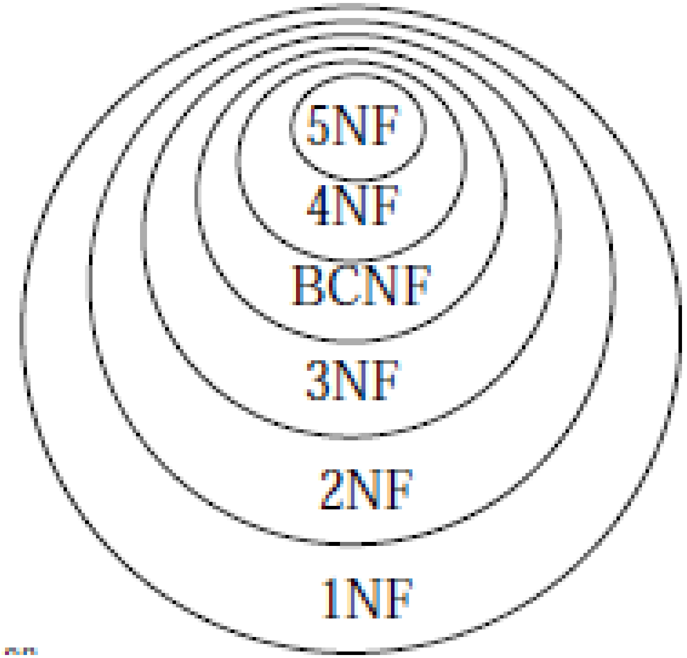
NORMALIZATION

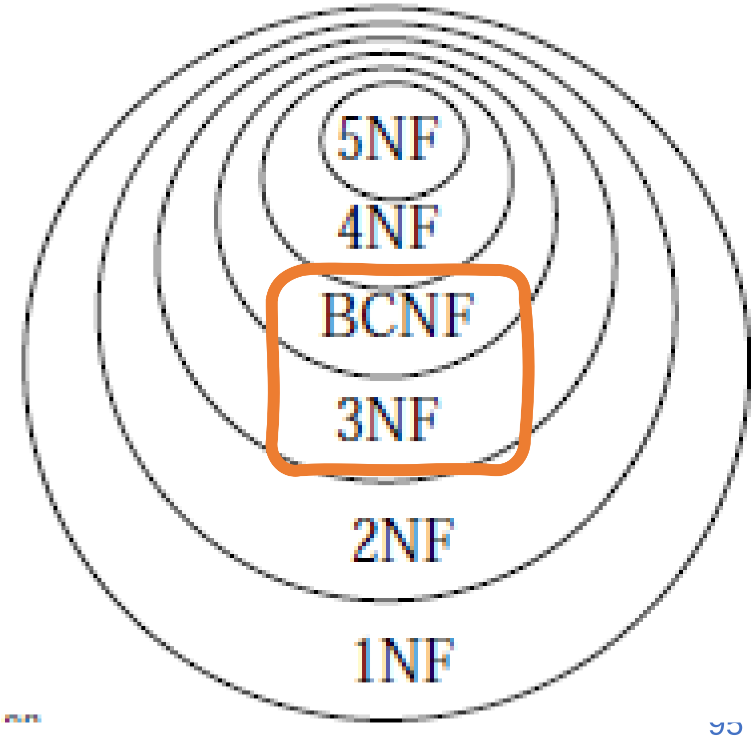
THE COMPLETE BOOK by Ullman

Chapter 3: Design Theory for Relational Databases



Normalization

- The process of decomposing "bad" relation by breaking it into smaller relations
- 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*



Normal Forms

- 1st Normal Form (1NF) = trivial (All tables are flat)

- *2nd Normal Form = obsolete (remove partial Dependencies)*

- **3rd Normal Form (3NF)**

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

DB designs based on *functional dependencies*, intended to prevent data ***anomalies***

Major focus is on these dependencies

- *4th Normal Forms = remove Multi-values dependencies*

1st Normal Form (1NF)

Student	Courses
Mary	{CS145,CS229}
Joe	{CS145,CS106}
…	…

Violates 1NF.

<u>Student</u>	<u>Courses</u>
Mary	CS145
Mary	CS229
Joe	CS145
Joe	CS106

In 1st NF

1NF Constraint: Types must be atomic!

1^{st} Normal Form (1NF)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

Boyce-Codd Normal Form

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*

Recall: there are several normal forms…

Boyce-Codd Normal Form (BCNF)

- Main idea is that we define "good" and "bad" FDs as follows:
  - *$X \rightarrow A$ is a "good FD" if X is a (super)key*
  - *$X \rightarrow A$ is a "bad FD" otherwise*
- We will try to eliminate the "bad" FDs!

Boyce-Codd Normal Form (BCNF)

- Why does this definition of "good" and "bad" FDs make sense?
- If X is *not* a (super)key, it functionally determines *some* of the attributes; therefore, those other attributes can be duplicated
  - *Recall: this means there is redundancy*
  - *And redundancy like this can lead to data anomalies!*

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

Boyce-Codd Normal Form

BCNF is a simple condition for removing anomalies from relations:

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

if $\{\mathbf{A_1}, \ldots, \mathbf{A_n}\} \rightarrow \mathbf{B}$ is a *non-trivial* FD in R

then $\{\mathbf{A_1}, \ldots, \mathbf{A_n}\}$ **is a superkey** for R

Equivalently: $\forall$ sets of attributes X, either ($X^+ = X$) or (X^+ = all attributes)

In other words: there are no "bad" FDs

Example

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

$\{EmpID\} \rightarrow \{Name, Phone, Position\}$

This FD is *good* because EmpID is a superkey

$\{Position\} \rightarrow \{Phone\}$

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

$\Longrightarrow$ **Not** in BCNF

What is the key?
{EmpID}

Example 2

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 2

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

{SSN} → {Name,City}

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

Let's check anomalies:

- Redundancy ?
- Update ?
- Delete ?

Now in BCNF!

BCNF Decomposition Algorithm

Input: A relation R_0 and a set of FDs F.

1. Set D := $\{R_0\}$;
2. While there is a relation schema R in *D* that is not in BCNF
   do {
   - Choose a ***R*** in ***D*** that is not in BCNF
   - Find a FD $\boldsymbol{X \rightarrow Y}$ in R that violates BCNF
   - Decompose R in *D* by two relations R_1= $\boldsymbol{X^+}$ **and** $\boldsymbol{R_2}$**= (X U Z),** *where Z is the set of attributes not in* X^+

   }

Find a non-trivial "bad" FD X->Y, i.e. X is not a superkey

BCNF Decomposition Algorithm

Input: A relation R_0 and a set of FDs F.

1. Set D := $\{R_0\}$;
2. While there is a relation schema R in D that is not in BCNF
   do {
   Choose a $\mathbf{R}$ in $\mathbf{D}$ that is not in BCNF
   Find a FD $\mathbf{X} \rightarrow \mathbf{Y}$ in R that violates BCNF
   Decompose R in D by two relations $R_1 = \mathbf{X}^+$ **and** $\mathbf{R_2} = (\mathbf{X \cup Z})$, *where Z is the set of attributes not in* X^+
   }

X$^+$ is set of the attributes that ***X functionally determines***

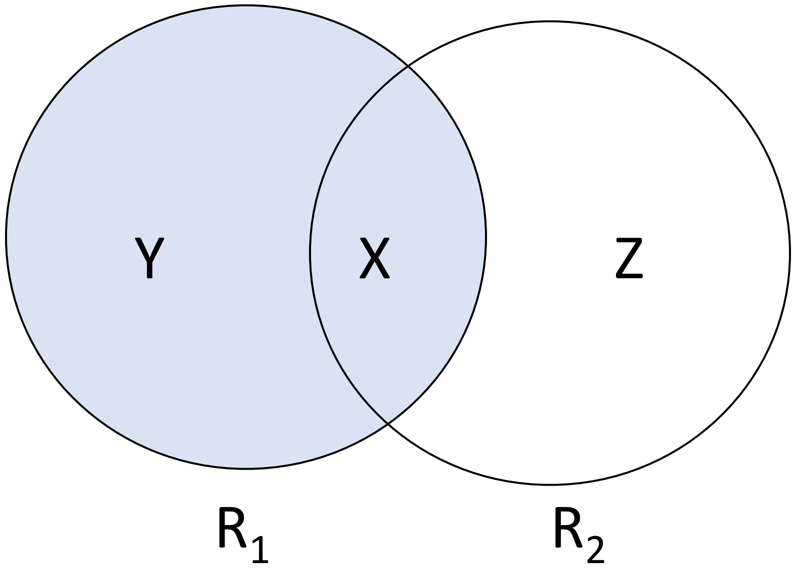
And Z is **set of attributes that it *doesn't***

BCNF Decomposition Algorithm

Input: A relation R_o and a set of FDs F.

1. Set D := {R_o};
2. While there is a relation schema R in *D* that is not in BCNF
   do {
      Choose a ***R*** in ***D*** that is not in BCNF
      Find a FD ***X*** → ***Y*** in *R* that violates BCNF
      Decompose *R* in *D* by two relations R_1= ***X***$^+$ **and R_2= (X U Z),** *where Z is the set of attributes not in X*$^+$
   }

Split into one relation with X plus the attributes that X determines (i.e. Y)…



BCNF Decomposition Algorithm

Input: A relation R_o and a set of FDs F.

1. Set D := $\{R_o\}$;
2. While there is a relation schema R in *D* that is not in BCNF

   do {

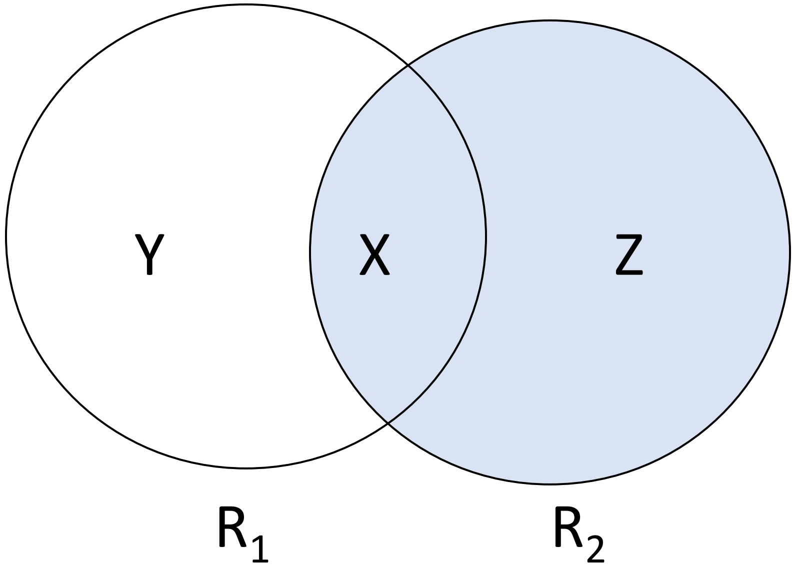
   Choose a ***R*** in ***D*** that is not in BCNF

   Find a FD ***X*** → ***Y*** in R that violates BCNF

   Decompose R in *D* by two relations R_1= ***X⁺*** **and** $\mathbf{R_2}$**= (X U Z),** *where Z is the set of attributes not in X⁺*

   }

And one relation with X plus the attributes X *does not* determine (i.e Z)



BCNF Decomposition Algorithm

Input: A relation R_0 and a set of FDs F.

1. Set D := $\{R_0\}$;
2. While there is a relation schema R in *D* that is not in BCNF do {

   Choose a ***R*** in ***D*** that is not in BCNF

   Find a FD ***X*** → ***Y*** in *R* that violates BCNF

   Decompose *R* in *D* by two relations $R_1 = \boldsymbol{X^+}$ **and** $\mathbf{R_2 = (X \cup Z)}$, *where Z is the set of attributes not in* X^+

   }

Proceed until no more "bad" FDs!

Example

Input: A relation R_0 and a set of FDs F.

1. Set D := $\{R_0\}$;
2. While there is a relation schema *R* in *D* that is not in BCNF
   do {
   Choose a ***R*** in ***D*** that is not in BCNF
   Find a FD ***X*** → ***Y*** in *R* that violates BCNF
   Decompose *R* in *D* by two relations $R_1 = X^+$ **and** $R_2 = (X \cup Z)$, *where Z is the set of attributes not in* X^+
   }

Example

R(A,B,C,D,E)
{A} → {B,C}
{C} → {D}

First find Key??

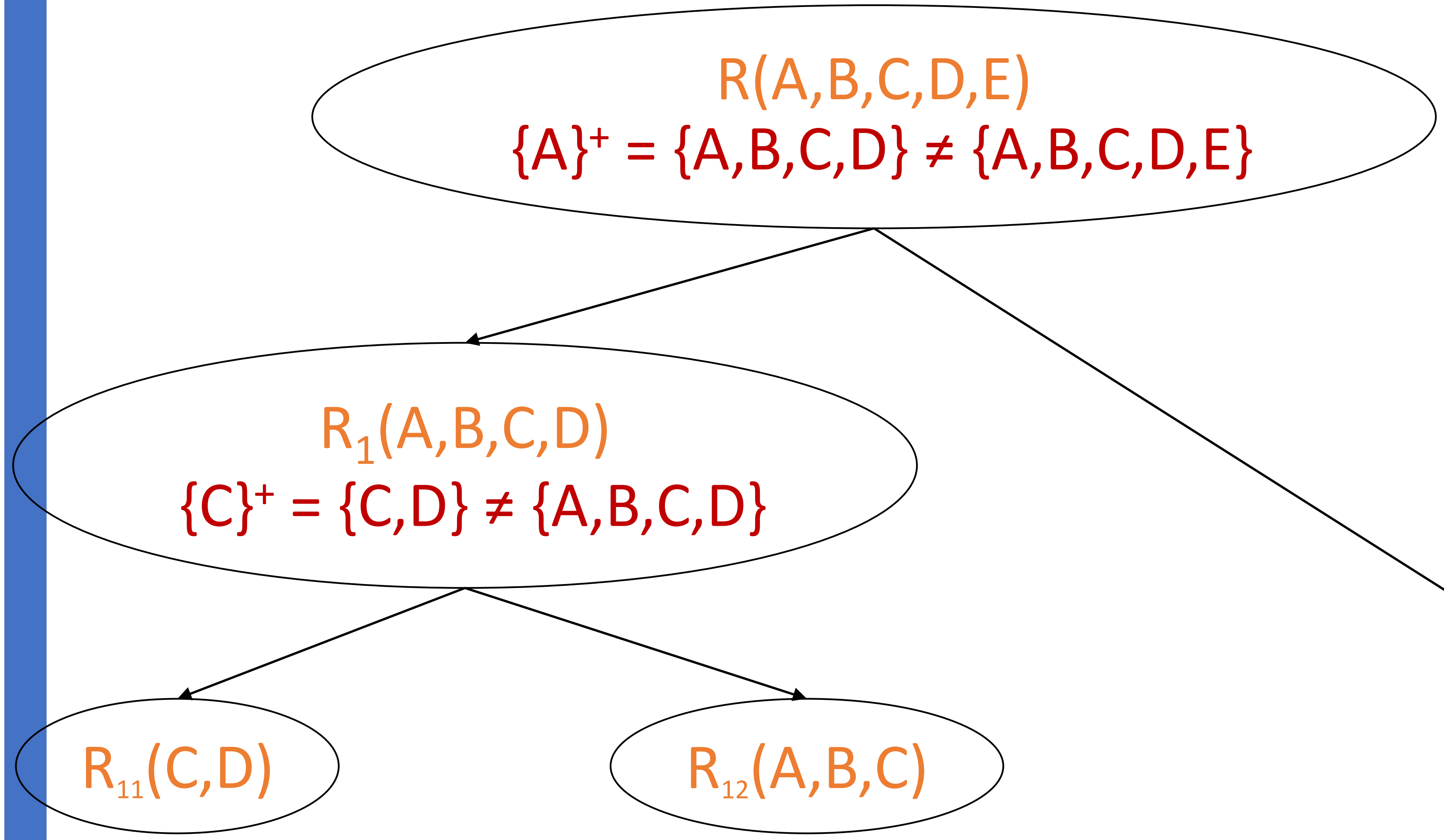
Example

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

Key AE

$\{A\} \rightarrow \{B,C\}$

$\{C\} \rightarrow \{D\}$



ACTIVITY: BCNF

Consider the relation **_Contracts(Cid, Sid, Pid, dept, part, qty)_**

- *In this relation, the contract with Cid is an agreement that supplier Sid (supplierid) will supply Q items of Part to project Pid (projectid) associated with department Dept*
- FD's
  - *Cid is a key*
    - *Cid -> Cid, Sid, Pid, Dept, Part, Qty.*
  - *A project purchases a given part using a single contract*
    - Pid, Part -> *Cid.*
  - *A department purchases at most one part from a supplier*
    - *Sid, Dept -> Part.*
  - *Each project deals with a single supplier*
    - Pid -> Sid

Is the relation Contract in BCNF?

Key ?

1. Cid
2. Pid, Part
3. Pid, Dept

ACTIVITY: BCNF Decomposition

Consider the relation ***Contracts(Cid, Sid, Pid, dept, part, qty)***

- FD's
  - *Cid -> Cid, Sid, Pid, Dept, Part, Qty.*
  - *Pid, Part -> Cid.*
  - ***Sid, Dept -> Part.***
  - ***Pid -> Sid***

Lets take Sid, Dept -> Part

- **R2(Sid, Dept, Part)**
  - Now its in BCNF
- *Contracts(Cid, Sid, Pid, Dept, qty)*
  - *Still not in BCNF because of* Pid -> Sid
  - **R3(Pid , Sid)**
  - ***Contracts(Cid, Pid, dept, qty)***

Another decomposition

Lets start with Pid -> Sid

- R2(Pid, Sid)
  - Now its in BCNF
- *Contracts(Cid, Pid, Dept, Part, qty)*
  - *In BCNF*
  - *But lost dependency* ***Sid, Dept -> Part***

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

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

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

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

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

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

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

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

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

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 $((R_1 \cap R_2) \rightarrow (R_1 - R_2))$ is in F^+, or**
    - **FD $((R_1 \cap R_2) \rightarrow (R_2 - R_1))$ 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. $\mathbf{(R_1 \cap R_2)}$ 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

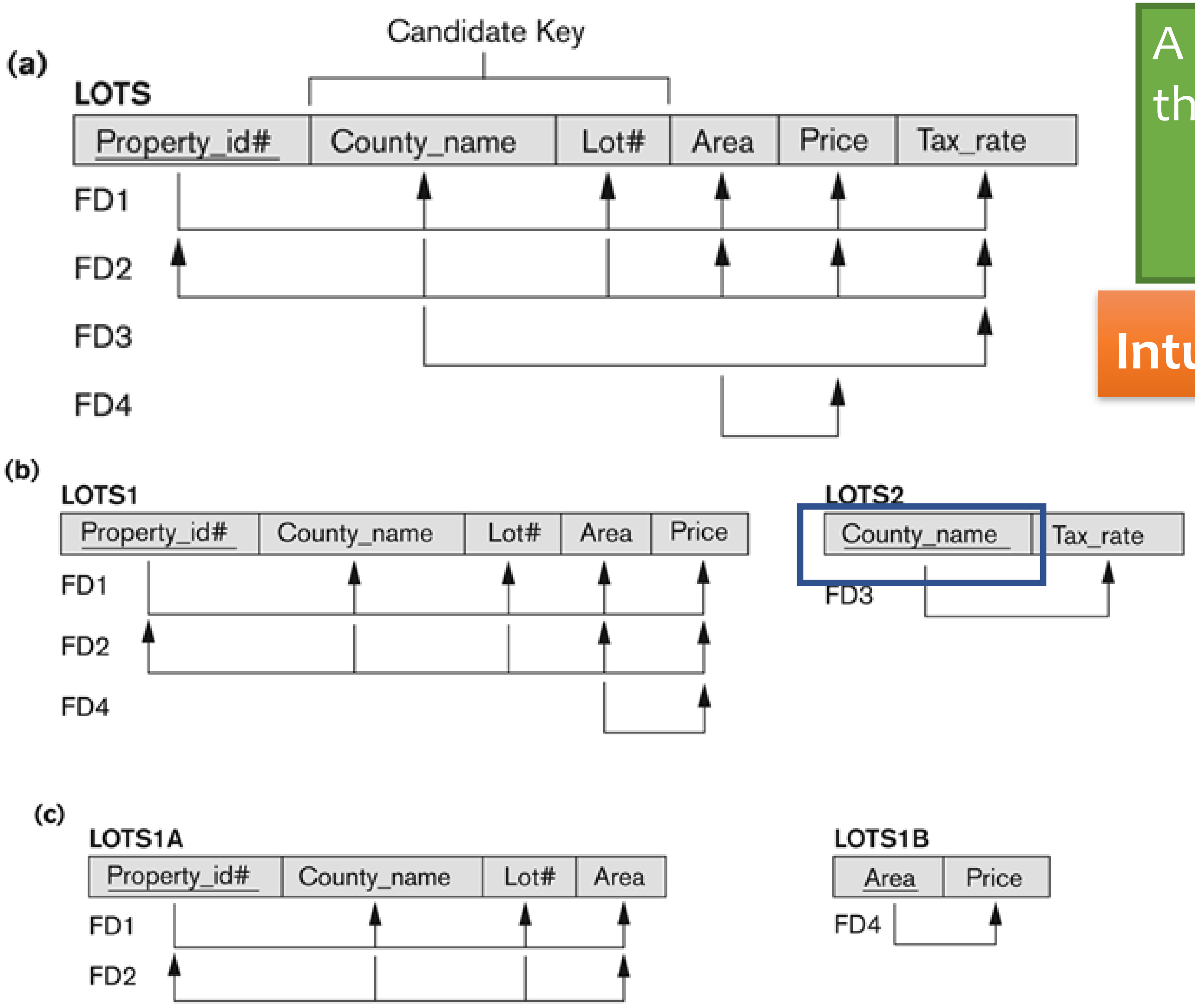
We do not lose the
FD1: { student, course} -> instructor

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: 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

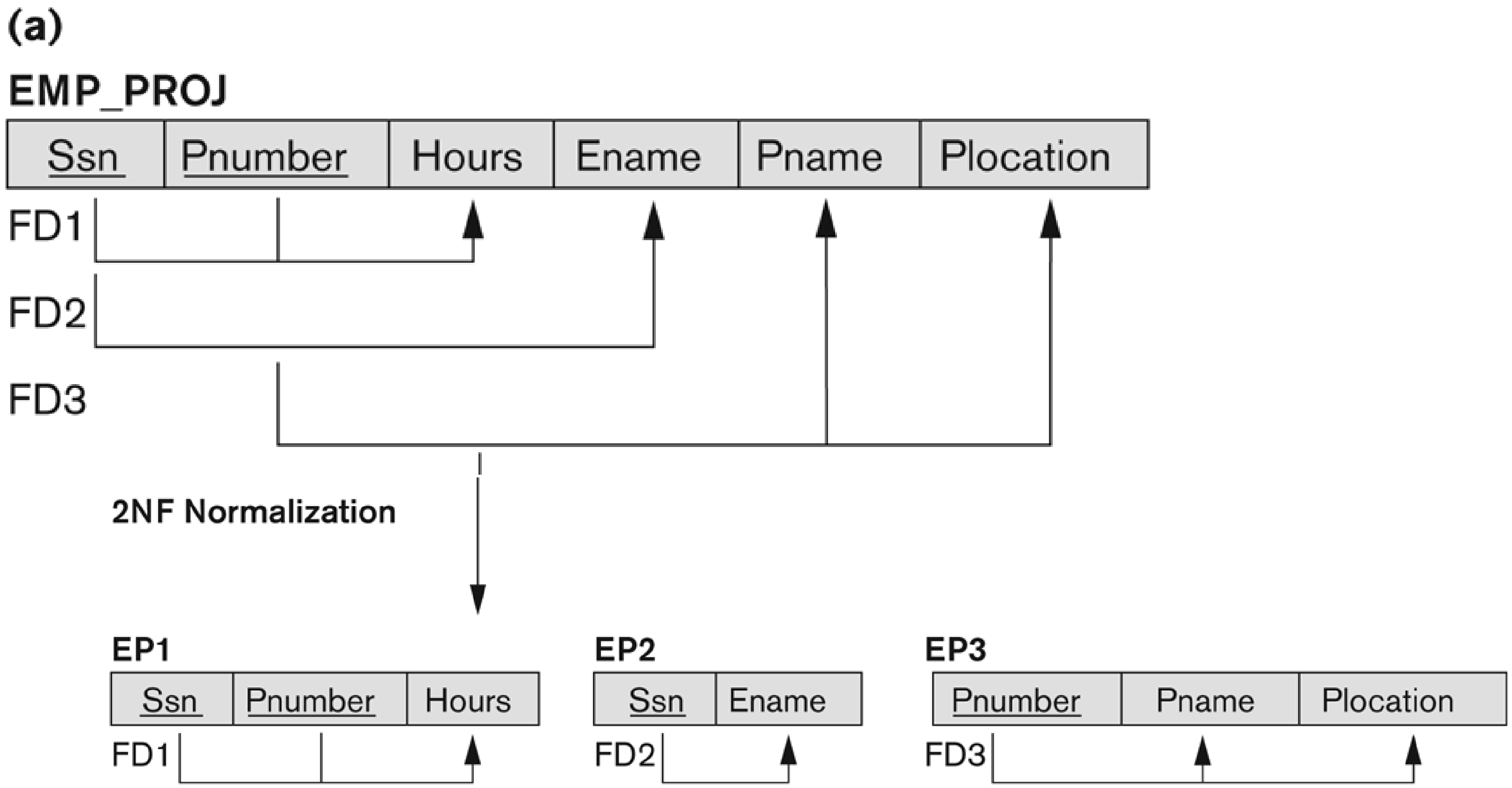
A relation R is in **3NF** if X -> A holds in R, then either:

a) X is a superkey of R, or

b) A is a prime attribute of R

Normalizing into 2NF

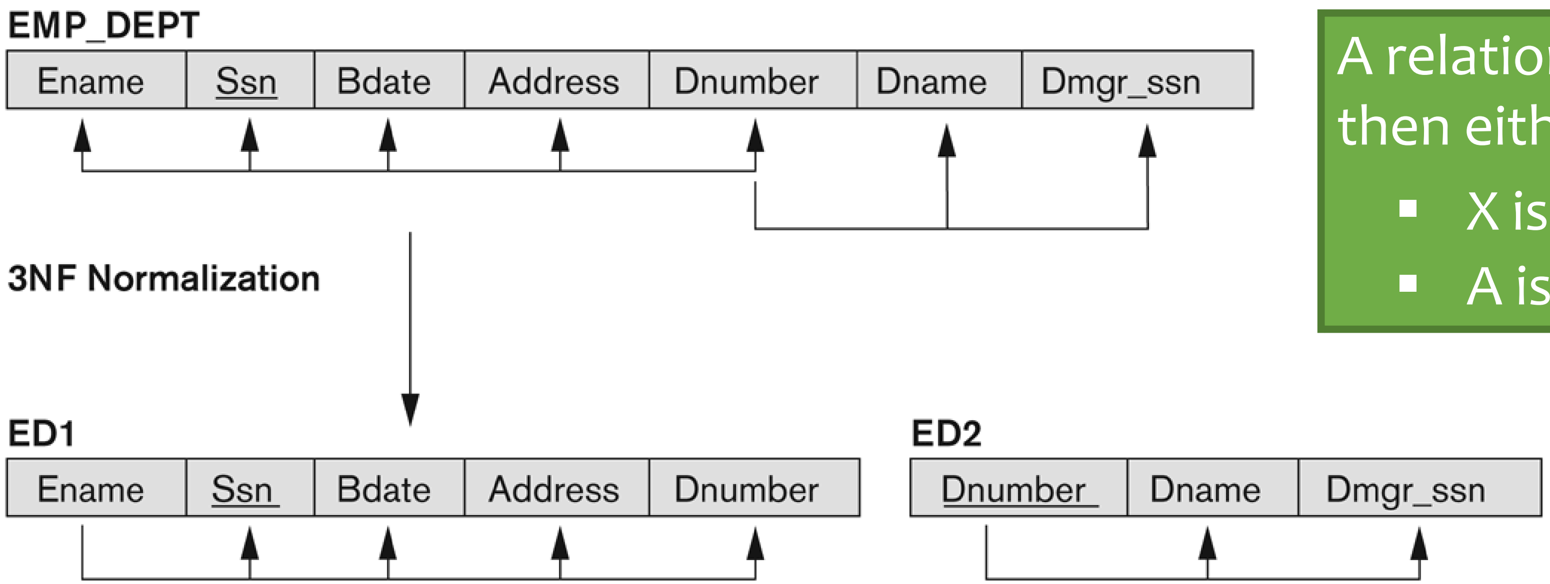
A relation with no partial dependency is said to be in 2NF



Third Normal Form

Third Normal Form:

A relation that is in 2NF, and in which **no non-prime attribute** is **transitively dependent** on the **primary key.**



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

SSN -> DMGRSSN is a transitive FD

Since SSN -> DNUMBER and DNUMBER -> DMGRSSN hold

Normal Forms Defined Informally

- 1st normal form
  - *All attributes depend on **the key***
- 2nd normal form
  - *All attributes depend on **the whole key (no partial dependencies)***
- 3rd normal form
  - *All attributes depend on **nothing but the key***

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

Key is ABD

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

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
  - *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)

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

3NF and Dependency Preservation

Consider the Relation Contracts(Cid, Sid, Pid, dept, part, qty)

- FD's
  - *Cid -> Cid, Sid, Pid, Dept, Part, Qty.*
  - *Pid, Part -> Cid.*
  - ***Sid, Dept -> Part.***
  - ***Pid -> Sid***
- 3NF
  - ***Pid -> Sid***
  - *R2(Pid, Sid)*
    - Now its in 3NF
  - *Contracts(Cid, Pid, Dept, Part, qty)*
    - In 3NF
    - But lost dependency **Sid, Dept -> Part**

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

Keys are
Cid
Pid, Part
Pid, Dept

Dependency Preserving 3NF

- Consider the Relation R(A,B,C,D) with the following FDs
  - *A-> B C D*
  - *B-> D*
  - *C-> D*
  - *3NF Decomposition*
  - *R1(B, D)*
  - *R2(A B C)*
  - *Dependencies are lost ? C->D*

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 **A**

Algorithm: Decomposition into 3NF

(with Dependency Preservation &Lossless Join)

Input: A universal relation *R* and a set of functional dependencies *F* on the attributes of *R*.

1. *Find a **minimal cover** G for F*

2. *For each X of a FD in G, create a relation in D with attributes*

   $\{X \cup \{A_1\} \cup \{A_2\} \ldots \cup \{A_k\}\}$,

3. ***If none of the relation in D contains a key of R, then create one more relation in D that contains key of R.***

4. *Eliminate Redundant relations from D*

Dependency Preserving 3NF

- Consider the Relation R(A,B,C,D) with the following FDs
  - *A-> B C D*
  - *B-> D*
  - *C-> D*

- 3NF Decomposition using synthesis Algorithm

- Minimal Cover
  - *A-> B C*
  - *B-> D*
  - *C-> D*
  - *A->D is a transitive dependencies so no need to preserve*
- Make a relation for each dependency

Example

- Consider R(ssn, pno, sal, phone, dno, pname, ploc)
- Consider following FDs
  - *FD1: ssn ->sal, phone, dno*
  - *FD2: pno-> pname, ploc*
  - *FD3: {ssn, pno }->sal, ephone, dno, pname, ploc*
- Key: {ssn, pno}
- Step1: minimal cover
  - *G: {ssn->sal,phone,dno ; pno->pname,ploc}*
- Step2:
  - *R1(ssn,sal,phone,dno)*
  - *R2(pno,pname,ploc)*
- **Step3:**
  - ***R3(ssn,pno)***

Algorithm: Decomposition into 3NF

(with Dependency Preservation &Lossless Join)

Input: A universal relation *R* and a set of functional dependencies *F* on the attributes of *R*.

1. *Find a minimal cover G for F*
2. *For each X of a FD in G, create a relation in D with attributes* $\{X \cup \{A_1\} \cup \{A_2\} \ldots \cup \{A_k\}\}$,
3. ***If none of the relation in D contains a key of R, then create one more relation in D that contains key of R.***
4. ***Eliminate Redundant relat***

a key
Note we do not have to preserve all keys

3NF and Dependency Preservation

Consider the Relation Contracts(Cid, Sid, Pid, dept, part, qty)

- FD's
  - *Cid -> Cid, Sid, Pid, Dept, Part, Qty.*
  - *Pid, Part -> Cid.*
  - ***Sid, Dept -> Part.***
  - ***Pid -> Sid***

3NF decomposition Rule

if X -> A holds in R, then either:

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

Keys are

Cid

Pid, Part

Pid, Dept

Create a relation for each FD

- R1 (Cid, Pid, Dept, Qty)
- R2 (Pid, Part, Cid)
- R3(Sid, Dept, Part)
- R4(Pid , Sid)

REAL WORLD EXAMPLES

Example: Designing a Student Database

1. Students take courses
2. Students typically take more than one course
3. Students can fail courses and can repeat the same course in different semesters => Students can take the same course more than once.
4. Students are assigned a grade for each course they take.

studentID(sID),
sname,
dept,
advisor,
course(ID),
credit,
semester,
grade,
course-room,
instr,
instr-office

Example: Designing a Student Database

- **Student_course(sID, sname, dept, advisor, course, credit, semester, grade, course-room, instr, instr-office)**
- 1NF: Flat table
- Problems:
  - *Redundancy*
  - *Insert anomalies*
  - *Delete anomalies*
  - *Update problems*

Example: Designing a Student Database

- *Define FDs*
  - **sname, dept, advisor** *are dependent only upon* ***sID***
    - sID -> sname, dept, advisor
  - ***credit*** *dependent only on* ***course*** *and is independent of which* ***semester*** *it is offered and which* ***student*** *is taking it.*
    - Course -> credit
  - **course-room, instructor and instructor-office** *only depend upon the* **course and the semester** *(are independent of which student is taking the course).*
    - Course,Semester -> course-room, instructor, instructor-office
  - *Only* ***grade*** *is dependent upon all 3 parts of the original key.*
    - Stud-Id, Course, Semester -> grade
  - ***Instructor-> instructor-office***

Example: Designing a Student Database

Student_course(<u>sID</u>, sname, dept, advisor, <u>course</u>, credit, <u>semester</u>, grade, course-room, instr, instr-office)

- Student (<u>sID</u>, sname, dept, advisor)
- Student_Reg (<u>sID, course, semester</u>, grade)
- Course (<u>course</u>, credit)
- Course_Offering (<u>course, semester</u>, course-room, instructor)
- Instructor (<u>instructor</u>, instructor-office)

  – *More organized, Less redundancy (save space??)*
  – *Performance problems?? (indirect references)*

Normalization Example -- Sales Order

Sales Order

Fiction Company
202 N. Main
Mahattan, KS 66502

CustomerNumber:	*1001*	Sales Order Number:	405
Customer Name:	*ABC Company*	Sales Order Date:	2/1/2000
Customer Address:	*100 Points*	Clerk Number:	*210*
	Manhattan, KS 66502	Clerk Name:	*Martin Lawrence*

Item Ordered	Description	Quantity	Unit Price	Total
800	*widgit small*	*40*	*60.00*	*2,400.00*
801	*tingimajigger*	*20*	*20.00*	*400.00*
805	*thingibob*	*10*	*100.00*	*1,000.00*

Order Total

Fields will be as follows:

SalesOrderNo, Date,

CustomerNo, CustomerName, CustomerAdd,

ClerkNo, ClerkName,

ItemNo, Description, Qty, UnitPrice

ItemNo -> Description
SalesOrderNo, ItemNo -> Qty, UnitPrice
SalesOrderNo -> Date, CustomerNo, CustomerName, CustomerAdd, ClerkNo, ClerkName
CustomerNo -> CustomerName, CustomerAdd
ClerkNo -> ClerkName

Normalization: First Normal Form

- Separate Repeating Groups into New Tables.
- ***Repeating Groups:*** Fields that may be repeated several times for one document/entity
- The primary key of the new table (repeating group) is always a composite key;

- Relations in 1NF:
  - *SalesOrderNo, ItemNo, Description, Qty, UnitPrice*
  - *SalesOrderNo, Date, CustomerNo, CustomerName, CustomerAdd, ClerkNo, ClerkName*

Normalization: Third Normal Form

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

- **Relations in 3NF**
  - *Customers: <u>CustomerNo</u>, CustomerName, CustomerAdd*
  - *Clerks: <u>ClerkNo</u>, ClerkName*
  - *Inventory Items: ItemNo, Description*
  - *Sales Orders: <u>SalesOrderNo</u>, Date, CustomerNo, ClerkNo*
  - *SalesOrderDetail: <u>SalesOrderNo</u>, <u>ItemNo</u>, Qty, UnitPrice*

Exercise Problem 3NF

- Consider the universal relation $R = \{A, B, C, D, E, F, G, H, I, J\}$ and the set of functional dependencies
- $F = \{ \{A, B\} \rightarrow \{C\},$
  - $\{A\} \rightarrow \{D, E\},$
  - $\{B\} \rightarrow \{F\},$
  - $\{F\} \rightarrow \{G, H\},$
  - $\{D\} \rightarrow \{I, J\} \}.$

- **What is the key for *R*?**
- **Decompose *R* into 3NF relations**
- **Apply the dependency preserving synthesis algorithm to decompose *R* into 3NF relations**
- **Decompose *R* into BCNF relations.**

Solution

- AB is a key.
  - *This can be determined by calculating AB+ with respect to the set F.*
- 2NF
  - *R1 (A, B, C)*
  - *R2 (A,D,E,I,J)*
  - *R3 (B, F, G, H)*
- 3NF
  - *R1 (A, B, C)*
  - *R2.1 (A,D,E)*
  - *R2.2 (D, I, J)*
  - *R3.1 (B, F)*
  - *R3.2 (F, G, H)*

DECOMPOSITION REVISITED

Properties of Relational Decomposition

Attribute preservation condition:

- *Each attribute in R will appear in at least one relation schema R_i in the decomposition*

Dependency Preservation Property

- It is not necessary that the exact dependencies specified in *F* appear themselves in individual relations of the decomposition *D*.
- It is sufficient that the union of the dependencies that hold on the individual relations in *D* be equivalent to *F*.

Lossless (Non-additive) Join Property

- *lossless refers to loss of information, not to loss of tuples.*
- *In fact, for "loss of information" a better term is "<u>addition of spurious information</u>"*

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 $((R_1 \cap R_2) \rightarrow (R_1 - R_2))$ is in F^+, or**
    - **FD $((R_1 \cap R_2) \rightarrow (R_2 - R_1))$ is in F^+.**

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

Example

A universal relation ***R(SSN, Ename, Pnumber, Pname, Plocation, Hours),***

with FDs

SSN-> Ename

Pnumber -> Pname, Plocation

SSN, Pnumber -> Hours

is decomposed into

R1 (Ename, Pnumber)

R2(SSN, Pnumber, Pname, Plocation, Hours)

Is it lossy or lossless ?

Activity: Decompositions Lossless or Lossy ?

A universal relation ***R(SSN, Ename, Pnumber, Pname, Plocation, Hours),***

with FDs

SSN-> Ename

Pnumber -> Pname, Plocation

SSN, Pnumber -> Hours

is decomposed into

EMP

SSN	ENAME

PROJECT

PNUMBER	PNAME	PLOCATION

WORKS_ON

SSN	PNUMBER	HOURS

Is it lossy or lossless ?

Decompositions

- Determine whether each decomposition has
  - *The dependency preservation property, and*
  - *the lossless join property, with respect to F.*
- Also determine the normal form of each relation in the decomposition

R = {A, B, C, D, E, F, G, H, I, J}

F = { {A, B}→{C},
{A}→{D, E},
{B}→{F},
{F}→{G,H},
{D}→{I, J} }.

- **D1** = {R1, R2, R3, R4, R5} ;
  - *R1 = {A, B, C} , R2 = {A, D, E} , R3 = {B, F} , R4 = {F, G, H} , R5 = {D, I, J}*
- **D2** = {R1, R2, R3} ;
  - *R1 = {A, B, C, D, E} , R2 = {B, F, G, H} , R3 = {D, I, J}*
- **D3** = {R1, R2, R3, R4, R5} ;
  - *R1 = {A, B, C, D} , R2 = {D, E} , R3 = {B, F} , R4 = {F, G, H} , R5 = {D, I, J}*

MVDS

A multi-value dependency (MVD) is another type of dependency that could hold in our data, ***which is not captured by FDs***

Multivalued Dependencies

(a) **EMP**

ENAME	PNAME	DNAME
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

Are there any functional dependencies that might hold here?

(b) **EMP_PROJECTS**

ENAME	PNAME
Smith	X
Smith	Y

EMP_DEPENDENTS

ENAME	DNAME
Smith	John
Smith	Anna

Multivalued Dependencies (MVD)

- **MVD** $X \longrightarrow\!\!> Y$ specifies the following constraint on any state r of R:

  If two tuples t_1 and t_2 exist in r such that $t_1[X] = t_2[X]$, then two tuples t_3 and t_4 should also exist in r with the following properties, where we use Z to denote $(R - (X \cup Y))$:

- $t_3[X] = t_4[X] = t_1[X] = t_2[X]$.
- $t_3[Y] = t_1[Y]$ and $t_4[Y] = t_2[Y]$.
- $t_3[Z] = t_2[Z]$ and $t_4[Z] = t_1[Z]$.

Multivalued Dependencies (MVD)

- **MVD** $X \longrightarrow\!\!> Y$ specifies the following constraint on any state r of R:

  If two tuples t_1 and t_2 exist in r such that $t_1[X] = t_2[X]$, then two tuples t_3 and t_4 should also exist in r with the following properties, where we use Z to denote $(R - (X \cup Y))$:

- $t_3[X] = t_4[X] = t_1[X] = t_2[X]$.
- $t_3[Y] = t_1[Y]$ and $t_4[Y] = t_2[Y]$.
- $t_3[Z] = t_2[Z]$ and $t_4[Z] = t_1[Z]$.

Example: MVD

Customer(name, addr, phones, itemsLiked)

- A customer's phones are independent of the items they like.
  - *name* $\twoheadrightarrow$ *phones* *and* *name* $\twoheadrightarrow$ *itemsLiked*.
- Thus, each of a customer's phones appears with each of the item they like in all combinations.
- This repetition is unlike FD redundancy.
  - *name* $\rightarrow$ *addr* *is the only FD.*

Example: MVD

If we have first two tuples:

name	addr	phones	itemsLiked
Ali	Xyz	P1	Cake
Ali	Xyz	P2	Chips

Then last two tuples must also be in the relation

Example: MVD

If we have first two tuples:

name	addr	phones	itemsLiked
Ali	Xyz	P1	Cake
Ali	Xyz	P2	Chips
Ali	***Xyz***	***P1***	***Chips***
Ali	***Xyz***	***P2***	***Cake***

Then last two tuples must also be in the relation

Fourth Normal Form

- Relation R is in **4NF** with respect to a set of dependencies *F* if, for every *nontrivial* multivalued dependency $X \longrightarrow\!\!> Y$ in F^+, X is a superkey for R.
  - *F^+ includes functional dependencies and multivalued dependencies*
- An MVD X —>> Y in R is called a **trivial MVD** if

  a) *Y is a subset of X, or*

  b) $X \cup Y = R$.
- **Example:** EMP_PROJECTS has the trivial MVD Ename —>> Pname.
- An MVD that satisfies neither (a) nor (b) is called a **nontrivial MVD.**

EMP_PROJECTS

ENAME	PNAME
Smith	X
Smith	Y

Fourth Normal Form

(a) **EMP**

ENAME	PNAME	DNAME
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John
Brown	W	Jim
Brown	X	Jim
Brown	Y	Jim
Brown	Z	Jim
Brown	W	Joan
Brown	X	Joan
Brown	Y	Joan
Brown	Z	Joan
Brown	W	Bob
Brown	X	Bob
Brown	Y	Bob
Brown	Z	Bob

(b) **EMP_PROJECTS**

ENAME	PNAME
Smith	X
Smith	Y
Brown	W
Brown	X
Brown	Y
Brown	Z

EMP_DEPENDENTS

ENAME	DNAME
Smith	Anna
Smith	John
Brown	Jim
Brown	Joan
Brown	Bob

Fourth Normal Form

Algorithm: Relational decomposition into 4NF relations with non-additive join property

Input: A relation R and a set of functional and multivalued dependencies F.

1. Set D := { R };
2. While there is a relation schema *Q* in *D* that is not in 4NF do

   { choose a relation schema *Q* in *D* that is not in 4NF;

   find a nontrivial MVD $X \twoheadrightarrow Y$ in *Q* that violates 4NF;

   replace *Q* in *D* by two relation schemas $(Q - Y)$ and $(X \cup Y)$;

   };

MVD

- Whenever $X \longrightarrow\!\!> Y$ holds, we say that X **multidetermines** Y.
- Because of the symmetry in the definition,
- whenever $X \longrightarrow\!\!> Y$ holds in R, so does $X \longrightarrow\!\!> Z$. Hence, $X \longrightarrow\!\!> Y$ implies $X \longrightarrow\!\!> Z$ and
- therefore it is sometimes written as $X \longrightarrow\!\!> Y|Z$.

Book Exercise Problem

Book_Name	Author	Edition	Copyright_Year
DB_fundamentals	Navathe	4	2004
DB_fundamentals	Elmasri	4	2004
DB_fundamentals	Elmasri	5	2007
DB_fundamentals	Navathe	5	2007

a) Based on a common-sense understanding of the above data, what are the possible candidate keys of this relation?
   a) Assume that only one edition of a book comes in a one year
   b) Each future editions of a book have the name of all the authors on first edition.

b) Justify that this relation has the MVD {Book} –>> {Author} | {Edition, Year}.

c) What would be the decomposition of this relation based on the above MVD? Evaluate each resulting relation for the highest normal form it possesses

Trip Exercise Problem

- Consider the following relation:
  TRIP (Trip_id, Start_date, Cities_visited, Cards_used)

- This relation refers to business trips made by company salespeople.
- Suppose the TRIP has a single Start_date but involves many Cities and salespeople may use multiple credit cards on the trip. Make up a mock-up population of the table.
  - *Discuss what FDs and/or MVDs exist in this relation.*
  - *Show how you will go about normalizing the relation.*

FD REVISION

Example: Minimal Cover

- Given a relation R (A, B, C, D, E, F) and a set of FDs
- F = {A $\rightarrow$BCE,
- CD $\rightarrow$EF,
- E $\rightarrow$F,
- B $\rightarrow$ E,
- AB $\rightarrow$ CF}.
- Compute the minimal cover for F

- Is minimal cover in 3NF ? If no convert to 3NF relations
- Is it in BCNF ? If no convert to BCNF relations

Example

- R(A, B, C, D) with FDs: AB->C, C->D, D->A

- Find keys
- Is it in 3NF

- Is it BCNF

Yet another Example

- R(ABCDE) with FD’s AB->C, C->D, D->B, D->E
- Keys: ?
- Current highest normal form ?

- Convert to BCNF
- Convert to 3NF
- Convert to 3NF using minimal cover

Summary

- Constraints allow one to reason about **redundancy** in the data

- Normal forms describe how to **remove** this redundancy by **decomposing** relations
  - *Elegant—by representing data appropriately certain errors are essentially impossible*
  - *For FDs, BCNF is the normal form.*

- A tradeoff for insert performance: 3NF

What have we covered

1. Overview of design theory
2. Data anomalies & constraints
3. Functional dependencies
4. Closure
5. How to Find FDs
6. Minimal Cover
7. Normal forms
8. Loss Less Join

Decomposition algo