Design Theory: Functional Dependencies and Normal Forms, Part II

Instructor: Shel Finkelstein

Reference:
A First Course in Database Systems,
3rd edition, Chapter 3

Important Notices

- Lab4 assignment is due on **Sunday, June 2**, by 11:59pm.
  - Subject of Lab4 is Lecture 10 (Application Programming).
  - Lab4 has been/will be discussed at Lab Sections.
  - Your solution should be submitted via Canvas as a zip file.
    - Canvas is used for both Lab submission and grading.
    - Late Lab Assignments will not be accepted.
    - Be sure that you post the correct file!
  - Load file for Lab4 has been posted to Piazza.
    - You <u>must</u> use load file to do Lab4.
    - Load data helps with testing, but we won't post query solutions.
- See Small Group Tutoring website for LSS Tutoring with Chandler Hawkins.

Important Notices (Final)

CMPS 182 Final Exam is on **Monday June 10, 4:00 – 7:00pm**, in our usual classroom.

- **No** early/late Finals, **no** make-up Finals.
- **No** devices.
- Includes a Multiple Choice Section and a Longer Answers Section.
  - Bring Red Scantron sheets (ParSCORE form number f-1712) sold at Bookstore, and #2 pencils for Multiple Choice Section.
    - Ink and #3 pencils don't work.
- Covers entire quarter, with slightly greater emphasis on second half of quarter.
- You may bring in one double-sided 8.5 by 11 sheet, with anything that you can read unassisted printed or written on both sides of the paper.
  - **No sharing** of sheets is permitted.
  - Include name on top right of sheet. Sheets will be collected with Finals.
- You **must** show your UCSC ID at end of Final.
- Will post Practice Final from Spring 2017 (2 Sections) on Piazza.

Important Notices

- Gradiance #5 was assigned on Tuesday, May 28, and is due on **Friday, June 7 by 11:59pm**.
  - You should have enough information to complete this Gradiance Assignment by Friday, May 31.
  - Some of the questions may be difficult.

Spring 2019 Student Experience of Teaching Surveys - SETs are now open, and SETs close on Sunday June 9 at 11:59pm.

- SETs is the new term for campus-wide student course evaluations.
- Instructors **are not** able to identify individual responses.
- Constructive responses help improve future courses.

Normal Forms

Given a relation schema, we want to understand whether it is a good design or a bad design.

- Intuitively, a good design is one that does not store data redundantly, and does not lead to anomalies.

If we know that rank determines salary_scale, which is a better design?

Employees(eid, name, addr, rank, salary_scale)

OR

Employees2(eid, name, addr, rank)
Salary_Table(rank, salary_scale)

Remember that sometimes database designers **may choose** to live with redundancy in order to improve query performance. But then they'll have to cope with anomalies, which can be difficult.

First Normal Form (1NF)

- A relation schema is in *first normal form (1NF)* if the type of every attribute is atomic.
- Very basic requirement of the relational data model.
  - Not based on FDs.
  - Every other Normal Form we'll discuss assumes 1NF.

Example:

R(ssn: char(9), name: string, age: int)

- All our examples so far have been in 1NF.

Example of a non-first normal form relation:

R(ssn: char(9), name: Record[firstname: string, lastname: string], age: int, children: Set(string))

Second Normal Form (2NF)

- Not particularly important
  - We won’t discuss this.
  - (Neither does the textbook.)

Keeping FDs Simple

- We proved in previous Lecture that:

  If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

  and: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$

- So from now on, we'll assume that right side of every FD is a <u>single</u> attribute.
  - For example, instead of writing $AC \rightarrow BDE$, we will write:

    $AC \rightarrow B$, $AC \rightarrow D$, $AC \rightarrow E$

Boyce-Codd Normal Form (BCNF)

- Let R be a relation schema, $\mathcal{F}$ be a set of FDs that holds for R, with A as an attribute in R, and X as a subset of the attributes in R.

- R is in *Boyce-Codd Normal Form (BCNF)* if
  – For every FD X → A in $\mathcal{F}$, at least one of following is true:
    - X → A is a trivial FD (i.e., A ∈ X) or,
    - X is a superkey.

- BNCF is desirable for avoiding redundancy.
  – Recall our Employees2/Salary_Table example.

Is this relation in BCNF?

A	B	C
a1	b1	c1
a1	b2	c1

- The only functional dependency given is A → C.
- (to fill in)

Is this relation in BCNF?

A	B	C
a1	b1	c1
a1	b2	c1

- The relation is <u>not</u> in BCNF because:

  $A \rightarrow C$ is not a trivial FD and A is not a superkey.

- Given that $A \rightarrow C$, we can infer that C value of second tuple is also c1.
- But note that c1 is redundantly stored twice.

Third Normal Form (3NF)

- Let R be a relation schema, $\mathcal{F}$ be a set of FDs that holds for R, with A as an attribute in R, and X as a subset of the attributes in R.

- R is in *third normal form (3NF)* if
  - For every FD $X \rightarrow A$ in $\mathcal{F}$, at least one of following is true:
    - $X \rightarrow A$ is a trivial FD (i.e., $A \in X$), or
    - X is a superkey, or
    - A is part of some key of R.

- Note that red condition says that A is part of <u>some key</u> for R, <u>not some superkey</u> for R.

BCNF/3NF Example 1

Consider R(<u>A, B, C</u>, D)

with FD: $A \rightarrow D$

Note: Trivial FDs and FDs based on Primary Keys are implicit, and are often not listed, because 3NF obviously covers all of those.

- Is it in BCNF?
- Is it in 3NF?

A	B	C	D
a1	b1	c1	d1
a1	b2	c2	d1
a1	b2	c3	d1
a2	b2	c3	d2

BCNF/3NF Example 2

Now consider R(<u>A, B, C</u>, D)

with FD's: A → D, and D → A.

– Note that BCD is also a key for R.

- Is it in BCNF?
- Is it in 3NF?

A	B	C	D
a1	b1	c1	d1
a1	b2	c2	d1
a1	b2	c3	d1
a2	b2	c3	d2

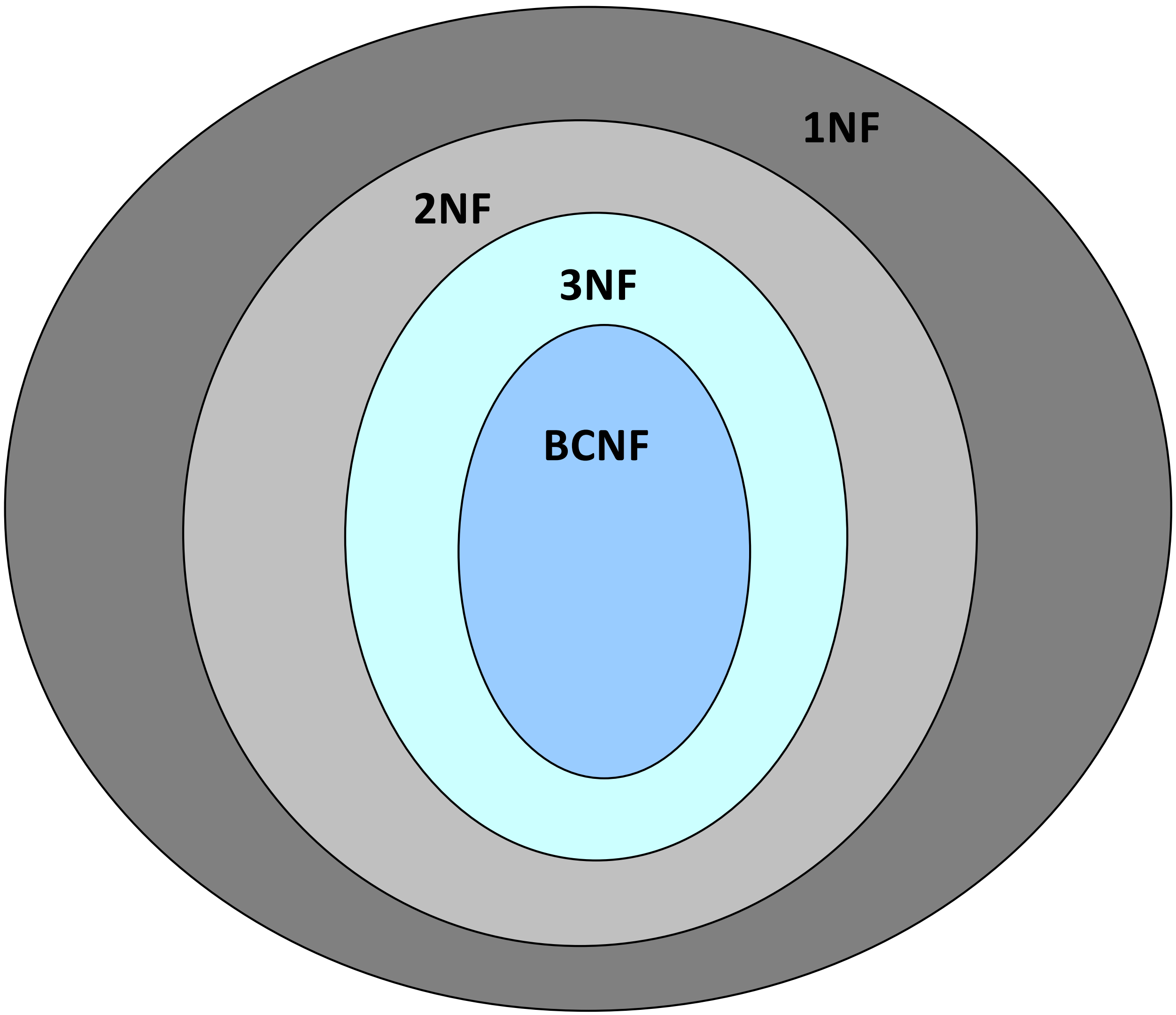
- Note that there is still redundancy in R, even though it is in 3NF!

BCNF and 3NF

- By definition, any BCNF relation must also be a 3NF relation.
- Definition says:

  … if at least one of the following holds for each FD X → A:
    - X → A is a trivial dependency (i.e., A ∈ X). BCNF, 3NF
    - X is a superkey. BCNF, 3NF
    - A is part of some key of R. 3NF
- However, a 3NF relation is <u>not</u> always in BCNF.
  - Example 2 is an example of a 3NF relation that is **not** in BCNF.

Relationships Among Normal Forms – The Big Picture



BCNF/3NF Example 3

Company_Info(<u>emp</u>, dept, manager)

dept → manager

Is it in BCNF?

Is it in 3NF?

BCNF/3NF Example 4

R(<u>city, street</u>, zip)

    city, street → zip

    zip → city

The above FDs are true of most post office policies. Note that a city may have <u>multiple</u> zips, but a zip must be in a <u>single</u> city.

Is it in BCNF?

Is it in 3NF?

- Despite 3NF, there can be Redundancy: The association of a zip with a city could appear in multiple records of R.
- **So although R is in 3NF, there can be Anomalies:**
  - zip → city. So if the city is changed in one (city, street, zip) record, but is <u>not</u> changed for another (city, street, zip) record that has the same zip, that's an anomaly.

BCNF/3NF Example 5

Customers(ssn, name, address)

ssn → name

ssn → address

Is it in BCNF?

Is it in 3NF?

Algorithm for Testing Whether a Relation is in BCNF using Attribute Closure

Given R and $\mathcal{F}$, determine whether R is in BCNF.

- For each FD $X \rightarrow Y \in \mathcal{F}$ such that $Y \not\subseteq X$ (i.e., the FD is non-trivial), compute X^+.
  - If every such X is a superkey (i.e., X^+ = attr(R)), then R is in BCNF.
  - If there is a set X of attributes such that $X^+ \neq$ attr(R), then R is not in BCNF.

Examples: BCNF Testing

- CompanyInfo(emp, dept, manager)
  - emp $\rightarrow$ dept, dept $\rightarrow$ manager
  - $dept^+ \neq$ attr(CompanyInfo). Hence CompanyInfo **is not** in BCNF.

- Customers(ssn, name, address)
  - ssn $\rightarrow$ name
  - ssn $\rightarrow$ address
  - $ssn^+ =$ attr(Customers) Hence Customers **is** in BCNF.

- R(city, street, zip)
  - city, street $\rightarrow$ zip
  - zip $\rightarrow$ city
  - $zip^+ \neq$ attr(R) Hence R **is not** in BCNF.

More on BCNF

Is R(A,B) is in BCNF?

- **Fact**: Any binary relation schema is in BCNF. (Why?)

How can we “improve” a relation that is not in BCNF?

- **Approach**: Decompose (“break up”) R into smaller relations so that each smaller relation is in BCNF.
- We did this when we decomposed Employees, separating out Salary_Table because of FD: rank → salary_scale.

  Employees2(<u>eid</u>, name, addr, rank)

  Salary_Table(<u>rank</u>, salary_scale)

Decomposition of a Relation

A *decomposition of a relation R* is defined by sets of attributes $X_1, \ldots, X_k$ (which don't have to be disjoint) such that:

1. Each $X_i \subseteq attr(R)$
2. $X_1 \cup X_2 \cup \ldots \cup X_k = attr(R)$

For a decomposition, we will write $\pi_{X_i}(R)$ as R_i, with instances of R written as r and instances of R_i written as r_i.

Examples:

- CompanyInfo(emp, dept, manager)
  - R_1(emp, dept), R_2(dept, manager)
- R(A,B,C,D,E,F,G)
  - R_1(A,C), R_2(A,B,C,D), R_3(C,D,E,F,G)

Goals for Redesigning Schema Using A Decomposition

1. The decomposition Eliminates Anomalies.
2. The decomposition doesn't lead to any "extra data" (that was not in instance r) when the r_i's are re-joined back together.
   - Such Decompositions are called Lossless Join decompositions.
   - Why must the Natural Join of all the r_i's always give at least all the data that was in r?
3. Dependency Preservation: (not required for Final)
   - The FD's on R_i are the FD's in $\mathcal{F}^+$ that mention only attr(R_i).
   - The decomposition is Dependency-Preserving if when the R_i's are re-joined back together, the FD's that were on the R_i's imply all of the original FD's in $\mathcal{F}$.

Are All BCNF Decompositions Good?

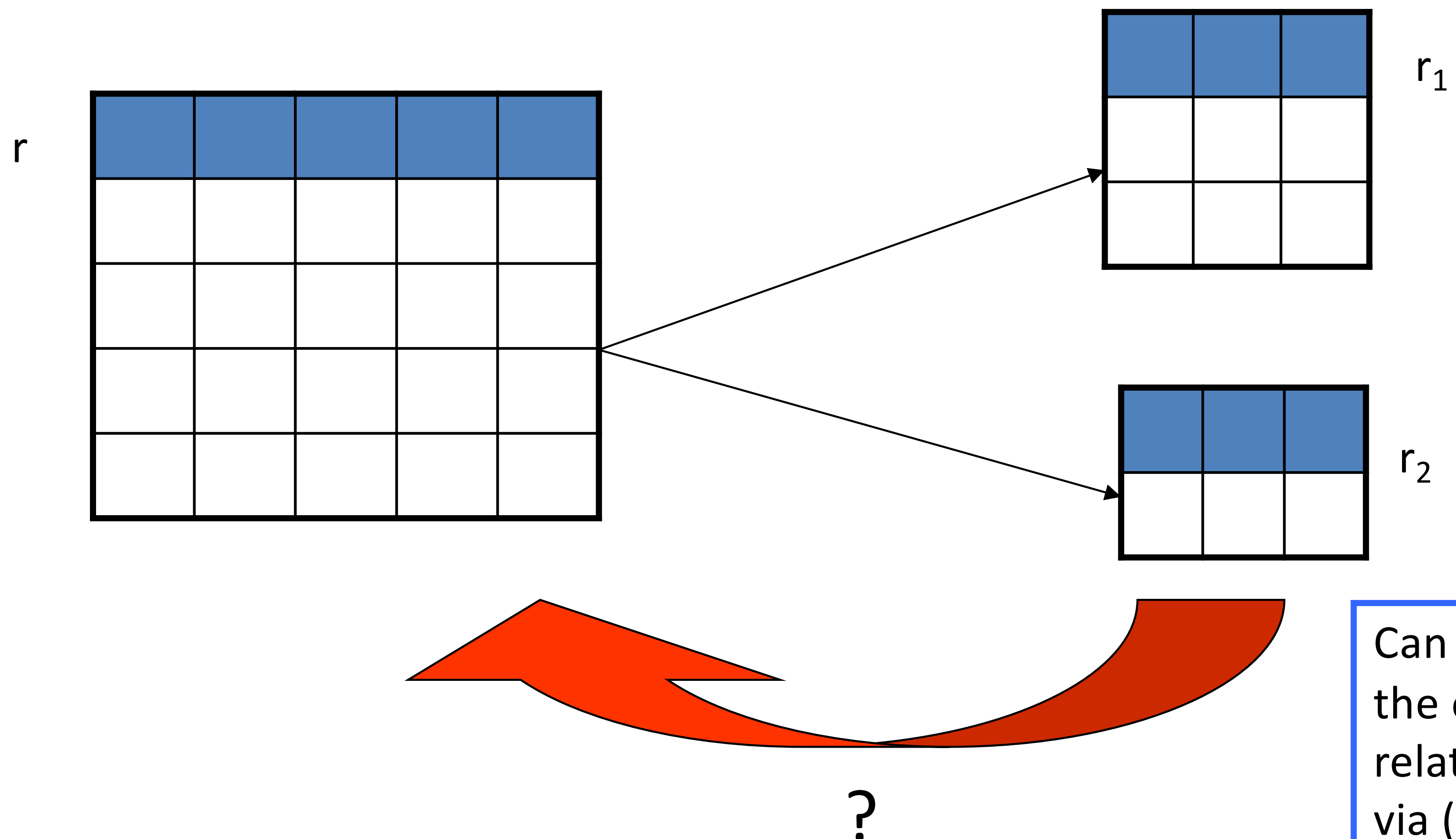
- Is it always possible to decompose R so that each smaller relation is in BCNF?

- YES
- One strategy: decompose R into a set of relation schemas $R_1, \ldots, R_k$ such that each R_i is a binary relation schema.

- Are all BCNF decompositions good?
- NO

Decomposing a Relation

- Suppose we have decomposed R into R_1 and R_2. Given an instance r of R, we decompose r into r_1 and r_2. Can we get back the original instance r by (Natural) Joining r_1 and r_2?



Can we reconstruct the original relation instance r via (Natural) Join?

Lossless Join Decomposition

In general, can we obtain r by Natural Joining r_1 with r_2 … with r_k?

- That is, must it always true for any instance r, that:
  $r = r_1 \bowtie r_2 \bowtie \ldots \bowtie r_k$ (Natural Join) ?

More precise definition:

- Let R be a relation schema and $\mathcal{F}$ be a set of FDs over R.
- A decomposition of R into *k* schemas, with attribute sets X_1 , …, X_k, is a *Lossless Join decomposition with respect to* $\mathcal{F}$ if:

For every instance r of R that satisfies $\mathcal{F}$, we have:

$$r = \pi_{X1}(r) \bowtie \ldots \bowtie \pi_{Xk}(r)$$
$$= r_1 \bowtie r_2 \bowtie \ldots \bowtie r_k$$

Lossless Join Example 1

- Let R(A,B,C) be a relation schema with no functional dependencies
- Is the decomposition of R into schemas $R_1(A,B)$ and $R_2(B,C)$ a Lossless Join decomposition?

Instance rx

A	B	C
a1	b1	c1
a1	b1	c2
a1	b2	c3

$\pi_{A,\,B}(rx)$

A	B
a1	b1
a1	b2

$\pi_{B,C}(rx)$

B	C
b1	c1
b1	c2
b2	c3

$\pi_{A,\,B}(rx) \bowtie \pi_{B,C}(rx)$

A	B	C
a1	b1	c1
a1	b1	c2
a1	b2	c3

Lossless Join Example 2

Instance ry

A	B	C
a1	b1	c1
a2	b1	c2

$\pi_{A,\ B}(ry)$

A	B
a1	b1
a2	b1

$\pi_{B,C}(ry)$

B	C
b1	c1
b1	c2

$\pi_{A,\ B}(ry)$ 1 $\pi_{B,C}(ry)$

A	B	C
a1	b1	c1
a1	b1	c2
a2	b1	c1
a2	b1	c2

Lossy!

- By projecting on $R_1(A,B)$ and $R_2(B,C)$, some information may be lost (sometimes).
- We no longer know that (a1,b1,c2) did not exist in the original relation!
- Hence R_1 and R_2 is <u>not</u> a Lossless Join decomposition of R.

FD’s and Lossless Joins

- Let R(A,B,C) be a relation schema
- Is the decomposition of R into schemas $R_1(A,B)$ and $R_2(B,C)$ a Lossless Join decomposition if we know $B \rightarrow C$?
  - ry is not a legal instance, since it does not satisfy $B \rightarrow C$.
- But that doesn’t prove that $R_1(A,B)$ and $R_2(B,C)$ is a Lossless Join decomposition in the presence of the FD $B \rightarrow C$.
  - Is it Lossless?
  - Yes; see textbook, Sections 3.4.1 and 3.4.2 …
  - … or later slides in this lecture!!

Lossless Join Example 3

CompanyInfo(emp, salary, dept, manager)

emp → salary, dept, manager

dept → manager

- CompanyInfo is not in BCNF because of dept → manager.
- Let's decompose into R_1(emp, salary) and R_2(dept, manager).

Instance r of CompanyInfo:

(Bolt, 85K, Math, Tromb)

(Montgomery, 90K, Math, Tromb)

(Brandt, 88K, CS, Pohl)

Lossless Join Example 3 (cont'd)

- Decompose instance r of CompanyInfo(emp, salary, dept, manager)
  - r_1(emp, salary)
    (Bolt, 85K)
    (Montgomery, 90K)
    (Brandt, 88K)
  - r_2(dept, manager)
    (Math, Tromb)
    (CS, Pohl)
- $r_1 \bowtie r_2 = r_1 \times r_2$ has 6 tuples in it.
- But instance r had only 3 tuples in it!
  - Hence the decomposition of CompanyInfo(emp, salary, dept, manager) into R_1(emp, salary) and R_2(dept, manager) <u>is not</u> a Lossless Join decomposition.

A Necessary and Sufficient Condition for Lossless Join Decomposition

- We would like our decompositions to be Lossless, and we'd like to be able to decide when a decomposition is Lossless.

Let R be a relation and $\mathcal{F}$ be set of FDs that hold over R.

Fact: A decomposition of R into two relation schemas R_1 with attributes X_1 and R_2 with attributes X_2 is Lossless if and only if $\mathcal{F}^+$ contains either:

1. $X_1 \cap X_2 \rightarrow X_1$, or
2. $X_1 \cap X_2 \rightarrow X_2$

That is, the intersection of the attributes of R_1 and R_2 is a superkey of either R_1 or R_2

Testing Whether a Decomposition is a Lossless Join Decomposition

Fact: A decomposition of R into relation schemas R_1 and R_2 is Lossless if and only if $\mathcal{F}^+$ contains either:

1. $X_1 \cap X_2 \rightarrow X_1$, or
2. $X_1 \cap X_2 \rightarrow X_2$

That is, the intersection of the attributes of R_1 and R_2 is a superkey of either R_1 or R_2

- This **Fact** works only for decompositions into **two** relations.
  - And note that it's not the definition of Lossless Join Decomposition!
- "The Chase" (see textbook) is a procedural algorithm for checking whether any decomposition is a Lossless Join decomposition.
  - We won't cover "The Chase" in this class.

Two More Lossless Join Examples

- Decompose R(A,B,C) into R_1(A,B) and R_2(B,C), with $\mathcal{F}$ being the empty set.
  - Since B → AB and B → BC are not in $\mathcal{F}^+$, this decomposition is not a Lossless Join decomposition.

- CompanyInfo(emp, salary, dept, manager)

  emp → salary, dept, manager

  dept → manager

  CompanyInfo is not in BCNF.

  Decompose into R_1(emp, salary) and R_2(dept, manager)
  - Since FDs {} → emp,salary and {} → dept, manager are not in $\mathcal{F}^+$, this decomposition is not a Lossless Join decomposition.

A Final Lossless Join Example

Employees(<u>eid</u>, name, addr, rank, salary_scale)
with FD: rank → salary_scale

Decomposition:
Employees2(<u>eid</u>, name, addr, rank)
Salary_Table(<u>rank</u>, salary_scale)

Employees2 ∩ Salary_Table = {rank}
rank → attr(Salary_Table).
Therefore, the decomposition <u>is</u> Lossless.

Decomposition and Normalization [Not required for Final]

Given a relation schema and functional dependencies, it is always possible to decompose schema into a set of **BCNF** relations that:

- 1) *Eliminates Anomalies,*
- and is 2) a *Lossless Join* decomposition*.*
- However, the schema might not always be 3) *Dependency-Preserving.*

Given a relation schema and functional dependencies, it is always possible to decompose schema into a set of **3NF** relations that:

- is 2) a *Lossless Join* decomposition*,*
- and is *3) Dependency-Preserving*.
- However, the schema might not always 1) *Eliminate Anomalies.*