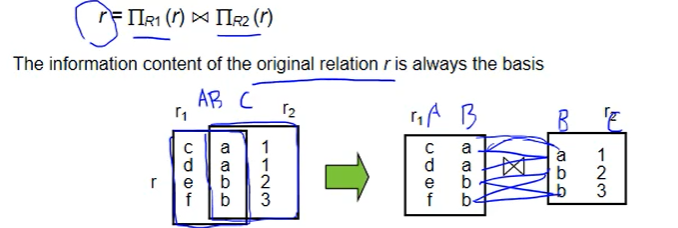


위의 예시 같은 경우 공유하는 attribute가 없기 때문에 cartesian product가 됨. 근데 A와 B의 연결 관계가 없어졌기 때문에 lossy.

**Lossless-join Decomposition**

r(R)을 {R1, R2}로 decompose하면, 항상 

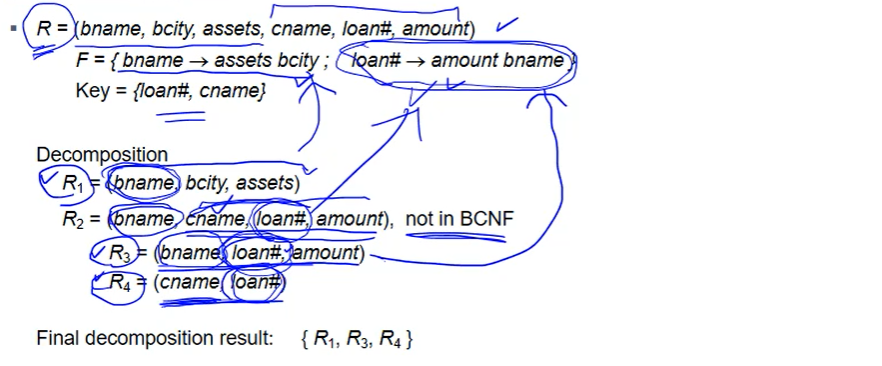
정의: Decomposition {R1, R2}는 다음을 만족해야 lossless-join decomposition이다.



Lemma: {R1, R2}는  이면 lossless join decomposition이다.

즉, decomposition이 공유하는 키가 최소 하나 있어야 함. 두 sub-schema 중 하나는 다른 sub-schema의 키를 가지고 있음.

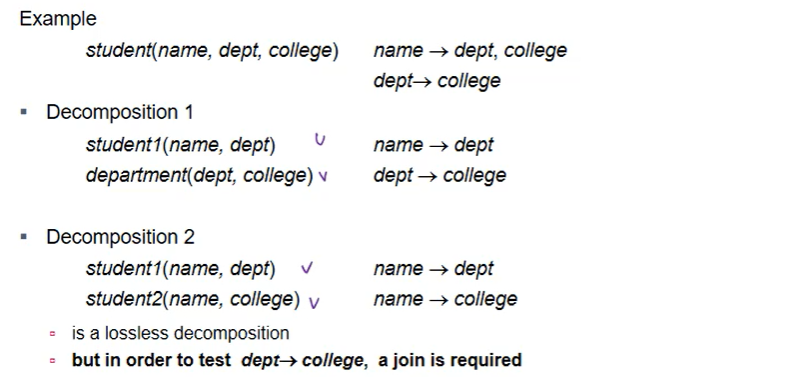
**BCNF Example**



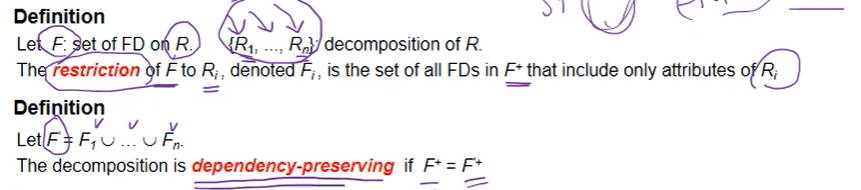
처음엔 bname을 공유로 나누고, 그 다음엔 loan#을 공유로 나눠서 최종 결과가 나옴.

**이거 시험 준비할 때 반대로 loan#을 기준으로 먼저 나눠도 같은 결과가 나오는지 확인하자.**

**Dependency Preservation**



Decomposition 2도 lossless이고 BCNF이지만 dept->college FD가 반영되지 못함.



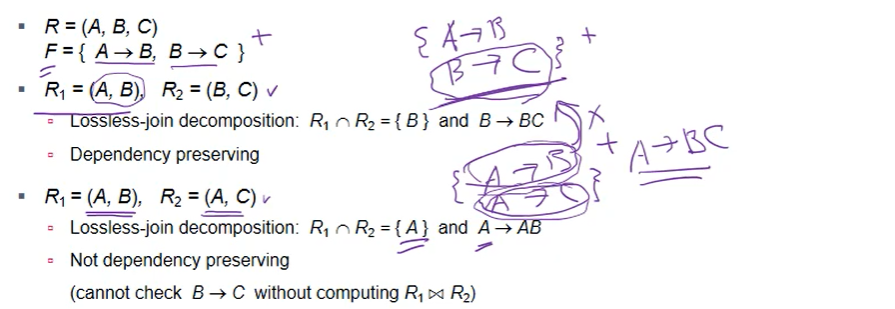
**Definition은 F의 closure가 같아야 한다는 점에 유의. F = F’이면 closure는 당연히 같음.**

동기: 각각의 테이블의 restriction F\_i만 확인하면 F 전체가 enforce 되었다는 것을 보장하려고 함.

Join은 비싼 연산임.

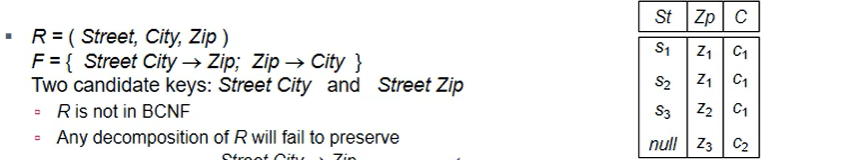
SQL은 superkey를 제외하고 따로 FD를 확인할 직접적인 방법을 제공하지 않음.

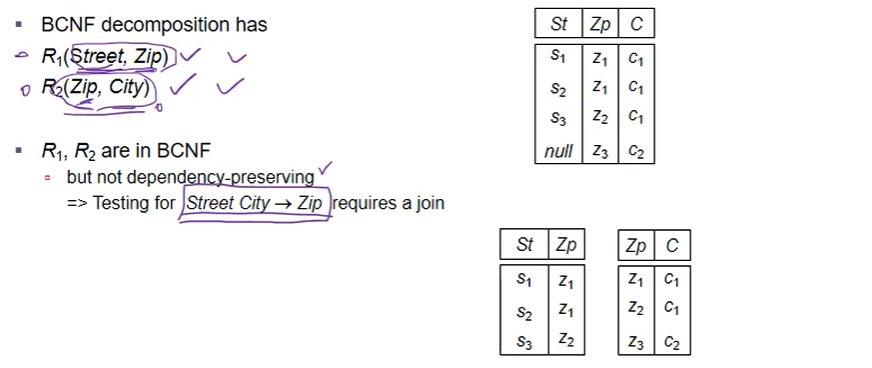
**Example**



**BCNF and Dependency Preservation**

BCNF는 항상 가능한데, 그것이 dependency preservation을 하는 경우는 항상 가능하지 않을 수도 있음.

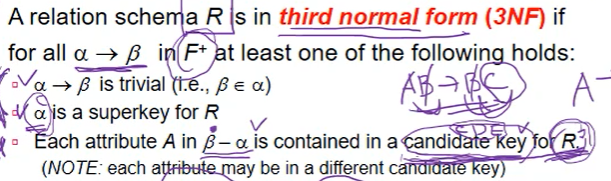




위의 예제는 BCNF는 되지만 dependency는 지켜지지 않음.

따라서 완화된 normal form이 필요함.

**Third Normal Form**

****

기본적으로 BCNF와 같은 정의를 가지고 있음.

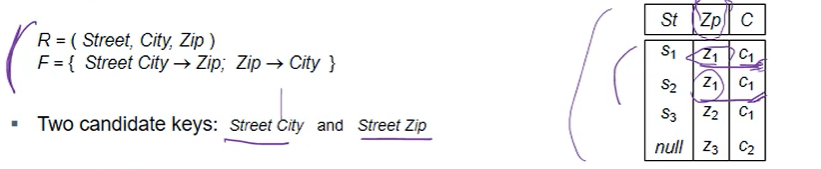
다른 점: b – a의 각각 Attribute A는 R의 candidate key에 속함. (각각의 attribute는 서로 다른 candidate key에 있을 수도 있음)

약간의 중복을 허용함. 그래서 FD는 join 없이 체크 가능.

항상 3NF로 만들 수 있는 lossless-join과 dependency-preserving decomposition이 존재함.

**BCNF는 항상 3NF임.**

**Example**



R은 3NF임. 

하지만 BCNF는 아님.

이 스키마에 약간의 중복이 있음.

- information의 반복 (e.g. z1과 c1의 관계)

- null value를 사용해야 함 (e.g. z3, c2의 관계를 나타낼 때 St에 연관된 값이 없는 경우)

**Comparison of BCNF and 3NF**

3NF는 항상 lossless, dependency preserving한 분해를 할 수 있음.

BCNF는 항상 lossless이지만 dependency는 지켜지지 않을 수도 있음.

**Design Goals**

Relation schema R with FD F을 R1, R2, …, Rn으로 분해할 때, 우리는

1. lossless decomposition
2. no redundancy
3. dependency preservation

을 원함.

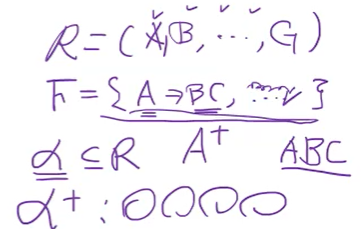
처음에는 BCNF, lossless join, dependency preservation을 달성하려고 함.

근데 이게 안되면, dependency preservation의 부족함을 받아들이거나 3NF를 사용함으로써 redundancy가 생기는 것을 받아들여야 함.

**Closure of Attribute Sets**

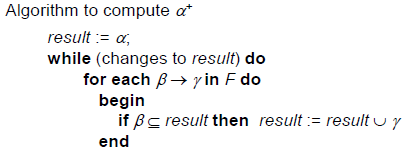
attribute들의 집합 a에 대해서, a의 closure는 F의 FD로부터 도출되는 모든 attribute의 집합.



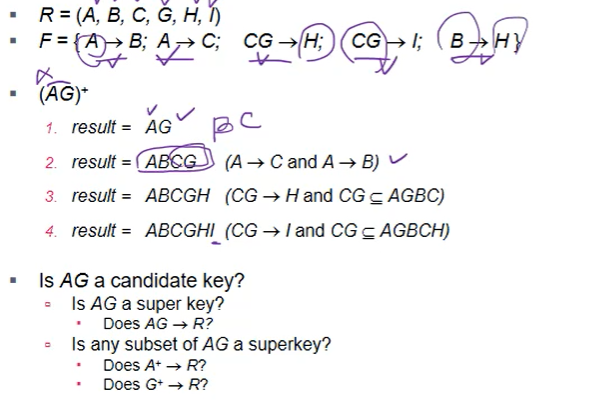


Imply, derive의 두 가지 경우가 있음.

Derived된 set = Armstrong 규칙 사용하면 Imply된 set은 sound 하다. F로부터 F+를 구할 수 있는 것이 Completeness



**Example**



AG는 super key. A랑 G는 super key가 아니므로 AG는 candidate key임.

**Use of Attribute Closure**

1. super key 체크: a+를 계산해서 a+가 R을 포함하는지 확인.
2. FD 체크: a->b가 F+에 포함되는지 확인. b가 a+에 속하는지 확인하면 됨.
3. F의 closure의 계산: 스키마 R의 모든 subset에 대해 closure를 구하면 F+가 됨.

**Testing for BCNF**

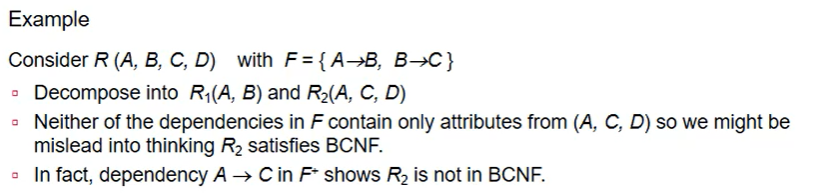
a->b가 BCNF를 위반하는지 체크함.

1. a+를 계산.
2. a+가 R의 모든 attribute를 포함하는지 확인 (그러면 R의 superkey)

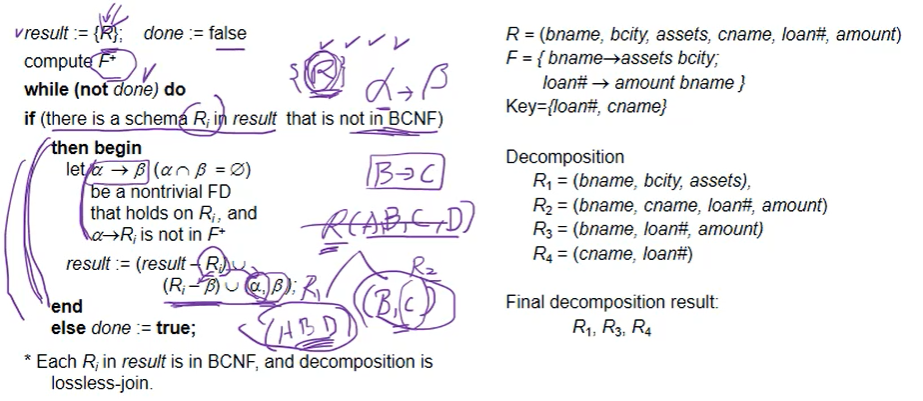
R이 BCNF인지 체크함.

1. F+를 구하지 않고 F의 FD만 BCNF를 위반하지 않으면 됨. 그러면 F+도 BCNF를 위반하지 않음이 증명되어 있음.

하지만 R의 decomposition을 체크하려면 F+를 써야함. (이 경우 F는 부정확)



**BCNF Decomposition Algorithm**



**Overall Database Design Process**

처음 R은 주어졌다고 가정함.

R은 다음과 같은 것들이 될 수 있음

1. 모든 attribute를 포함하는 single relation

- Universal Relation이라고 불림.

- Normalization으로 R을 더 작은 relation들로 나눔.

2. ER 다이어그램을 테이블의 집합으로 만들 때 생성될 수도 있음.

3. 우리가 normal form을 테스트할 때 사용하는, relation들의 ad hoc 디자인의 결과일 수도 있음.

**Denormalization for Performance**

Redundancy를 피하고 싶음. 하지만 performance를 위해 가끔씩은 비정규화를 사용할 수도 있음.

대체 1: denormalized된 relation을 사용함.

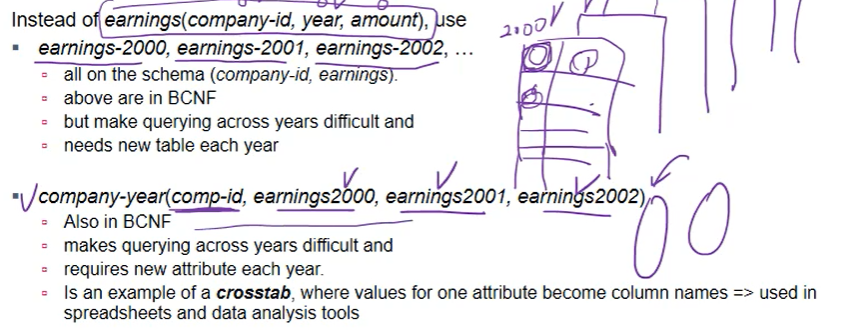
- lookup이 빠름, update에 더 많은 공간과 실행 시간이 필요함. 프로그래머들이 더 coding을 해야하고 에러의 확률이 있음. 얘는 r1과 r2를 보존하지 않고 r만 가지고 있음.

대체 2: materialized view를 사용함.

- DBMs에서 가상의 view를 만드는데, 단점은 1번과 모두 같고 대신 DBMS가 해주기 때문에 프로그래머가 할 일은 없고 에러의 확률을 피해줌.

**Other Design Issues**

DB 디자인의 어떤 측면들은 normalization으로 잡히지 않을 수도 있음.



이번 학기에 8장 뒷 부분은 다루지 않기로 함.