Decompositions

Theory of dependencies can tell us

- about ***redundancy*** and
- give us clues about ***possible decompositions***

But it *cannot discriminate* between decomposition alternatives.

A designer has to consider the alternatives and choose one based on the semantics of the application

Recap: Decompose to remove redundancies

1. We saw that **redundancies** in the data ("bad FDs") can lead to data anomalies

2. We developed mechanisms to **detect and remove redundancies by decomposing tables into BCNF**
   1. *BCNF decomposition is standard practice- very powerful & widely used!*

3. However, sometimes decompositions can lead to **more subtle unwanted effects…**

When does this happen?

Decompositions in General

$$\begin{tikzcd}[column sep=small] & R(A_1,\dots,A_n,B_1,\dots,B_m,C_1,\dots,C_p) \arrow[dl] \arrow[dr] & \\ R_1(A_1,\dots,A_n,B_1,\dots,B_m) & & R_2(A_1,\dots,A_n,C_1,\dots,C_p) \end{tikzcd}$$

R_1 = the *projection* of R on $A_1, \dots, A_n, B_1, \dots, B_m$

R_2 = the *projection* of R on $A_1, \dots, A_n, C_1, \dots, C_p$

Theory of Decomposition

FD1: { student, course} -> instructor
FD2: instructor -> course

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

Course	Instructor
Database	Mark
Database	Navathe
Database	Omiecinski
Operating System	Ammar
Operating System	Ahamad
Theory	Schulman

Student	Instructor
Narayan	Mark
Smith	Navathe
Smith	Ammar
Smith	Schulman
Wallace	Mark
Wallace	Ahamad
Wong	Omiecinski
Zelaya	Navathe
Narayan	Ammar

Sometimes a decomposition is "correct"

I.e. it is a **Lossless decomposition**

Theory of Decomposition

FD1: { student, course} -> instructor
FD2: instructor -> course

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

Course	Instructor
Database	Mark
Database	Navathe
Database	Omiecinski
Operating System	Ammar
Operating System	Ahamad
Theory	Schulman

Student	Course
Narayan	Database
Smith	Database
Smith	Operating Systems
Smith	Theory
Wallace	Database
Wallace	Operating Systems
Wong	Database
Zelaya	Database
Narayan	Operating Systems

However sometimes it isn't

What's wrong here?

Lossy Decomposition

Theory of Decomposition

FD1: { student, course} -> instructor
FD2: instructor -> course

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

Student	Instructor
Narayan	Mark
Smith	Navathe
Smith	Ammar
Smith	Schulman
Wallace	Mark
Wallace	Ahamad
Wong	Omiecinski
Zelaya	Navathe
Narayan	Ammar

Student	Course
Narayan	Database
Smith	Database
Smith	Operating Systems
Smith	Theory
Wallace	Database
Wallace	Operating Systems
Wong	Database
Zelaya	Database
Narayan	Operating Systems

However sometimes it isn't

What's wrong here?

Lossy Decomposition

Lossless Decompositions

$R(A_1,\ldots,A_n,B_1,\ldots,B_m,C_1,\ldots,C_p)$

$R_1(A_1,\ldots,A_n,B_1,\ldots,B_m)$ $\quad$ $R_2(A_1,\ldots,A_n,C_1,\ldots,C_p)$

What (set) relationship holds between R1 Join R2 and R if lossless?

Hint: Which tuples of R will be present?

It's lossless if we have equality!

Lossless Decompositions

$R(A_1,\ldots,A_n,B_1,\ldots,B_m,C_1,\ldots,C_p)$

$R_1(A_1,\ldots,A_n,B_1,\ldots,B_m)$

$R_2(A_1,\ldots,A_n,C_1,\ldots,C_p)$

A decomposition R to (R1, R2) is **lossless** if R = R1 Join R2

Lossless Decompositions

$R(A_1,\ldots,A_n,B_1,\ldots,B_m,C_1,\ldots,C_p)$

$R_1(A_1,\ldots,A_n,B_1,\ldots,B_m)$

$R_2(A_1,\ldots,A_n,C_1,\ldots,C_p)$

If $\{A_1, \ldots, A_n\} \rightarrow \{B_1, \ldots, B_m\}$
Then the decomposition is lossless

Note: don't need
$\{A_1, \ldots, A_n\} \rightarrow \{C_1, \ldots, C_p\}$

BCNF decomposition is always lossless. Why?

Testing Binary Decompositions for Lossless Join Property

- **Binary Decomposition:** decomposition of a relation *R* into two relations.
- **Lossless join test for binary decompositions:**
  - *A decomposition D = {R_1, R_2} of R has the lossless join property with respect to a set of functional dependencies F on R if and only if either*
    - **FD (($R_1 \cap R_2$) → (R_1- R_2)) is in F^+, or**
    - **FD (($R_1 \cap R_2$) → (R_2 - R_1)) is in F^+.**

In other words, the decomposition is lossless if the set of attributes used to join R_1 and R_2 i.e. **($R_1 \cap R_2$)** should be key either in R_1 or R_2

A problem with BCNF

Problem: To enforce a FD, must reconstruct original relation—*on each insert!*

Note: This is historically inaccurate, but it makes it easier to explain

Theory of Decomposition

TEACH

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

Lossless decomposition

Course	Instructor
Database	Mark
Database	Navathe
Database	Omiecinski
Operating System	Ammar
Operating System	Ahamad
Theory	Schulman

We lose the FD **{ student, course} -> instructor**

Narayan	Mark
Smith	Navathe
Smith	Ammar
Smith	Schulman
Wallace	Mark
Wallace	Ahamad
Wong	Omiecinski
Zelaya	Navathe
Narayan	Ammar

FD1: { student, course} -> instructor
FD2: instructor -> course

We do a BCNF decomposition on a "bad"

FD: Instructor-> course

So Why is that a Problem?

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

Lossless decomposition

Course	Instructor
Database	Mark
Database	Navathe
Database	Omiecinski
Operating System	Ammar
Operating System	Ahamad
Theory	Schulman

Insert row Database XYZ

Student	Instructor
Narayan	Mark
Smith	Navathe
Smith	Ammar
Smith	Schulman
Wallace	Mark
Wallace	Ahamad
Wong	Omiecinski
Zelaya	Navathe
Narayan	Ammar

No violation in FD Instructor ->course

Insert row Smith XYZ

Join the two decomposed relation to get relation Teach this

Violates the FD1 !

FD1: { student, course} -> instructor

The Problem

- We started with a table R and FDs F

- We decomposed R into BCNF tables R_1, R_2, …
  with their own FDs F_1, F_2, …

- We insert some tuples into each of the relations—which satisfy their local FDs but when reconstruct it violates some FD **across** tables!

Practical Problem: To enforce FD, must reconstruct R—*on each insert!*

Possible Solutions

- Various ways to handle so that decompositions are all lossless / no FDs lost
  - *For example 3NF- stop short of full BCNF decompositions.*

- Usually a tradeoff between redundancy / data anomalies and FD preservation…

BCNF still most common- with additional steps to keep track of lost FDs…

3NF

3NF

- 3NF can give a loss-less and dependency preserving decomposition
  - *But at a cost of redundancy*

- A tradeoff between
  - ***Dependency Preservation***
  - ***Redundancy & Anomalies***

3NF –Third Normal Form

A relation R is in **3NF** if whenever the FD X -> A holds in R, then either:

- X is a superkey of R, or
- A is a prime attribute of R

- **Prime attribute:** it must be a member of *some* candidate key
- **Nonprime attribute:** it is not a member of any candidate key.

3NF

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

FD1: { student, course} -> instructor
FD2: instructor -> course

Lets do 3NF decomposition:
if X -> A holds in R, then either:
a) X is a superkey of R, or
b) A is a prime attribute of R

Key is {**student, course**}
Relation R is already in 3NF so no decomposition required

3NF Benefit:
We do not lose the FD
FD1: { student, course} -> instructor

A Problem with 3NF

We do not lose the

FD1: { student, course} -> instructor

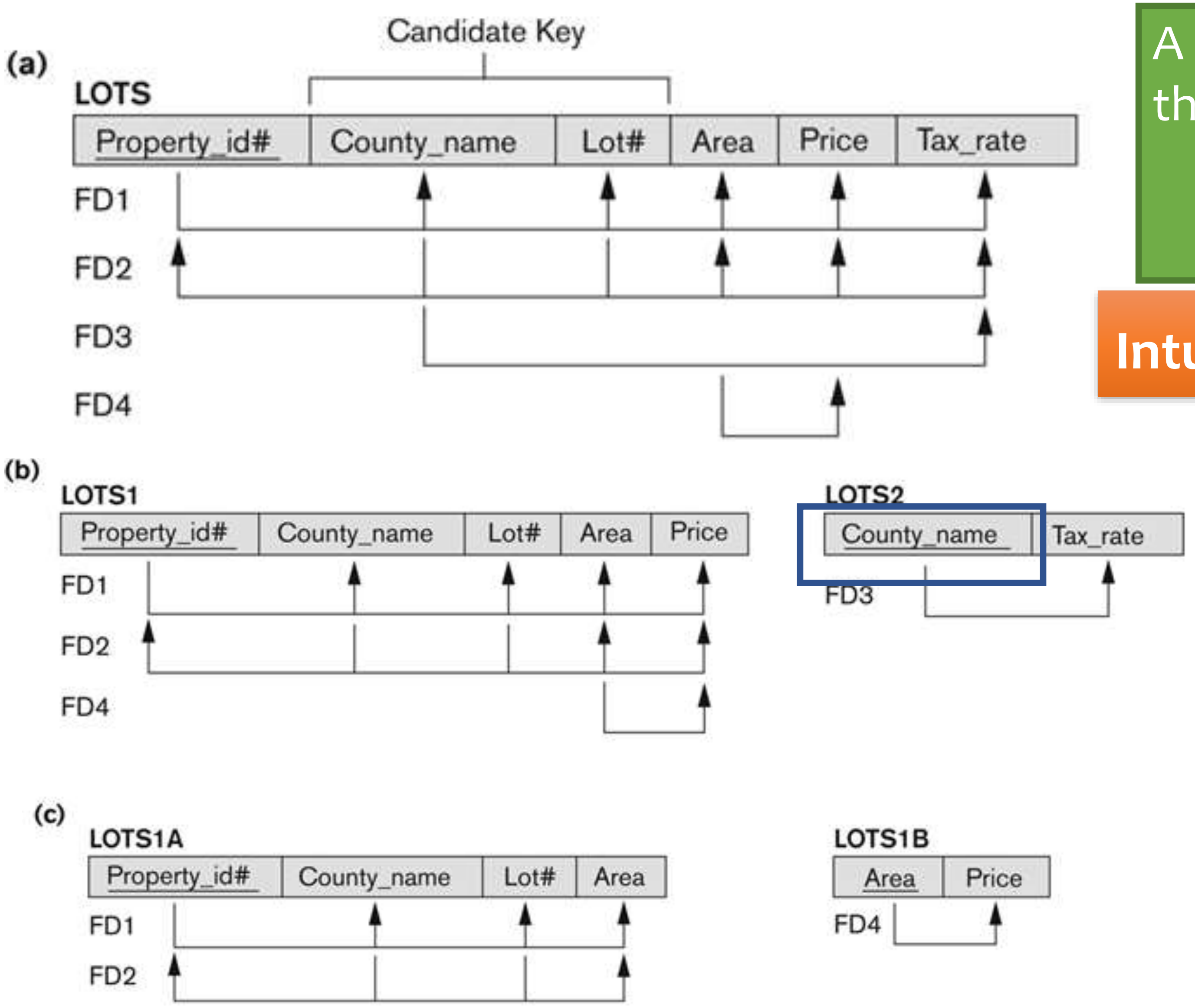
TEACH

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

Let's check anomalies:

- Redundancy ?
- Update ?
- Delete ?

Example 2: Relation LOTS



A relation R is in **3NF** if X -> A holds in R, then either:

- X is a superkey of R, **OR**
- A is a prime attribute of R

Intuitively what does this means ???

X is a superkey of R,

No Partial Dependency
A dependency where part of the key determine **non prime attribute**.
This is 2NF condition

A is a prime attribute of R

No **Transitive dependency** with **non-prime attribute**

2NF& 3NF

- A relation with no partial dependency is said to be in 2NF
- **Partial Dependency**
  - *A dependency where part of the key determine non prime attribute.*
- A relation with no partial dependency and transitive dependency is said to be in 3NF

How to detect if relation is in 3NF ?

Use the following rule

A relation R is in **3NF** if X -> A holds in R, then either:

a) X is a superkey of R, or
b) A is a prime attribute of R

Activity: 3NF

- What is the key for R ? What is the current Normal Form of R ?
- Given F = {A, B->C,

  B, D->E, F,

  A, D->G, H,

  A->I,

  H->J }
- Current NF is 1NF
- 3NF decomposition

3NF decomposition:
if X -> A holds in R, then either:
  a) X is a superkey of R, or
  b) A is a prime attribute of R

Key is ABD

Activity: 3NF

- What is the key for R ? What is the current Normal Form of R ?
- Given F = {A, B->C,

  B, D->E, F,

  A, D->G, H,

  A->I,

  H->J }

3NF decomposition:
if X -> A holds in R, then either:
  a) X is a superkey of R, or
  b) A is a prime attribute of R

Key is ABD

- Current NF is 1NF
- 3NF decomposition
  - *Removing partial dependencies*
    - R1 =(A, B, C) R2= (B,D, E, F) R3= (A, D, G, H, J) R4= (A, I) R=(A,B, D)
  - *Removing transitive dependencies*
    - R1 = (A, B, C) R2 = (B,D, E, F) R3.1 = (A, D, G, H) R3.2 =(H, J) R4= (A, I), R=(A,B,D)