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

- Final Exam is on Wednesday, March 22, noon-3pm in our usual classroom.
  - Final is Cumulative, with more focus on second half of quarter.
  - Please bring a red Scantron sheet (ParSCORE form number f-1712) sold at the Bookstore, and #2 pencils. (Some questions will be multiple choice.)
    - Ink and #3 pencils <u>don't</u> work.
  - You may bring a single two-sided 8.5" x 11" sheet of paper with as much info written (or printed) on it as you can fit and read unassisted, just as for the Midterm.
    - No sharing of these sheets will be permitted.
  - You must show your UCSC id when you turn in your Final and Scantron.
  - The Final from Fall 2016 has been posted on Piazza (Resources → Exams). Answers to that Final will be posted during the last week of classes.
- Gradiance Assignment #5 (on Functional Dependencies and Normal Forms) is due by Friday, March 17, 11:59pm.

More Important Notices

- Lab4 assignment was posted on Monday, Feb 27.
  - Due by Sunday, March 12, 11:59pm (2 weeks).
  - Lab4 focusses on material in Lecture 10 (Application Programming), including JDBC and Stored Procedures/Functions.
  - If you don't attend Lectures and Labs, you probably will find Lab4 difficult.
- There will be Lab Sections during the last week of classes.
  - These Lab Sections are an opportunity go over the answers to Lab4 and other Labs, or ask questions about other course material.
- Online course evaluations began Monday, March 5, and run through Sunday, March 19 at 11:59pm.
  - 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(<u>eid</u>, name, addr, rank, salary_scale)

OR

Employees(<u>eid</u>, name, addr, rank)
Salary_Table(<u>rank</u>, 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.

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.)

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 $X \longrightarrow A$ in $\mathcal{F}$, at least one of following is true:
    - $X \longrightarrow A$ is a trivial FD (i.e., $A \in X$) or,
    - X is a superkey.

- BNCF is desirable for avoiding redundancy.
  - Recall our Employees/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 not in BCNF because
  - $A \longrightarrow C$ is not a trivial FD and A is not a superkey.

- Given that $A \longrightarrow C$, we can infer that C value of second tuple is also c1.
- But a1 and c1 are obviously redundantly stored.

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 has to be the part of <u>some key</u> for R, not some superkey for R.

BCNF/3NF Example 1

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

with FD: A → D

- 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

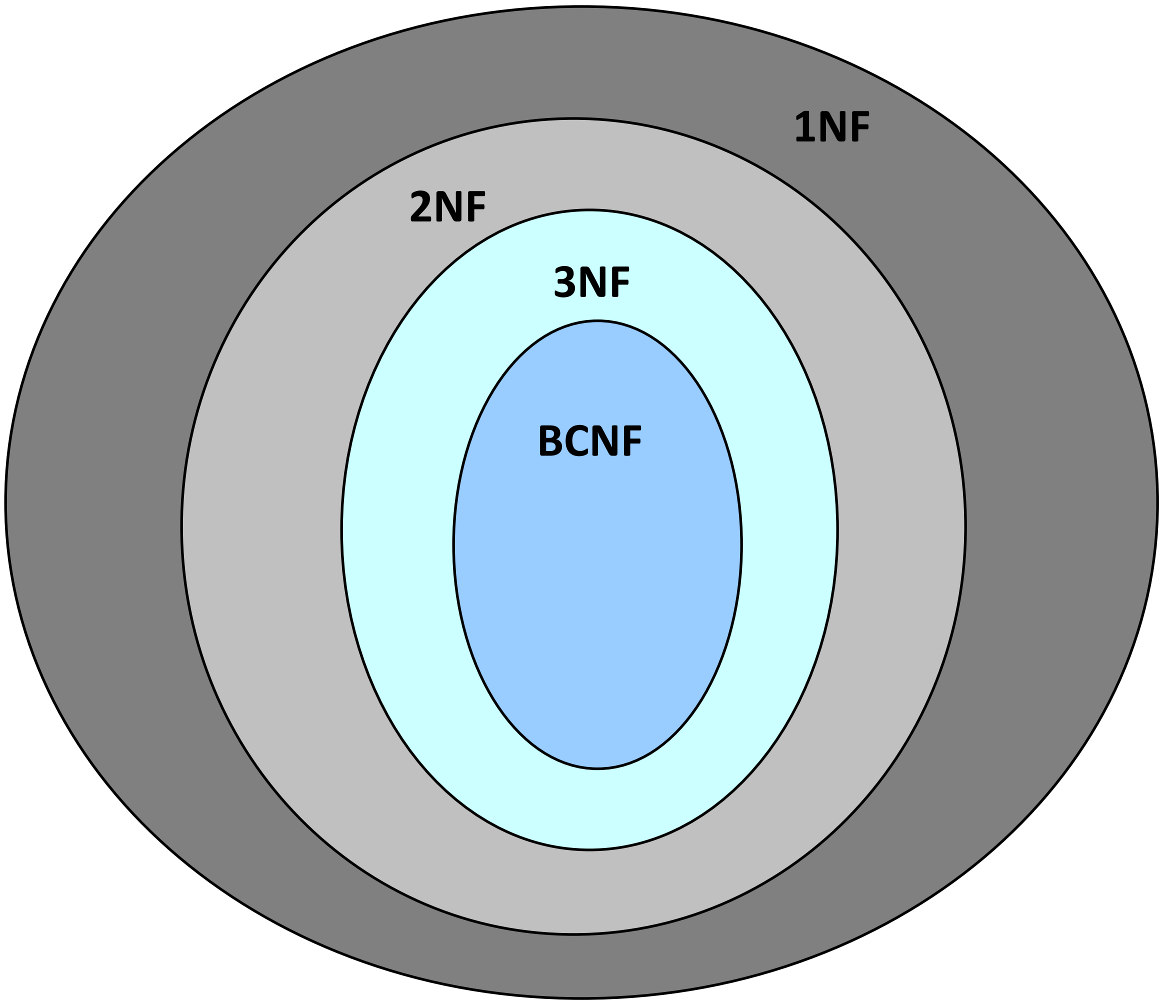
- By definition, a BCNF relation is also a 3NF relation.

- Definition says:

  … if at least one of the following holds for each FD $X \longrightarrow A$:
  - $X \longrightarrow A$ is a trivial dependency (i.e., $A \in X$). 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(emp, dept, manager)

emp → dept

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 multiple zips, but a zip is in a single 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 $Y \nsubseteq X$ (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 → dept, dept → manager
  - $dept^+ \neq attr(CompanyInfo)$. Hence CompanyInfo **is not** in BCNF.

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

- R(city, street, zip)
  - city, street → zip
  - zip → 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 $\rightarrow$ salary_scale.

  Employees(<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 π_{Xi} (R) as R_i, with instance 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:
   - 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 $\mathcal{F}$.

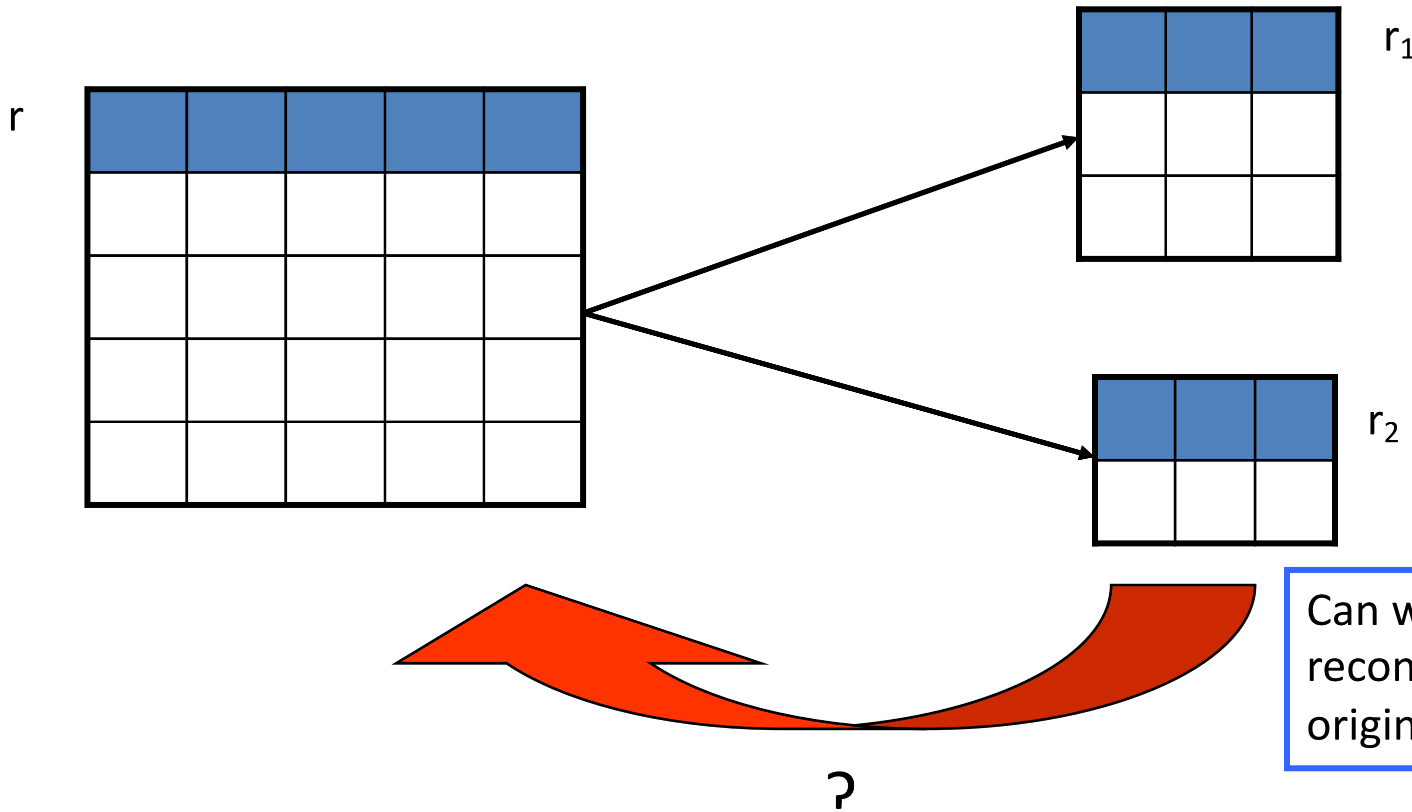
- Is it always possible to decompose R so that each smaller relation is in BCNF?

- YES
- One strategy: decompose R into a set of relation schemas R_1, …, 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 recover or reconstruct the original relation r?

Lossless Join Decomposition

In general, can we obtain r by 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 in general.
- We no longer know that (a1,b1,c2) does not exist in the original relation.
- Hence R_1 and R_2 is not 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 $B \rightarrow C$.
  - rx, however, is a legal instance with respect to $B \rightarrow C$.
- 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)

- r_1
  (Bolt, 85K)
  (Montgomery, 90K)
  (Brandt, 88K)

- r_2
  (Math, Tromb)
  (CS, Pohl)

- $r_1 \bowtie r_2 = r_1 \times r_2$ has 6 tuples
- That’s 3 more tuples than in r. Therefore the decomposition <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 relation schemas R_1 and R_2 is Lossless if and only if $\mathcal{F}^+$ contains either:

1. $R_1 \cap R_2 \rightarrow R_1$, or
2. $R_1 \cap R_2 \rightarrow R_2$

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 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. $R_1 \cap R_2 \rightarrow R_1$, or
2. $R_1 \cap R_2 \rightarrow R_2$

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

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 <u>is not</u> a Lossless Join decomposition.

- CompanyInfo(<u>emp</u>, 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 <u>is not</u> a Lossless Join decomposition.

A Final Lossless Join Example

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

Decomposition:

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

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

Decomposition and Normalization

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.*

- Let R be a relation and $\mathcal{F}$ be set of FDs that hold over R.
- Fact: A decomposition of R into relation schemas R_1 and R_2 is Lossless if and only if $\mathcal{F}^+$ contains either
  1. $R_1 \cap R_2 \rightarrow R_1$, or
  2. $R_1 \cap R_2 \rightarrow R_2$

- This fact provides you a criteria for checking whether a decomposition is a Lossless Join decomposition. But it does not tell you exactly how to check for this criteria.
- Is there a more procedural algorithm for checking whether a decomposition is a Lossless Join decomposition?

The Chase Algorithm

Input: A relation $R(a_1, \ldots, a_k)$. Its decomposition relation schemas $R_1, \ldots, R_n$ and a set $\mathcal{F}$ of FDs.

Output: Decides whether the decomposition is a Lossless Join decomposition.

1. Create a tableau T (i.e., a symbolic relation) according to R and $R_1, \ldots, R_n$.
   - Let $t = (a_1, \ldots, a_k)$. The "canonical tuple".
   - T is a relation of arity k with n tuples such that the ith tuple $t_i[R_i] = t[R_i]$.
   - Every other attribute value in T that is not among $t_i[R_i]$, where $1 <= i <= n$, is a fresh new value of a higher subscript.
2. Apply the FDs $\mathcal{F}$ of to T until no more FDs can be applied.
3. Return YES if the canonical tuple $(a_1, \ldots, a_k)$ is in T. Return NO otherwise.

Example

- R(A,B,C) into R_1(A,B) and R_2(B,C) with $\mathcal{F}$ being the empty set.
- The tableau T is:

A	B	C
a1	b1	c2
a2	b1	c1

- Since $\mathcal{F}$ is the empty set, no FDs can be applied. The canonical tuple (a1,b1,c1) does not occur in the resulting tableau.
- Answer: NO. Not a Lossless Join decomposition.

- What does "apply the FDs $\mathcal{F}$ of to T" in step 2 mean?

Example

- CompanyInfo(emp, salary, dept, manager)
- R_1(emp, salary), R_2(dept, manager)
- $\mathcal{F}$ = { emp → salary, dept, manager, dept→manager }

- Tableau T:

emp	salary	dept	manager
e1	s1	d2	m2
e2	s2	d1	m1

- None of the FDs can be applied.
- The canonical tuple (e1,s1,d1,m1) does not occur in T.
- Answer: NO. Not a Lossless Join decomposition.

- Employees(<u>eid</u>, name, addr, rank, salary_scale)
- Decomposition: Employees(eid, name, addr, rank) Salary_Table(rank, salary_scale)
- $\mathcal{F}$ = { rank $\longrightarrow$ salary_scale }
- Tableau T:

eid	name	addr	rank	salary_scale
e1	n1	a1	r1	s2
e2	n2	a2	r1	s1

Apply rank $\longrightarrow$ salary_scale

eid	name	addr	rank	salary_scale
e1	n1	a1	r1	~~s2~~ s1
e2	n2	a2	r1	s1

When given a choice to replace s1 by s2 or s2 by s1, always replace with the value with a lower subscript.

eid	name	addr	rank	salary_scale
e1	n1	a1	r1	s2
e2	n2	a2	r1	s1

Apply rank $\longrightarrow$ salary_scale

eid	name	addr	rank	salary_scale
e1	n1	a1	r1	s1
e2	n2	a2	r1	s1

Apply the FDs until no more FDs can be applied.

- The canonical tuple (e1,n1,a1,r1,s1) occurs in T.
- Answer: YES. This is a Lossless Join decomposition.

- FDs can be applied in any order. The existence of the canonical tuple is agnostic to the order in which FDs are applied.

- The chase algorithm will always terminate since there is only a finite number of times one can replace a value with a value of a lower subscript.

An observation

- Given a relation R, an FD $X \rightarrow Y$ that holds over R, and $X \cap Y$ is empty, then the decomposition of R into R-Y and XY is lossless.
  - R_1 = R-Y, R_2 = XY
  - $R_1 \cap R_2$ = X, $X \longrightarrow Y$
  - Therefore, $R_1 \cap R_2 \longrightarrow R_2$
- We can apply this observation repeatedly.
  - Given a set of FDs $\mathcal{F}$, if R can be losslessly decomposed into R_1 and R_2 and, R_2 can be losslessly decomposed into R_3 and R_4, then the decomposition of R into relations R_1, R_3, and R_4 is also a lossless decomposition.

Algorithm for producing a BCNF Lossless Join decomposition of a relation schema R

- Input: R, $\mathcal{F}$
- Output: A lossless join decomposition of R into R_1, ..., R_k.
- Set D = {R}.
- While there is some R_i in D which is not in BCNF, do
  1. Find $X \rightarrow Y \in \mathcal{F}^+$ such that X is not a superkey for R_i and $Y^* \nsubseteq X$.
  2. Replace R_i by $R_i - Y$ and XY in D.

Examples

- R(A,B,C), $\mathcal{F}$={}.
  - R is in BCNF.

- R(A,B,C), $\mathcal{F}$={A→B}.
  - A→B violates 2nd condition of the BCNF definition.
  - Decompose R into R_1(A,C) and R_2(A,B).
  - R_1 and R_2 are each in BCNF. Done.

- R(city, street, zip), $\mathcal{F}$={city,street → zip, zip → street}
  - The 2nd FD violates the 2nd condition of the BCNF definition.
  - Decompose R into R_1(city,zip) and R_2(zip,street).
  - No more decompositions as R_1 and R_2 are each in BCNF. Done.

- CompanyInfo(emp, salary, dept, manager)
  $\mathcal{F}$={emp $\rightarrow$ salary, dept, manager, dept $\rightarrow$ manager}

  - The 2nd FD violates the 2nd condition of the BCNF definition.
  - Decompose CompanyInfo into R_1(emp, salary, dept) and R_2(dept, manager).
  - R_1 is in BCNF because $emp^+ = attr(R_1)$. Note that dept $\rightarrow$ manager does not apply to R_1.

A necessary and sufficient condition for Lossless Join decomposition

- We would like our decompositions to be lossless and 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 relation schemas R_1 and R_2 is lossless if and only if $\mathcal{F}^+$ contains either
  1. $R_1 \cap R_2 \rightarrow R_1$, or
  2. $R_1 \cap R_2 \rightarrow R_2$

Proof?

($\Leftarrow$) If $R1 \cap R2 \rightarrow R1$, then $R1 \bowtie R2 = R$.

Let t be a tuple in R. Clearly, $t[R1] \in R1$ and $t[R2] \in R2$ and therefore, $t \in R1 \bowtie R2$.
We have shown that $R \subseteq R1 \bowtie R2$.

Let t be a tuple in $R1 \bowtie R2$.
This means there exists a tuple $t1 \in R1$ and $t2 \in R2$ such that $t1 \bowtie t2 = t$.
Suppose $t \notin R$.
Since R1 is the projection of R on attributes of R1, we know there exists a tuple $t' \in R$ such that $t'[R1] = t1$ and $t'[R2] \neq t2$. (otherwise, $t = t'$ meaning that $t \in R$ and we are done.)
Similarly, there also exist a tuple t" in R such that $t''[R2] = t2$ and $t''[R1] \neq t1$ for the same reason as above.
Notice that t' and t" are distinct tuples but share the same values for attributes in $R1 \cap R2$.
However, $t'[R1] \neq t''[R1]$. Hence the FD is violated. Contradiction.

($\Rightarrow$) If $R1 \bowtie R2 = R$, then either $R1 \cap R2 \rightarrow R1$ or $R1 \cap R2 \rightarrow R2$
Assume that $R1 \cap R2 \nrightarrow R1$ AND $R1 \cap R2 \nrightarrow R2$, we will show that we can always construct a counterexample to show that $R1 \bowtie R2 \neq R$.
The basic idea is as follows.
Let R(A,B,C), R1(A,B), and R2(B,C) and R contains (a1,b1,c1) (a2,b1,c2) is a counter example. See the previous slide.