CSC 261/461 – Database Systems
Lecture 12

Spring 2018

Announcement

- Project 1 Milestone 2 due tonight!

- Read the textbook!
  - Chapter 8:
    - Will cover later; But self-study the chapter
  - Chapter 14:
    - Section 14.1 – 14.5
  - Chapter 15:
    - Section 15.1 – 15.4

Student Feedback

- http://www.cs.rochester.edu/courses/261/spring2018/ → Forms → Feedback Form

Student Feedback #1

- My schedule doesn't fit with any of the times. A Friday afternoon would be great. Or Tuesday, Thursday afternoon. Or can I just ask my questions and ask what happened at workshops during office hours?

- Yes. That’s fine.

- Workshops are meant to help you better.
- We know cases where students are too shy to come to office hours. In a group setting they usually perform better. Also, they can work as a team solving Problem Set problems.

Student Feedback #2

- Regarding today's (Feb. 19) announcement, I don't think laptops should be banned from class. I use my laptop as my primary means of taking notes, since I am much more organized digitally. My laptop is my only note-taking tool for all my classes and I find it much more useful than a normal notebook. …. .
- If the reason that laptops are banned is that they distract other students, then make a rule that people using laptops sit in the back, while those taking notes sit in the front. This gives people the freedom they should have in class without hurting other students. …

- If any of you have this same issue, please contact me after the class.

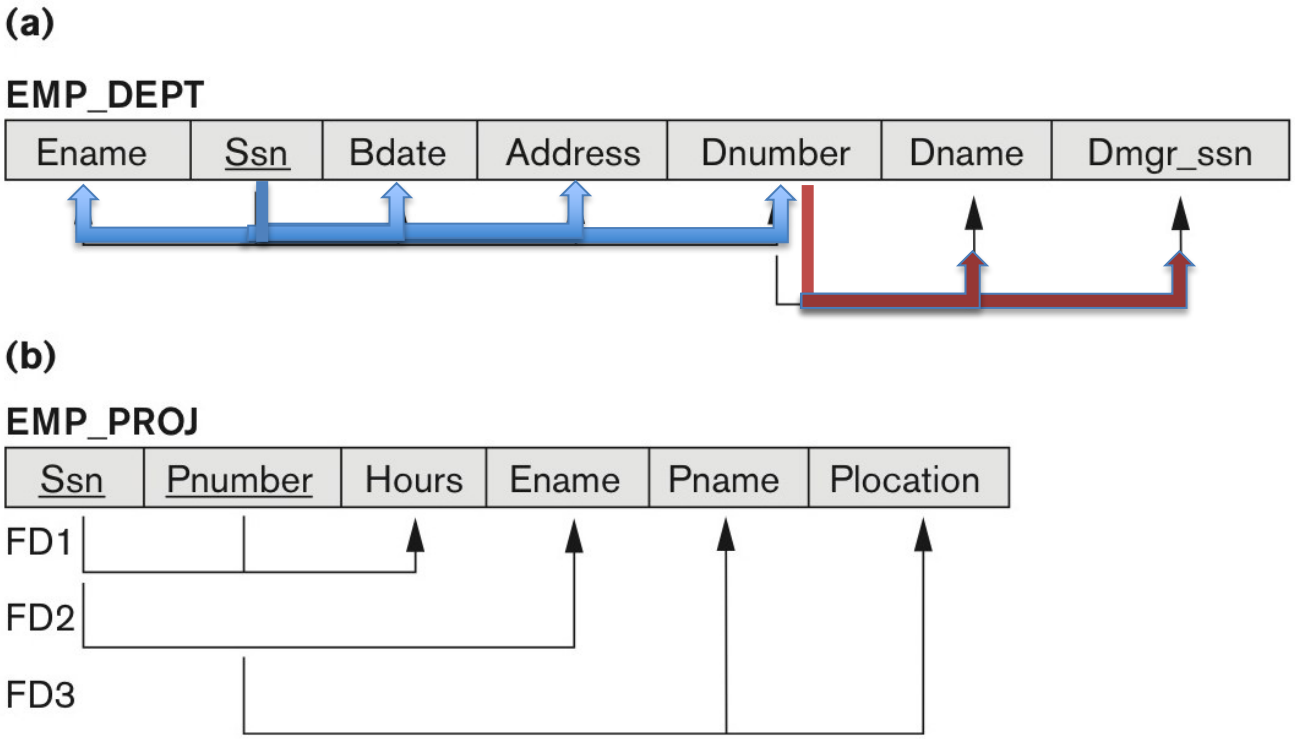
- Read this post:
- https://www.nytimes.com/2017/11/27/learning/should-teachers-and-professors-ban-student-use-of-laptops-in-class.html

Agenda

1. 2NF, 3NF and Boyce-Codd Normal Form

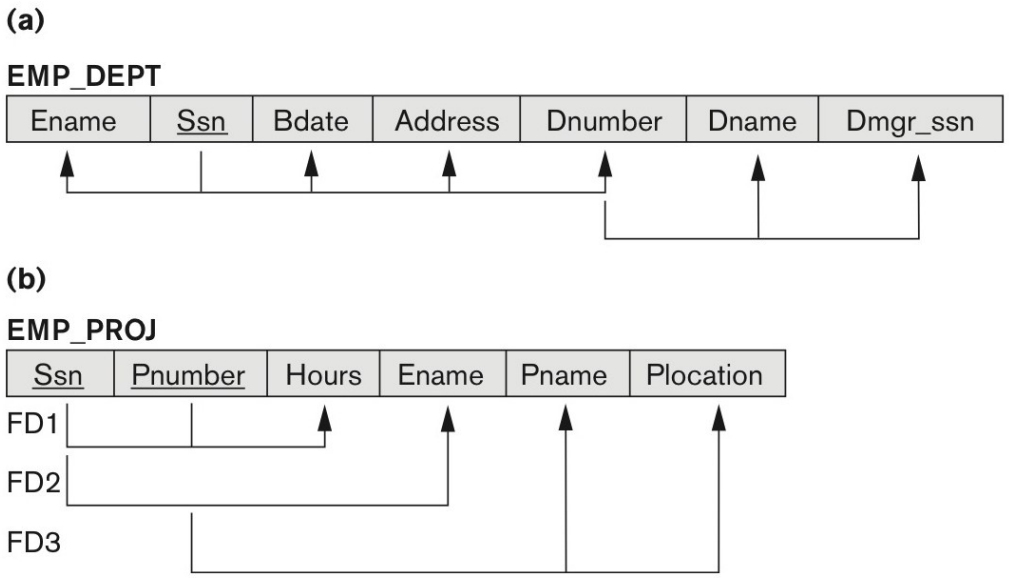
2. Decompositions

Functional Dependencies (Graphical Representation)



Prime and Non-prime attributes

- A **Prime attribute** must be a member of *some* candidate key
- A **Nonprime attribute** is not a prime attribute—that is, it is not a member of any candidate key.



Back to Conceptual Design

Now that we know how to find FDs, it's a straight-forward process:

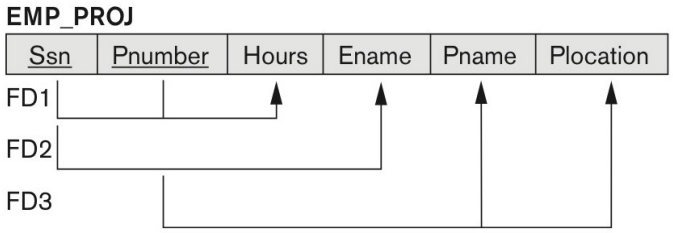
1. Search for "bad" FDs
2. If there are any, then *keep decomposing the table into sub-tables* until no more bad FDs
3. When done, the database schema is *normalized*

Boyce-Codd Normal Form (BCNF)

- Main idea is that we define "good" and "bad" FDs as follows:
  - X → A is a "*good FD" if X is a (super)key*
    - In other words, if A is the set of all attributes
  - X → A is a *"bad FD"* otherwise
- We will try to eliminate the "bad" FDs!
  - Via normalization

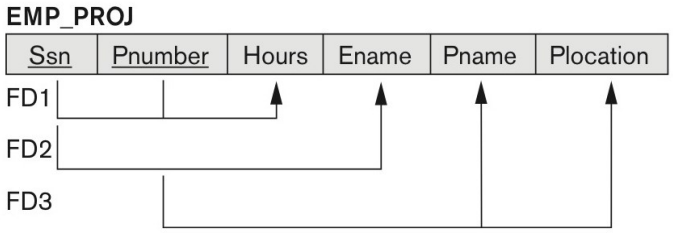
Second Normal Form

- Uses the concepts of **FDs, primary key**
- Definitions
  - **Full functional dependency:**
    - a FD Y → Z where removal of any attribute from Y means the FD does not hold any more



Second Normal Form (cont.)

- Examples:
  - {Ssn, Pnumber} → Hours is a full FD since neither
    - Ssn → Hours nor Pnumber → Hours hold
  - {Ssn, Pnumber} → Ename is not a full FD (it is called a partial dependency) since Ssn → Ename also holds

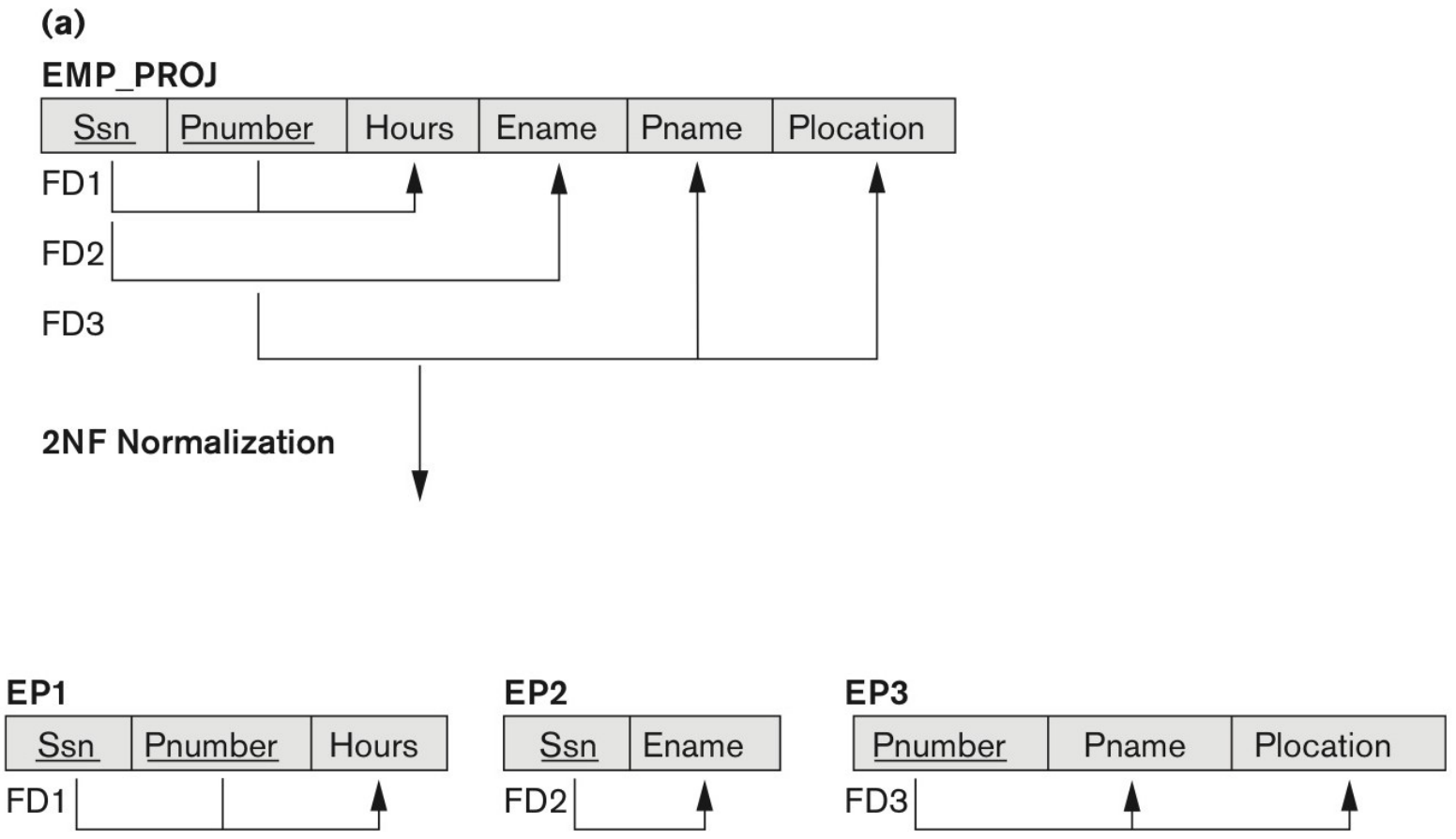


Second Normal Form (2)

- A relation schema R is in **second normal form (2NF)** if every non-prime attribute A in R is fully functionally dependent on the primary key

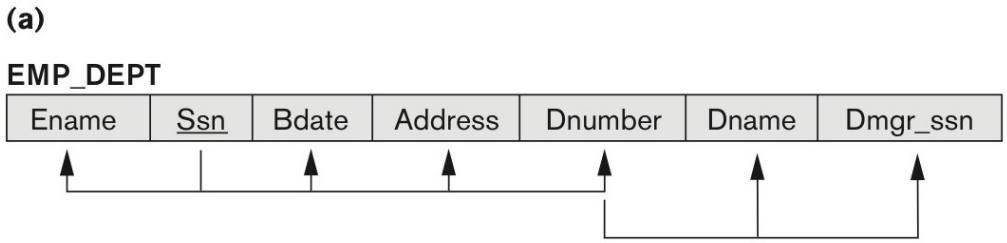
- R which is not in 2NF can be decomposed into 2NF relations via the process of 2NF normalization or “second normalization”

Normalizing into 2NF



Third Normal Form (1)

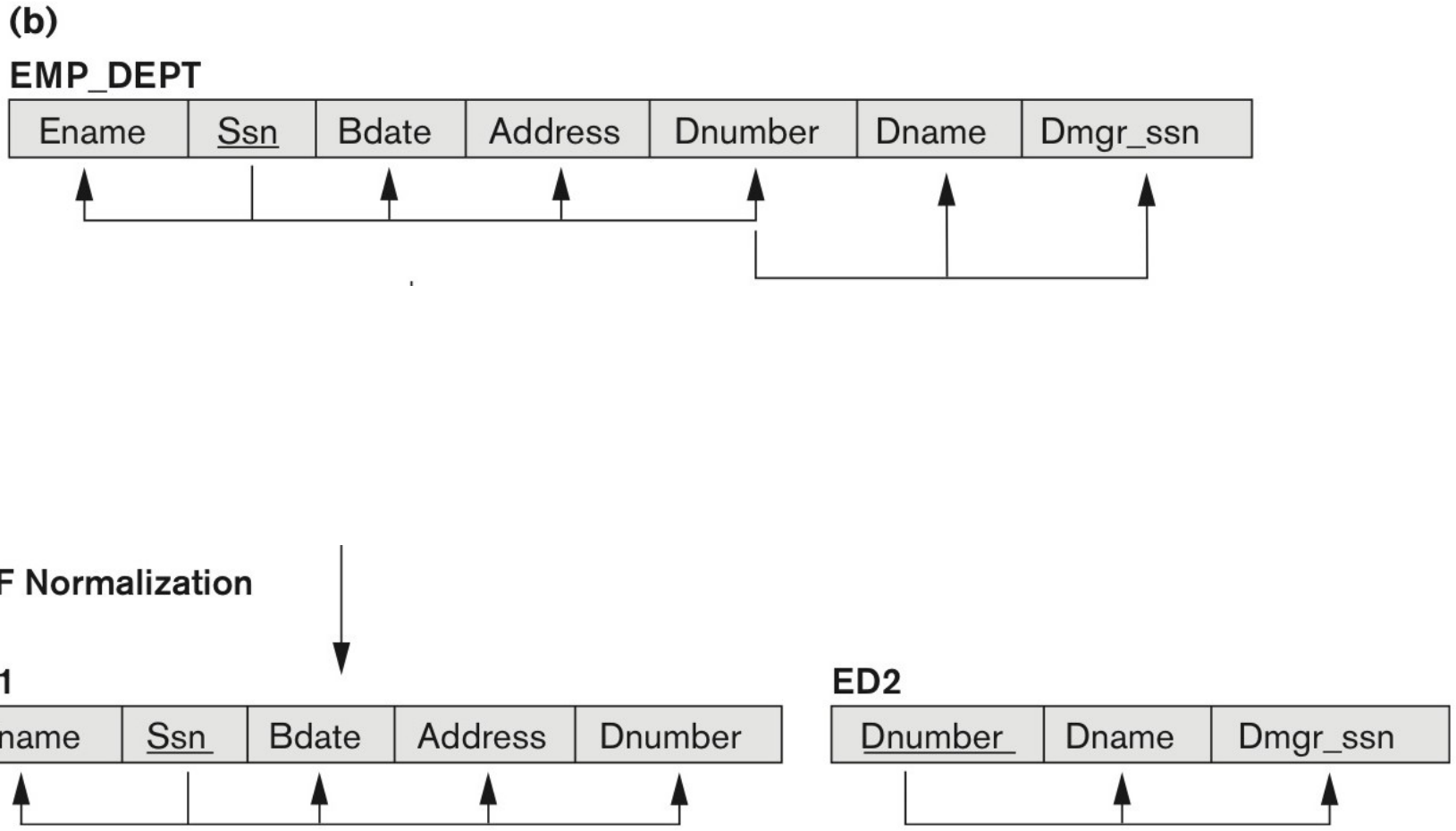
- Definition:
  - **Transitive functional dependency**:
    - a FD X → Z that can be derived from two FDs X → Y and Y → Z
- Examples:
  - Ssn -> Dmgr_ssn is a **transitive** FD
    - Since Ssn -> Dnumber and Dnumber -> Dmgr_ssn hold
  - Ssn -> Ename is **non-transitive**
    - Since there is no set of attributes X where Ssn → X and X → Ename



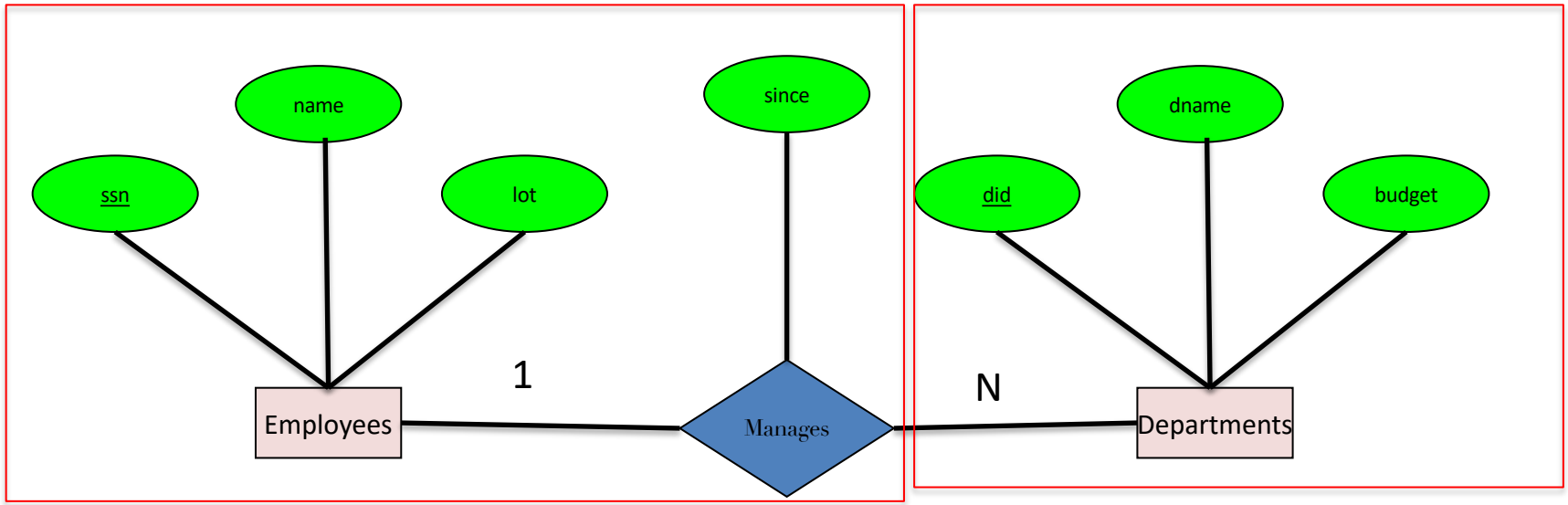
Third Normal Form (2)

- A relation schema R is in **third normal form (3NF)** if it is in 2NF *and* no non-prime attribute A in R is transitively dependent on the primary key
- R can be decomposed into 3NF relations via the process of 3NF normalization

Normalizing into 3NF



Now, Is it in 3NF?




```
CREATE TABLE Employee_Mgr( ssn char(11),
                           name varchar(30),
                           lot integer,
                           since date,
                           did integer,

PRIMARY KEY (did),
FOREIGN KEY (did) REFERENCES Departments (did)
ON DELETE RESTRICT,
)
```

Possible. But…

Our Next topic… It's an example of bad Functional Dependency

Answer: Is it in 2NF?

- Yes.
- Why?
- Every Non prime attribute is fully functionally dependent on primary key {did}

Answer: Is it in 3NF?

- No.
- Why?
- Non-prime attributes name, lot are transitively dependent on did
- did → ssn → {name,lot}

Remedy

- Take a new relation with {ssn, name, lot}

Remedy (Normalization)

- How to perform 1NF normalization?
  - Form new relations for each multivalued attribute
- How to perform 2NF normalization?
  - Decompose and set up a new relation for each partial key with it's dependent attributes.
- How to perform 3NF normalization?
  - Decompose and set up a new relation that includes the non-key attribute and every-other non-key attributes it determines.

Normal Forms Defined Informally

- 1st normal form
  - All attributes depend on **the key**
- 2nd normal form
  - All attributes depend on **the whole key**
- 3rd normal form
  - All attributes depend on **nothing but the key**

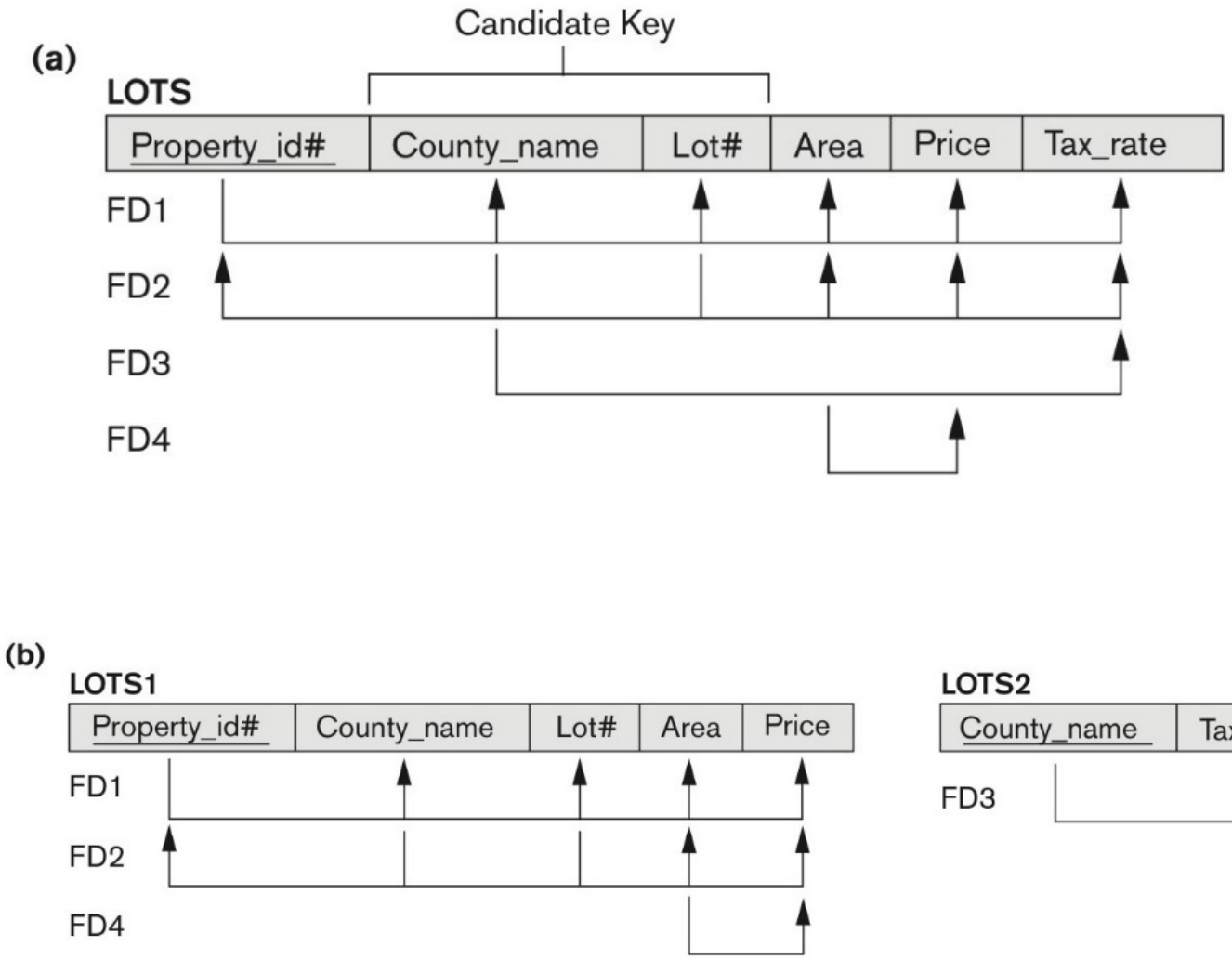
General Definition of 2NF and 3NF

(For Multiple Candidate Keys)

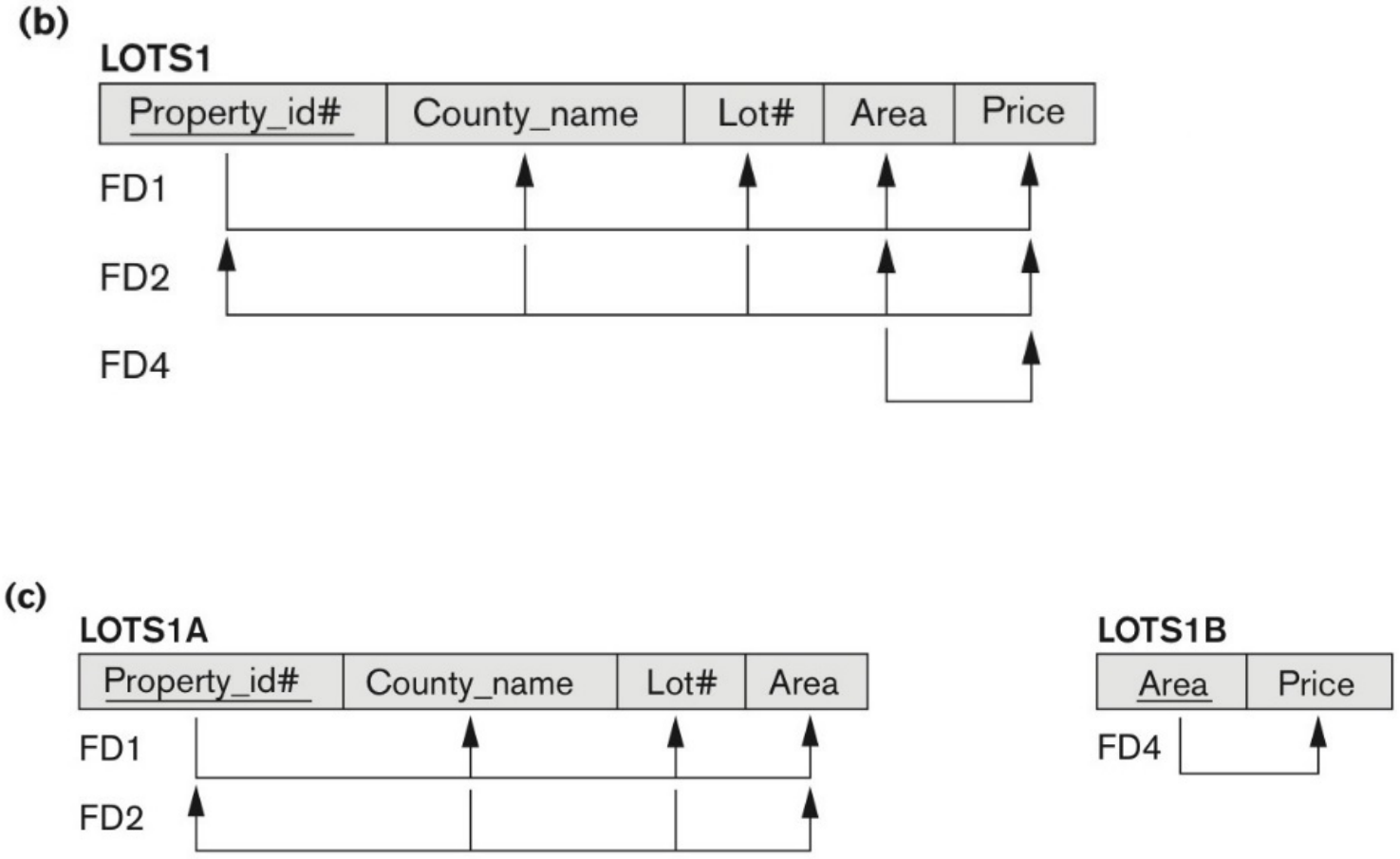
- A relation schema R is in **second normal form (2NF)** if every non-prime attribute A in R is fully functionally dependent on ***every*** key of R

- A relation schema R is in **third normal form (3NF)** if it is in 2NF *and* *no non-prime attribute A* in R is transitively dependent on ***any*** key of R

Normalization into 2NF



Normalization into 3NF



1. BOYCE-CODD NORMAL FORM

What you will learn about in this section

1. Conceptual Design
2. Boyce-Codd Normal Form
3. The BCNF Decomposition Algorithm

BCNF (Boyce-Codd Normal Form)

- A relation schema R is in Boyce-Codd Normal Form (BCNF) if whenever an **FD X → A** holds in R, then **X is a superkey** of R

- Each normal form is strictly stronger than the previous one
  - Every 2NF relation is in 1NF
  - Every 3NF relation is in 2NF
  - Every BCNF relation is in 3NF

General Definition of 2NF and 3NF

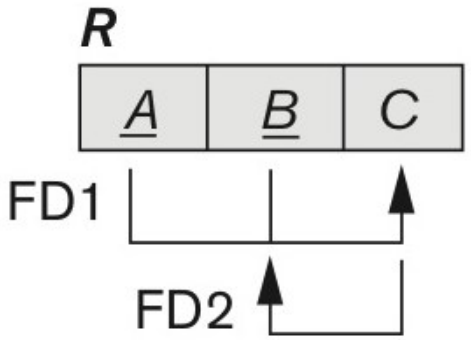
- A relation schema R is in **second normal form (2NF)** if every non-prime attribute A in R is fully functionally dependent on ***every*** key of R

- A relation schema R is in **third normal form (3NF)** if it is in 2NF *and* no non-prime attribute A in R is transitively dependent on ***any*** key of R

wait a sec… If A is a prime attribute, it's still in 3NF ?!

Yes! That's correct

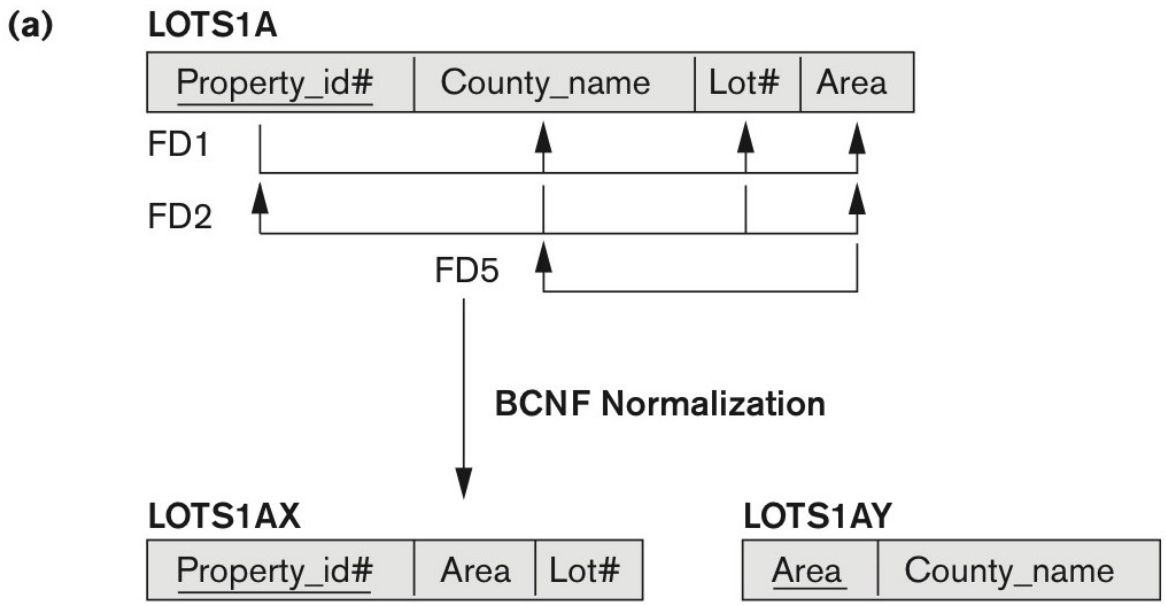
Which Normal form is this?



A relation schema R is in **third normal form (3NF)** if it is in 2NF *and* no non-prime attribute A in R is transitively dependent on ***any*** key of R

It is in 3NF, but not in BCNF because C → B

Boyce-Codd normal form



Interpreting the General Definition of Third Normal Form (2)

- **ALTERNATIVE DEFINITION of 3NF: We can restate the definition as:**

  A relation schema R is in **third normal form (3NF)** if, whenever a nontrivial FD X→A holds in R, either

  a) X is a superkey of R or

  b) A is a prime attribute of R

The condition (b) takes care of the dependencies that "slip through" (are allowable to) 3NF but are "caught by" BCNF which we discuss next.

BCNF (Boyce-Codd Normal Form)

- **Definition of 3NF:**
- A relation schema R is in **3NF** if, whenever a nontrivial FD X→A holds in R, either
  - a) X is a superkey of R or
  - b) A is a prime attribute of R

- A relation schema R is in **Boyce-Codd Normal Form (BCNF)** if whenever an FD X → A holds in R, then
  - a) X is a superkey of R
  - ~~b) There is no b~~
- Each normal form is strictly stronger than the previous one
  - Every 2NF relation is in 1NF
  - Every 3NF relation is in 2NF
  - Every BCNF relation is in 3NF

Boyce-Codd normal form

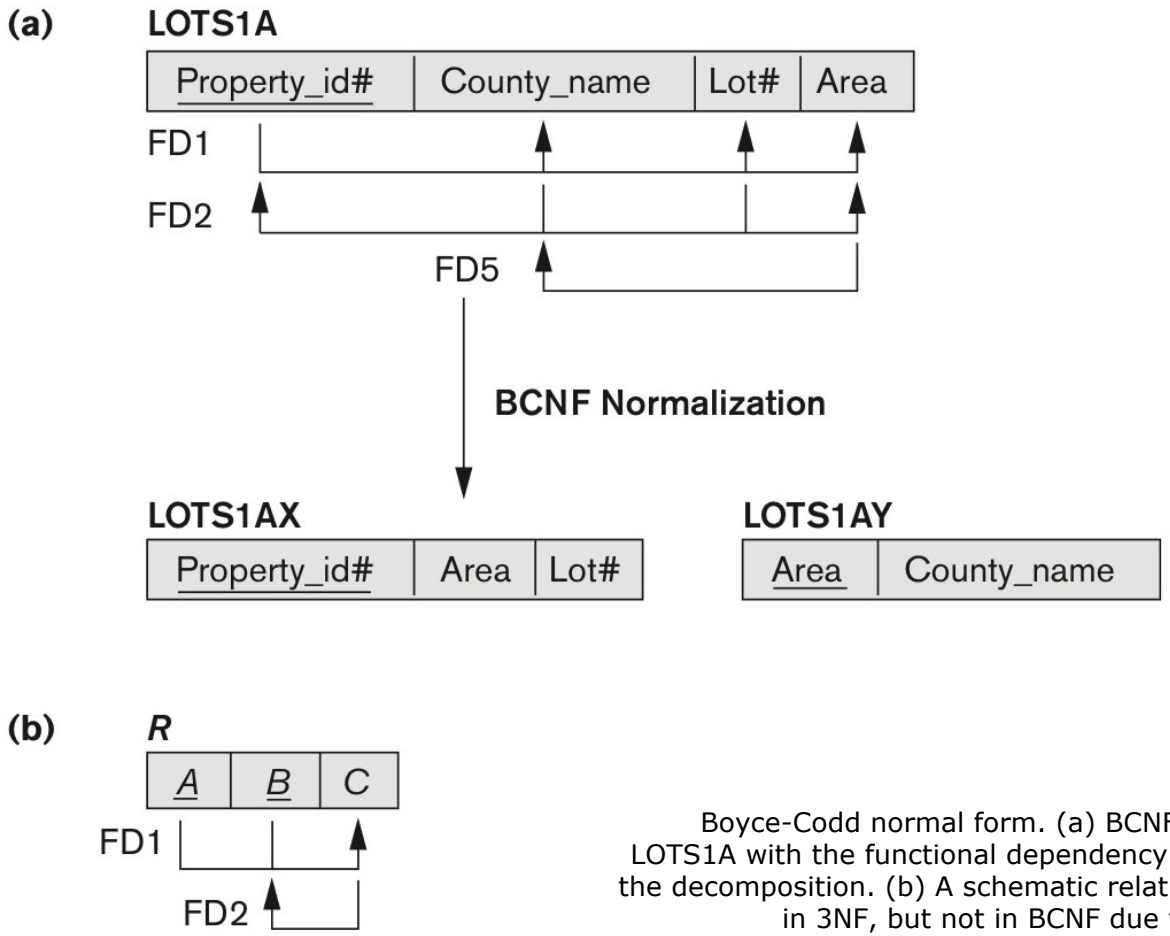


Figure 14.13
Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF due to the f.d. C → B.

A relation TEACH that is in 3NF but not in BCNF

TEACH

Student	Course	Instructor
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating Systems	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe
Narayan	Operating Systems	Ammar

- Two FDs exist in the relation TEACH:

  X → A

  – {student, course} → instructor
  – instructor → course

- {student, course} is a candidate key for this relation
- So this relation is in 3NF *but not in* BCNF
- A relation **NOT** in BCNF should be decomposed
  - while possibly forgoing the preservation of all functional dependencies in the decomposed relations.

Achieving the BCNF by Decomposition

- Three possible decompositions for relation TEACH
  - D1: {student, instructor} and {student, course}
  - D2: {course, instructor } and {course, student}
  - ✓ D3: {instructor, course } and {instructor, student}

Boyce-Codd Normal Form

BCNF is a simple condition for removing anomalies from relations:

A relation R is **in BCNF** if:

if $\{X_1, ..., X_n\} \rightarrow A$ is a *non-trivial* FD in R

then $\{X_1, ..., X_n\}$ **is a superkey** for R

In other words: there are no "bad" FDs

Example

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield
Joe	987-65-4321	908-555-1234	Westfield

`{SSN} → {Name,City}`

This FD is *bad* because it is **not** a superkey

⟹ **Not** in BCNF

What is the key?
{SSN, PhoneNumber}

Example

Name	SSN	City
Fred	123-45-6789	Seattle
Joe	987-65-4321	Madison

SSN	PhoneNumber
123-45-6789	206-555-1234
123-45-6789	206-555-6543
987-65-4321	908-555-2121
987-65-4321	908-555-1234

Now in BCNF!

{SSN} → {Name,City}

This FD is now *good* because it is the key

Let's check anomalies:

- Redundancy ?
- Update ?
- Delete ?

BCNF Decomposition Algorithm

BCNFDecomp(R):

BCNF Decomposition Algorithm

BCNFDecomp(R):

Find *a set of attributes* X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

Find a set of attributes X which has non-trivial "bad" FDs, i.e. is not a superkey, using closures

BCNF Decomposition Algorithm

BCNFDecomp(R):

Find a *set of attributes* X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

if (not found) **then Return** R

If no "bad" FDs found, in BCNF!

BCNF Decomposition Algorithm

BCNFDecomp(R):

Find a *set of attributes* X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

if (not found) **then Return** R

let $Y = X^+ - X$, $Z = (X^+)^C$

Let Y be the attributes that ***X functionally determines*** (+ that are not in X)

And let Z be **the other attributes that it *doesn't***

BCNF Decomposition Algorithm

BCNFDecomp(R):

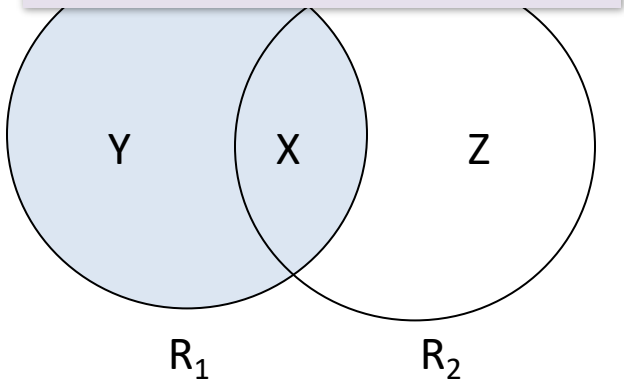
Find a *set of attributes* X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

if (not found) **then Return** R

let $Y = X^+ - X$, $Z = (X^+)^C$

decompose R into $\mathbf{R_1(X \cup Y)}$ and $\mathbf{R_2(X \cup Z)}$

Split into one relation (table) with X plus the attributes that X determines (Y)…



BCNF Decomposition Algorithm

BCNFDecomp(R):

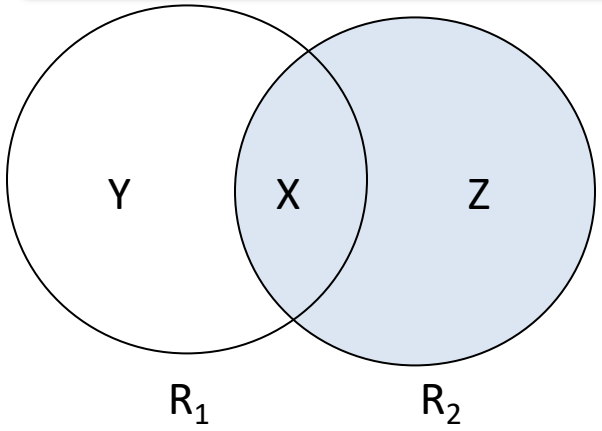
Find a *set of attributes* X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

if (not found) **then Return** R

let $Y = X^+ - X$, $Z = (X^+)^C$

decompose R into $\mathbf{R_1(X \cup Y)}$ and $\mathbf{R_2(X \cup Z)}$

And one relation with X plus the attributes it *does not* determine (Z)



BCNF Decomposition Algorithm

BCNFDecomp(R):

Find a *set of attributes* X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

if (not found) **then Return** R

let $Y = X^+ - X$, $Z = (X^+)^C$

decompose R into $\mathbf{R_1(X \cup Y)}$ and $\mathbf{R_2(X \cup Z)}$

Return BCNFDecomp(R_1), BCNFDecomp(R_2)

Proceed recursively until no more "bad" FDs!

Another way of representing the same concept

BCNFDecomp(R):
If X→ A causes BCNF violation:

Decompose R into

R1= XA
R2 = R –A

(Note: X is present in both R1 and R2)

Example

BCNFDecomp(R):

Find a *set of attributes* X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

if (not found) **then Return** R

let $Y = X^+ - X$, $Z = (X^+)^C$

decompose R into $\mathbf{R_1(X \cup Y)}$ and $\mathbf{R_2(X \cup Z)}$

Return BCNFDecomp(R_1), BCNFDecomp(R_2)

R(A,B,C,D,E)

{A} → {B,C}
{C} → {D}

BCNFDecomp(R):

If X→ A causes BCNF violation:

Decompose R into

R1= XA

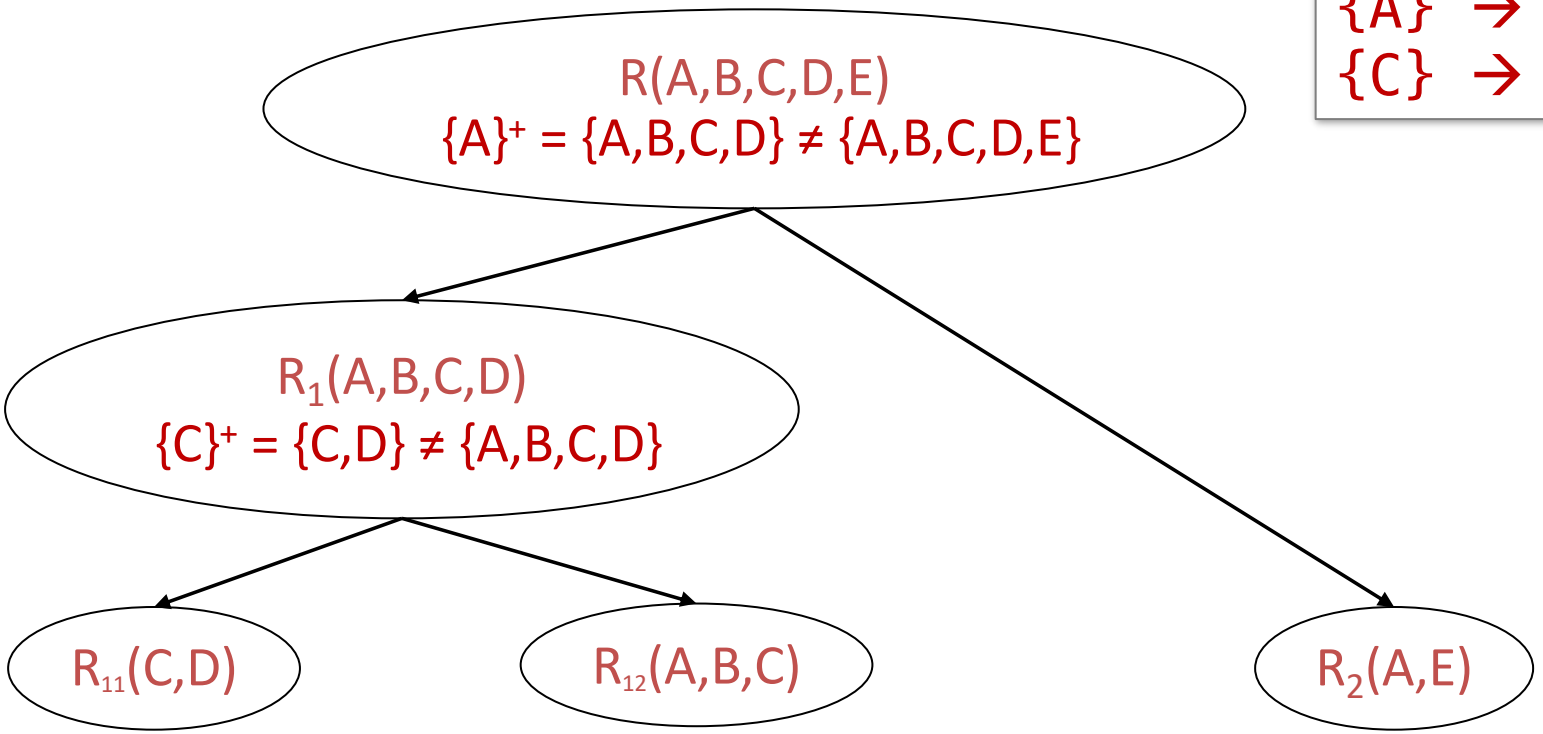
R2 = R –A

(Note: X is present in both R1 and R2)

Example

R(A,B,C,D,E)

{A} → {B,C}
{C} → {D}



Another Example

- Let the relation schema R(A,B,C,D) is given. For each of the following set of FDs do the following:

- i) indicate all the BCNF violations. Do not forget to consider FD's that are not in the given set. However, it is not required to give violations that have more than one attribute on the right side.

- ii) Decompose the relations, as necessary, into collections of relations that are in BCNF.

- 1. FDs AB→ C, C→D, and D→A

Find Non-trivial dependencies

There are 14 nontrivial dependencies, They are:
C→A, C→D, D→A, AB→D, AB→ C, AC→D,BC→A, BC→D, BD→A, BD→C, CD→A, ABC→D, ABD→C, and BCD→A.

Proceed From There

- One choice is to decompose using the violation C→D.
- Using the above FDs, we get ACD (Because of C→D and C→A) and BC as decomposed relations.
- BC is surely in BCNF, since any two-attribute relation is.
- we discover that ACD is not in BCNF since C is its only key.
- We must further decompose ACD into AD and CD.
- Thus, the three relations of the decomposition are BC, AD, and CD.

Two other topics to Study

- Cover and Minimal Cover

- Let F = {A → C, AC → D, E → AD, E → H}.
- Let G = {A → CD, E → AH}.

- Show that:
- 1. G covers F
- 2. F covers G
- 3. F and G are equivalent

Cover

- We say that a set of functional dependencies F covers another set of functional dependencies G,
  - if every functional dependency in G can be inferred from F.
  - More formally, F covers G if $G+ \subseteq F+$
  - F is a minimal cover of G if F is the smallest set of functional dependencies that cover G

1. Show: G covers F or ($F \subseteq G^+$)

   let's check each FD in F:

   - $A \rightarrow C$
     So, let's find A^+ in G.
     A^+ in $G = \{ACD\}$ which includes C. So, continue;
   - $AC \rightarrow D$
     Let's get AC^+ in G,
     AC^+ in $G = \{ACD\}$ which contains D. So, continue;
   - $E \rightarrow AD$
     Let's get E^+ in G,
     E^+ in $G = \{EACDH\}$ which contains AD. So, continue;

   We found that every dependency in F can be inferred from G.
   So, we can say that G covers F.

Acknowledgement

- Some of the slides in this presentation are taken from the slides provided by the authors.
- Many of these slides are taken from cs145 course offered by Stanford University.
- Thanks to YouTube, especially to Dr. Daniel Soper for his useful videos.