CS 5200
Database Systems

Schema Refinement and Normal Forms

Textbook Reference
Database Management Systems
Reading: Chapter 19. Concentrate on sections19.1-19.6 (except 19.5.2)

Hazra Imran

.

Data Redundancy and Data Integrity

"A man with a watch knows what time it is. A man with two watches is never sure."

- Lee Segal

- Redundant data takes up a great deal of extra disk space.
- If the redundant data has to be updated, it takes additional time to do so. This can be a major performance issue.
- When all copies of redundant data are not updated consistently, a data integrity problem exists.

Imagine an entity for mailing addresses at any UBC:

Name is faculty name

Meets all the criteria that we have for an entity
There is nothing wrong with this entity

What would an instance look like?

Name	Department	Mailing Location
Max Charles	Computer Science	201-2366 Main Mall
Brett Zhang	Computer Science	201-2366 Main Mall
Ray Chang	Cmputer Science	201-2366 Main Mall
Gladys	Computer Science	201-2366 Main Mall
Hazra Imran	Computer Science	201-2366 Man Mall
Hazra Imran	Math	121-1984 Mathematics Rd
Brian Marcus	Math	121-1984 Mathematics Rd

Is this a good design? What are some problems that might happen with this design?

Anomalies

Combining the two different ideas leads to some bad anomalies.

Name	Department	Mailing Location
Max Charles	Computer Science	201-2366 Main Mall
Brett Zhang	Computer Science	201-2366 Main Mall
Ray Chang	Cmputer Science	201-2366 Main Mall
Gladys	Computer Science	201-2366 Main Mall
Hazra Imran	Computer Science	201-2366 Man Mall
Hazra Imran	Math	121-1984 Mathematics Rd
Brian Marcus	Math	121-1984 Mathematics Rd

- Typically occur in poorly structured databases
  1. Deletion Anomaly
  2. Insertion Anomaly
  3. Update Anomaly

Any better Solution?

Name	Department	Mailing Location
Max Charles	Computer Science	201-2366 Main Mall
Brett Zhang	Computer Science	201-2366 Main Mall
Ray Chang	Computer Science	201-2366 Main Mall
Gladys	Computer Science	201-2366 Main Mall
Hazra Imran	Computer Science	201-2366 Main Mall
Hazra Imran	Math	121-1984 Mathematics Rd
Brian Marcus	Math	121-1984 Mathematics Rd

With this design, insertion, deletion and updating may be problematic

Any better solution?

One Possible Solution

Name	Department
Max Charles	Computer Science
Brett Zhang	Computer Science
Ray Chang	Computer Science
Gladys	Computer Science
Hazra Imran	Computer Science
Hazra Imran	Math
Brian Marcus	Math

Department	Mailing Location
Computer Science	201-2366 Main Mall
Math	121-1984 Mathematics Rd

How do I know for sure if departments have only one address?

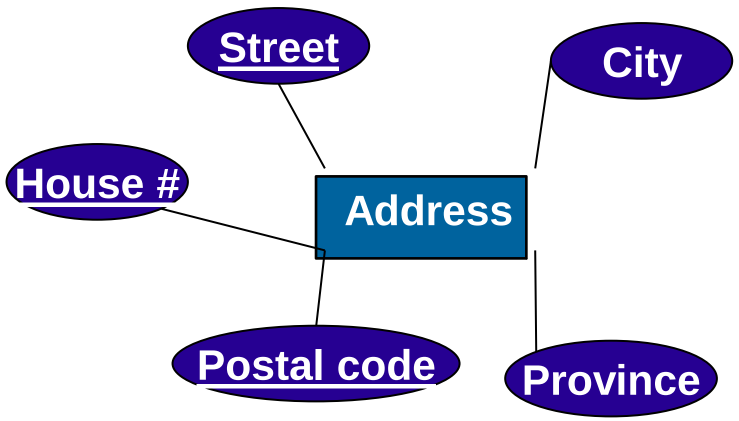
- Databases allow you to say that one attribute determines another through a *functional dependency*.

Computer Science **201-2366 Main Mall**

- So if Department determines Address but not Name, we say that there's a functional dependency from Department to Address. But Department is NOT a key.
- We write *Department → Address* to say each dept has at most one mailing location.
- Such statements are integrity constraints (ICs) called *functional dependencies* (FDs).

Another FD example

- Another example:
  Address(House#, Street, City, Province, PostalCode)



Postal Code → City, Province   V6T 1Z4 → Vancouver , BC

Postal Code → House# ?   V6T 1Z4 → 2238, 2356, 2386 ?

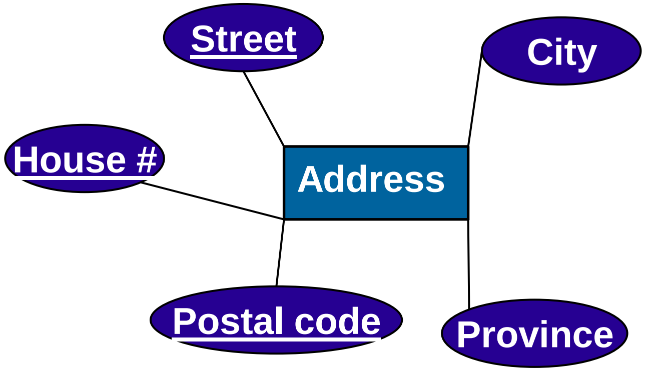
Functional Dependencies (FDs)

- A functional dependency X→Y (where X & Y are sets of attributes) holds if for every legal instance, for all tuples *t1*, *t2* :

  if $t1.X = t2.X$ ***then***

  $(t1.Y = t2.Y)$

- Example: PostalCode → City, Province



House #	Street	City	Province	Postal Code
101	Main Street	Vancouver	BC	V6A 2S5
103	Union Street	Vancouver	BC	V6A 2S5

- i.e., given two tuples in *r*, if the X values agree, then the Y values must also agree
- Also can be read as X *determines* Y or Y is functionally *determined* by X

Huh? Which functional dependencies where?

- A FD is a statement about *all* allowable instances.
  - Must be identified by application semantics
  - Given some instance *r1* of R, we can check if r1 violates some FD *f*, but we cannot tell if *f* holds over R! (i.e., whether f holds for all ***allowable*** instances of R)

Postal Code → Street ?
Department → Mailing Location?

- We'll concentrate mostly on cases where there's a single attribute on the RHS: (e.g., PostalCode → Province)

- There are boring, *trivial* cases:
  - e.g. PostalCode, House# → PostalCode

- We'll concentrate on the non-boring ones

What do we need to know to split apart addresses without losing information?

- FDs tell us when we’re storing redundant information
- Reducing redundancy helps eliminate anomalies and save storage space
- We’d like to split apart tables without losing information
- But first, we need to know both
  - what FDs are *explicit* (given) and
  - what FDs are *implicit* (can be derived)
- Among other things, this can help us derive additional keys from the given keys (spare keys are handy in databases, just like in real life – we’ll see why shortly)

Functional dependencies & keys all together

- In a functional dependency, a set of attributes determines other attributes, e.g., AB→C, means A and B together determine C
- A trivial FD determines what you already have, eg., AB→B
- A key is a minimal set of attributes determining the rest of the attributes of a relation
  For example, R(<u>House #</u>, <u>Street</u>, City, Province, <u>Postal Code</u>)
- Given a set of (explicit) functional dependencies, we can determine others…

Deriving Additional FDs: the basics

- Given some FDs, we can often infer additional FDs:
  - *studentid* → *city*, *city* → *acode* implies
  - *studentid* → *acode*
- An FD *fd* is *implied by* a set of FDs *F* if *fd* holds whenever all FDs in *F* hold.
  - ***closure of F*** : the set of all FDs implied by *F*.

- Armstrong's Axioms (X, Y, Z are sets of attributes):
  - *Reflexivity*: If Y $\subseteq$ X, then X → Y
    e.g., city,major→city
  - *Augmentation*: If X → Y, then X Z → Y Z for any Z
    e.g., if sid→city, then sid,major → city,major
  - *Transitivity*: If X → Y and Y → Z, then X → Z
    sid → *city*, *city* → *areacode* implies *sid* → *areacode*
- These are *sound* and *complete* inference rules for FDs.

Deriving Additional FDs: the extended dance remix

- Couple of additional rules (that follow from axioms):
  - ***Union:*** If X→Y and X→Z, then X→Y Z
    e.g., if sid→acode and sid→city, then sid→acode,city
  - ***Decomposition:*** If X→Y Z, then X→Y and X→Z
    e.g., if sid→acode,city then sid→acode, and sid→city

Closure

- *Closure* is a fool-proof method of checking FDs.
- Closure for a set of attributes X is denoted X^+

- Given a set of attributes ***A*** and a set of dependencies C, we want to find all the other attributes that are functionally determined by ***A***.

Closure

- X^+ includes all attributes of the relation IFF X is a (super)key

- **Algorithm for finding Closure of X:**

- **Let Closure = X**

  Until Closure doesn't change do

  if $a_1, \ldots, a_n \rightarrow C$ is a FD and

  $\{a_1, \ldots, a_n\} \in$ Closure and C is not in the Closure

  Then add C to the Closure

Computing the Closure of Attributes - Example

- Let's consider a relation with attributes A, B, C, D, E and F. Suppose that this relation satisfies the FD's:

  $AB \rightarrow C$,
  $BC \rightarrow AD$,
  $D \rightarrow E$,
  $CF \rightarrow B$.

What is $\{A,B\}^+$?

•**Let Closure = X**

Until Closure doesn't change do

if $a_1, \ldots, a_n \rightarrow C$ is a FD and

$\{a_1, \ldots, a_n\} \in$ Closure and C is not in the Closure

Then add C to the Closure

Closures and Keys

- Notice that $\{A_1, A_2, \dots, A_n\}^+$ is the set of **all** attributes if and only if $\{A_1, A_2, \dots, A_n\}$ is a **superkey** for the relation in question.

- Only then does $A_1, A_2, \dots, A_n$ functionally determines all the attributes.

- We can test if $A_1, A_2, \dots, A_n$ is a **key** for a relation by checking:
  - **first** that $\{A_1, A_2, \dots, A_n\}^+$ contains all attributes,
  - **and** that for **no** subset **S** of $\{A_1, A_2, \dots, A_n\}^+$, is S^+ the set of all attributes.

Approaching Normality

- Role of FDs in detecting redundancy:
  - Consider a relation R with 3 attributes, A B C.
    - No FDs hold: There is no redundancy here.
    - Given A → B: Several tuples could have the same A value, and if so, they'll all have the same B value!
- ***Normalization***: the process of removing redundancy from data

Normal Forms: Why have one rule when you can have four?

- Provide guidance for table refinement/reducing redundancy.

- Four important normal forms:
  - ***First normal form(1NF)***
  - ***Second normal form (2NF)***
  - ***Third normal form (3NF)***
  - ***Boyce-Codd Normal Form (BCNF)***

- If a relation is in a certain *normal form*, certain problems are avoided/minimized.

- Normal forms can help decide whether decomposition (i.e., splitting tables) will help.

1NF

- Each attribute in a tuple must have only one value
  - E.g., for “postal code” you can’t have both V6T 1Z4 and V6S 1W6

First Normal Form (1NF) - *A table is in 1NF if each row is unique and no column in any row contains multiple values*

- Every relational table is, by definition, in 1NF

How to Normalize?

Normalizing to 1NF involves eliminating groups of related multi-valued columns

1NF

VET CLINIC CLIENT

ClientID	ClientName	PetNo	PetName	PetType
111	Lisa	1	Tofu	Dog
222	Lydia	1 2 3	Fluffy JoJo Ziggy	Dog Bird Snake
333	Jane	1 2	Fluffy Cleo	Cat Cat

Relational or non-relational table?

Non-relational table (not in 1NF).

1NF

VET CLINIC CLIENT

ClientID	ClientName	PetNo	PetName	PetType
111	Lisa	1	Tofu	Dog
222	Lydia	1 2 3	Fluffy JoJo Ziggy	Dog Bird Snake
333	Jane	1 2	Fluffy Cleo	Cat Cat

Normalizing the table to 1NF by increasing the number of records

VET CLINIC CLIENT

ClientID	ClientName	PetNo	PetName	PetType
111	Lisa	1	Tofu	Dog
222	Lydia	1	Fluffy	Dog
222	Lydia	2	JoJo	Bird
222	Lydia	3	Ziggy	Snake
333	Jane	1	Fluffy	Cat
333	Jane	2	Cleo	Cat

1NF

VET CLINIC CLIENT

ClientID	ClientName	PetNo	PetName	PetType
111	Lisa	1	Tofu	Dog
222	Lydia	1	Fluffy	Dog
		2	JoJo	Bird
		3	Ziggy	Snake
333	Jane	1	Fluffy	Cat
		2	Cleo	Cat

Normalizing the table to 1NF by creating a new, separate table

VET CLINIC CLIENT

ClientID	ClientName
111	Lisa
222	Lydia
333	Jane

PET

ClientID	PetNo	PetName	PetType
111	1	Tofu	Dog
222	1	Fluffy	Dog
222	2	JoJo	Bird
222	3	Ziggy	Snake
333	1	Fluffy	Cat
333	2	Cleo	Cat

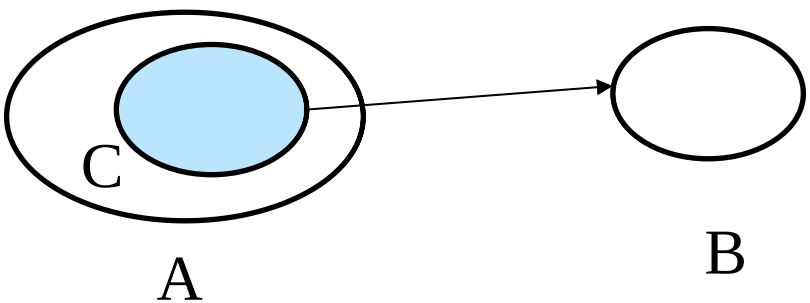
2NF Proper subset

- A proper subset of a set is a subset that is strictly contained in S and so necessarily excludes at least one member of S
- For example, consider a set {1,2,3,4,5}.

Partial FDs and 2NF

Partial FDs:

- A FD, A →B is a partial FD, if some attribute of A can be removed and the FD still holds.
- Formally, there is some proper subset of A, C ⊂ A, such that C →B



- Let us call attributes which are part of some candidate key, key attributes, and the rest non-key attributes.

2NF

A relation is in second normal form (2NF) if:

- it is in 1NF and
- It includes no partial dependencies:
  - No **non-key** attribute is dependent on only portion of the key

- If a relation has a single-column primary key, then there is no possibility of partial functional dependencies
  - Such a relation is automatically in 2NF and it does not have to be normalized to 2NF
- If a relation with a composite primary key has partial dependencies, then it is not in 2NF, and it has to be normalized it to 2NF

2NF with example

CityAddress (City, Street, HouseNumber, HouseColor, CityPopulation)

City, Street, HouseNumber → HouseColor
City → CityPopulation

- CityPopulation is functionally dependent on the City which is a proper subset of the key

House#	Street	City	HouseColor	CityPopulation
12	Burrad St	Vancouver	White	600,218
21	Burrad St	Vancouver	Red	600,218
23	Hamilton St	Richmond	Blue	216,288
12	Burrad St	Richmond	White	216,288

2NF

How to Normalize?

1. Normalization of a relation to 2NF creates additional relations for each set of partial dependencies in a relation
    - The primary key of the additional relation is the portion of the primary key that functionally determines the columns in the original relation
    - The columns that were partially determined in the original relation are part of the additional table

2. The original table remains after the process of normalizing to 2NF, but it no longer contains the partially dependent columns

2NF

City, Street, HouseNumber → HouseColor
City → CityPopulation

House#	Street	City	HouseColor	CityPopulation
12	Burrad St	Vancouver	White	600,218
21	Burrad St	Vancouver	Red	600,218
23	Hamilton St	Richmond	Blue	216,288
12	Burrad St	Richmond	White	216,288

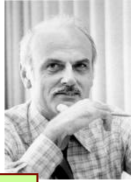
House#	Street	**City**	HouseColor
12	Burrad St	Vancouver	White
21	Burrad St	Vancouver	Red
23	Hamilton St	Richmond	Blue
12	Burrad St	Richmond	White

City, Street, HouseNumber → HouseColor

City	CityPopulation
Vancouver	600,218
Richmond	216,288

City → CityPopulation

Boyce-Codd Normal Form (BCNF)



Raymond Boyce & Ted Codd

A relation R is in BCNF if:

If X → b is a non-trivial dependency in R, then X is a superkey for R

(Must be true for every such dependency)

Recall: A dependency is trivial if the LHS contains the RHS, e.g., City, Province→ City is a trivial dependency

In English (though a bit vague):

Whenever a set of attributes of *R* determine another attribute, it should determine ***all*** the attributes of *R*.

What do we want?
Guaranteed freedom from redundancy!

A relation may be BCNF already

- bonus fact: all two attribute relations are in BCNF

What do we want? Guaranteed freedom from redundancy!

A relation may be BCNF already – bonus fact: all two attribute relations are in BCNF

R(X,Y)
- No FD so no redundancy
- X→Y so X is key, so in BCNF
- Y→X so Y is key, so in BCNF
- Y→X and X→Y, both X and Y are keys, so in BCNF

Otherwise? Decomposition

Before decomposition, lets look at each one

What if a relation is not in BCNF? Decomposing a Relation

- A *decomposition* of R replaces R by two or more relations s.t.:
  - Each new relation contains a subset of the attributes of R (and no attributes not appearing in R), and
  - Every attribute of R appears in at least one new relation.

"Shhhhh! ... the Maestro is decomposing!"

How can we decompose a relation w/o losing information?

- Address(House#,Street,City,Province,Postal Code).

Address(House#,Street,Postal Code) CP(City, Province, Postal Code)

- ➢ Does the above decomposition loose information?
- ➢ What does it mean to loose information?
- ➢ How can we tell if we loose?

A sneak preview: The join

- Definition: $R_1 \bowtie R_2$ is the (natural) join of the two relations; i.e., each tuple of R_1 is concatenated with every tuple in R_2 having the same values on the common attributes.

R_1

A	B
1	2
4	5
7	2

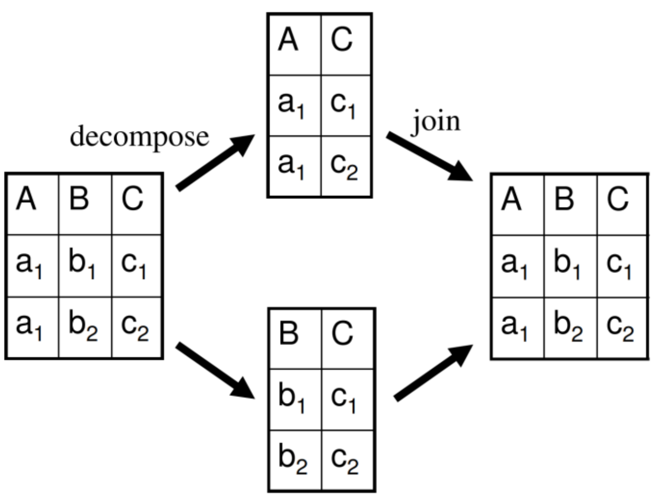
R_2

B	C
2	3
5	6
2	8

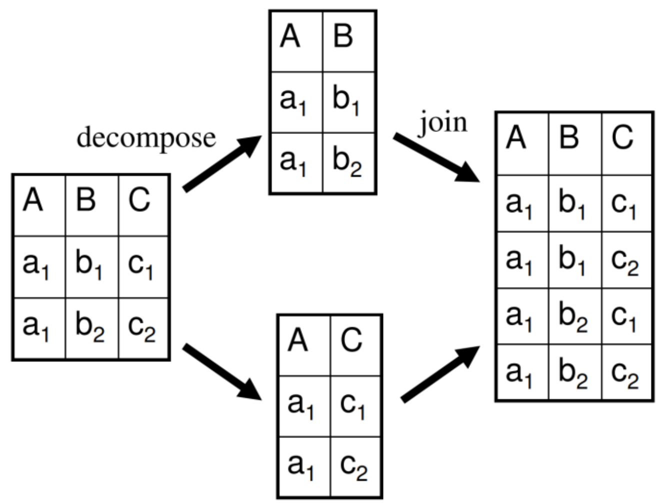
$R_1 \bowtie R_2$ →

Decompositions

Lossless



Lossy



Decomposition Goals

1. Lossless Join

If we break a relation, R, into bits, when we put the bits back together, we should get exactly R back again

Note: The word loss in lossless refers to loss of information, not to loss of tuples. In fact, for "loss of information" a better term is "addition of spurious information ".

We should get back exactly the original tuples.

2. Dependency Preservation

All the original FD's should hold.

3. No Anomalies

A decomposition should contain a minimum amount of redundancy.

Lossless-join Decomposition

- A decomposition of R into R1 and R2 is lossless join if and only if at least one of the following dependencies is in F^+
  - $R1 \cap R2 \rightarrow R1$
  - $R1 \cap R2 \rightarrow R2$

Lossless-Join Decompositions

- We should be able to construct the instance of the original table from the instances of the tables in the decomposition

SName	TutorialNum	TutorialGroup	Marks
John	T1	G1	2
John	T2	G3	1
Linda	T1	G2	1

Lossless-Join Decompositions

- We should be able to construct the instance of the original table from the instances of the tables in the decomposition

SName	TutorialNum	TutorialGroup	Marks
John	T1	G1	2
John	T2	G3	1
Linda	T1	G2	1

decompose in two tables

SName	TutorialGroup	Marks
John	G1	2
John	G3	1
Linda	G2	1

TutorialNum	Marks
T1	2
T2	1
T1	1

Lossless-Join Decompositions

- We should be able to construct the instance of the original table from the instances of the tables in the decomposition

SName	TutorialNum	TutorialGroup	Marks
John	T1	G1	2
John	T2	G3	1
Linda	T1	G2	1

decompose in two tables

SName	TutorialGroup	Marks
John	G1	2
John	G3	1
Linda	G2	1

TutorialNum	Marks
T1	2
T2	1
T1	1

After join

SName	TutorialNum	TutorialGroup	Marks
John	T1	G1	2
John	T2	G3	1
John	T1	G3	1
Linda	T2	G2	1
Linda	T1	G2	1

Previously… Lossless-Join Decompositions

- A decomposition $\{R_1, R_2\}$ of R is lossless if and only if the common attributes of R_1 and R_2 form a key for either schema, that is
- $R_1 \cap R_2 \rightarrow R_1$ or $R_1 \cap R_2 \rightarrow R_2$

In the previous **example** we had

R = {SName, Tutorial, TGroup, Mark} and

F = {SName, TutorialNum → TGroup, Mark}

R1 = {SName, TGroup, Mark}

R2 = {Tutorial, Mark}

SName	TutorialGroup	Marks
John	G1	2
John	G3	1
Linda	G2	1

TutorialNum	Marks
T1	2
T2	1
T1	1

R1 ∩ R2 = {Mark} and it is not a key of either R1 or R2

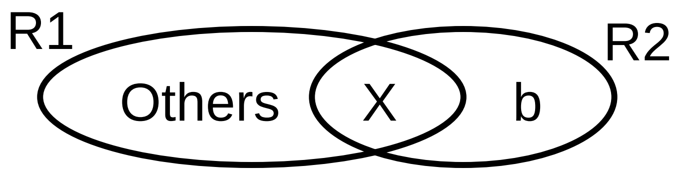
Therefore, decomposition {R1, R2} is lossy

How do we decompose into BCNF losslessly?

- Let R be a relation with attributes A, and FD be a set of FDs on R s.t. all FDs determine a single attribute

- Pick any $f \in$ FD that violates BCNF of the form $X \rightarrow b$
- Decompose R into two relations: $R_1(A-b)$ & $R_2(X \cup b)$
- Recurse on R_1 and R_2 using FD

- Pictorially:



Note: answer may vary depending on order you choose. That's okay

FD Preservation

- Given a relation R and a set of FDs F, decompose R into R1 and R2
- Suppose
    - R1 has a set of FDs F1
    - R2 has a set of FDs F2
  - F1 and F2 are computed from F.
- The decomposition is dependency preserving if by enforcing F1 over R1 and F2 over R2, we can enforce F over R

Example : FD Preservation

PID	Name	Age	canDrive	Phone#
P1	John	14	No	604 111 1111
P2	Raj	28	Yes	604 111 1111
P3	John	14	No	604 333 3333

PID → Name
Age → canDrive

PID	Name
P1	John
P2	Raj
P3	John

PID → Name

Age	canDrive
14	No
28	Yes

Age → canDrive

PID	Age	Phone#
P1	14	604 111 1111
P2	28	604 111 1111
P3	14	604 333 3333

After you decompose, how do you know which FDs apply?

- For an FD $X \rightarrow b$, if the decomposed relation S contains $\{X \cup b\}$, and $b \in X^+$ then the FD holds for S:

- For example. Consider relation R(A,B,C,D,E) with functional dependencies $AB \rightarrow C$, $BC \rightarrow D$, $CD \rightarrow E$, $DE \rightarrow A$, and $AE \rightarrow B$. Project these FD's onto the relation S(A,B,C,D).

- Does $AB \rightarrow D$ hold?
  - First check if A, B and D are all in S? They are
  - Find AB^+= ABCDE
  - Then yes $AB \rightarrow D$ does hold in S.

- Does $CD \rightarrow E$ hold?
- No

Yet Another BCNF Example: Do implicit FDs matter?

- Implicit FDs are just as important than the explicit ones.

- When decomposing into BCNF other than the given FDs, we should also consider implicit FDs.

Yet Another BCNF Example: Do implicit FDs matter?

Relation: R(ABCDEF)

FD = $A \rightarrow B$, $DE \rightarrow F$, $B \rightarrow C$

Yet Another BCNF Example: Do implicit FDs matter?

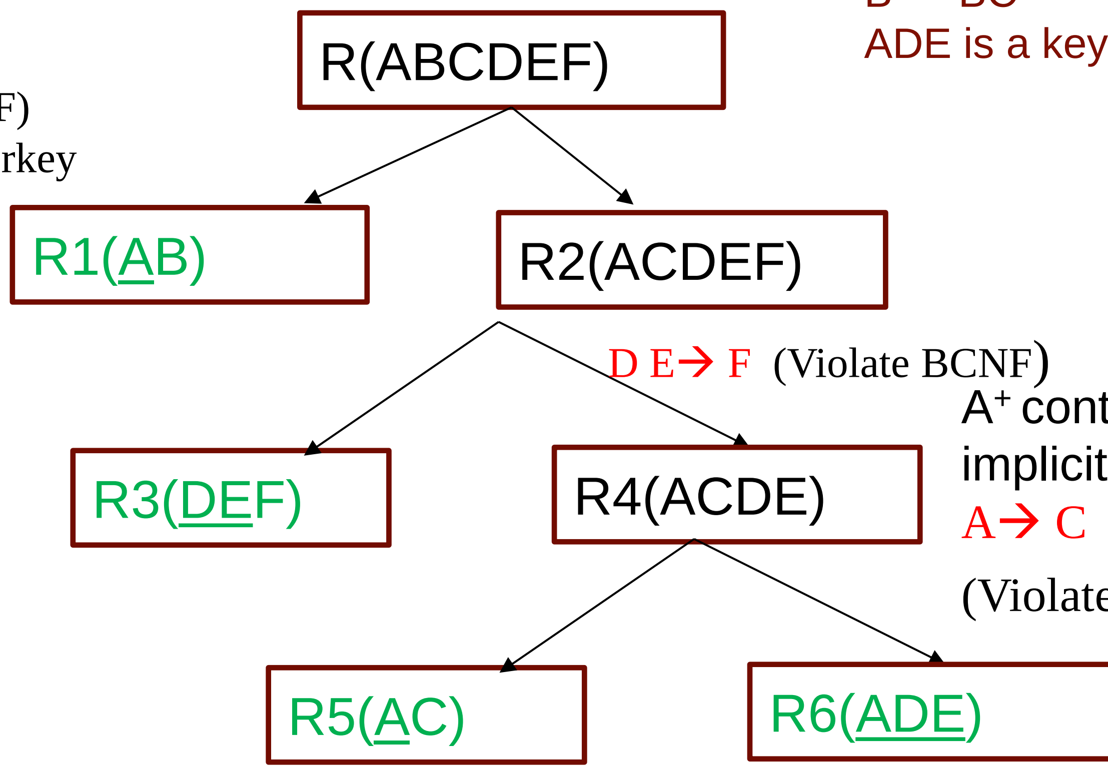
Relation: R(ABCDEF)

FD = A→B, DE→ F , B→C

Find closure of the following

A^+ = ABC
DE^+ = DEF
B^+ = BC
ADE is a key

A→ B
(Violate BCNF)
- A is not superkey



A^+ contains C so an implicit FD A→C
A→ C (implicit)
(Violate BCNF)

This BCNF stuff is great and easy!

- Guaranteed that there will be no redundancy of data

Lossless Join (Yes)
Dependency Preservation (No ☹)

- Easy to understand
  - No need to know any keys.
  - Only superkeys are needed.
  - Don’t worry about ALL superkeys
    - Only check if LHS of each FD is a superkey.
    - Use closure test !

- So what is the main problem with BCNF?
  - For one thing, BCNF may not preserve all dependencies

An illustrative BCNF example

Unit	Company	Product

$Unit \rightarrow Company$

$Company, Product \rightarrow Unit$

Unit	Company

Unit	Product

$Unit \rightarrow Company$

We lose the FD: Company, Product → Unit !!

So What's the Problem?

Unit	Company
SKYWill	UBC
Team Meat	UBC

Unit	Product
SKYWill	Databases
Team Meat	Databases

Unit → Company

No problem so far. All *local* FD's are satisfied.

Let's put all the data back into a single table again:

Unit	Company	Product
SKYWill	UBC	Databases
Team Meat	UBC	Databases

Violates the FD: Company, Product → Unit

What is offered by 3NF decomposition?

- Lossless Join   (Yes)
- Dependency Preservation (Yes)

Neither BCNF nor 3NF can guarantee all three!

Decompose too far → can't enforce all FDs.

Not far enough → can have redundancy.

A schema is considered "good" if it is in either BCNF or 3NF.

3NF come to Rescue!

3NF to the rescue!

A relation R is in 3NF if:

If X → b is a non-trivial dependency in R,
then X is a superkey for R } BCNF

or b is part of a minimal key.

(must be true for every such functional dependency)

Note: b must be part of a key *not* part of a superkey (if a key exists, *all* attributes are part of a superkey!)

3NF to the rescue!

Example: R(Unit,Company, Product)

Keys: {Company, Product}, {Unit,Product}

- Unit $\rightarrow$ Company

FDs

- Company, Product $\rightarrow$ Unit

3NF to the rescue!

Example: R(Unit,Company, Product)

Keys: {Company, Product}, {Unit,Product}

FDs

- Unit $\rightarrow$ Company

Not in BCNF. Company part of a key so 3NF

- Company, Product $\rightarrow$ Unit

Company, Product is superkey. BOTH

To decompose into 3NF we rely on the *minimal cover*

Minimal Cover for a Set of FDs

- Sets of functional dependencies may have redundant dependencies that can be inferred from the others .
- Eg: $A \rightarrow C$ is redundant in: $\{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$

- Goal: Transform FDs to be as small as possible

Minimal Cover

- Intuitively, every FD in U is needed, and is "*as small as possible*'' in order to get the same closure as F

F:

A→B,
ABCD→E,
EF→GH,
ACDF→EG

minimal cover →

U:

A→B,
ACD→E,
EF→G,
EF→H,

Let see how to find the minimal cover

Finding minimal covers of FDs

1. Put FDs in standard form (have only one attribute on RHS)
2. Minimize LHS of each FD
3. Delete Redundant FDs

Finding minimal covers of FDs

1. Put FDs in standard form (have only one attribute on RHS)
2. Minimize LHS of each FD
3. Delete Redundant FDs

Example:

$A \rightarrow B$
$ABCD \rightarrow E$
$EF \rightarrow G$
$EF \rightarrow H$
$ACDF \rightarrow EG$

Replace $ACDF \rightarrow EG$ with

- $ACDF \rightarrow E$
- $ACDF \rightarrow G$

Finding minimal covers of FDs

1. Put FDs in standard form (have only one attribute on RHS)
2. Minimize LHS of each FD
3. Delete Redundant FDs

Step 2: Eliminate redundant attributes from LHS.

Algorithm:
Remove B from the left-hand-side of X → A in F if A is in X^+(X-{B},F).

Example:
ABCD→E

ABC~~D~~ : can we get E now?
AB~~C~~D : can we get E now?
A~~B~~CD : can we get E now?
~~A~~BCD : can we get E now?

We check for all subsets by eliminating one attribute at a time

Finding minimal covers of FDs

1. Put FDs in standard form (have only one attribute on RHS)
2. Minimize LHS of each FD
3. Delete Redundant FDs

A→B
ABCD→E
EF→G
EF→H
ACDF→E
ACDF→G

	Reduce LHS	Final FDs
A→B		
ABCD→E		
EF→G		
EF→H		
ACDF→E		
ACDF→G		

Finding minimal covers of FDs

1. Put FDs in standard form (have only one attribute on RHS)
2. Minimize LHS of each FD
3. Delete Redundant FDs

A → B
ACD → E
EF → G
EF → H
~~ACD → E~~
ACDF → G

A→B
ACD→E
EF→G
EF→H
ACDF→G

Finding minimal covers of FDs

	Closure when given **FD is considered**	Closure **when given FD is not considered**	Decision
A →B			
ACD→ E			
EF→G			
EF→H			
ACDF→G			

3NF Synthesis

- Conceptually simpler.
- Given a set of FDs F, obtain a minimal cover F'
  - $\forall$FD $X \rightarrow A \in F'$, create a scheme XA.
  - Resulting decomposition is guaranteed to preserve all FDs (trivially) and each scheme is in 3NF. But no guarantee for LLJ!
  - Easy fix: add any schema that contains a key of the original relation scheme R.

- Revisit previous example:
- R(ABCDE) FD: AB→C, C→D.

Example: Decomposition into 3NF Using a Minimal Cover and 3NF Synthesis

- Suppose we have R(A,B,C) with FDs: A → B, C → B.
  1. Find a minimal cover F'. Already done.
  2. For each FD X→b, add relation Xb to the decomposition for R.
     - Result: R1(A,B) and R2(B,C). Are we done? No.
  3. Does it contain a key? What are the keys of R? AC. Add R3(A,C).

- Let’s see why we need step #3

In decompositions, you can often make some adjustments to make a “better” decomposition

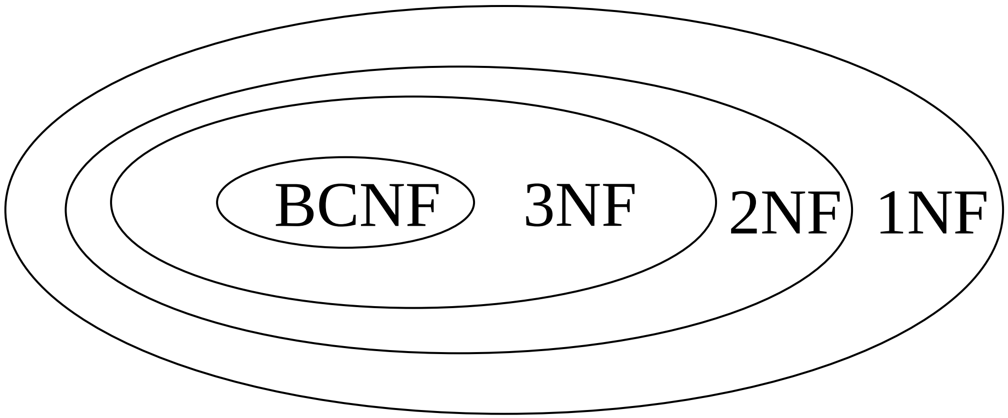
- For example, if,
  R1(ABC)
  R2(CD)
  R3(EFG)
  R4(EF)
  R5(ABEG)

have redundant relations – you don’t need R4 because all information is contained in R3

- You can make the same optimizations in BCNF

Comparing BCNF & 3NF

- BCNF guarantees removal of all anomalies
- 3NF has some anomalies, but preserves all dependencies
- If a relation R is in BCNF it is in 3NF.



Other normal forms

- Further normal forms exist which deal with issues not covered by functional dependencies
  - *Fourth Normal Form* deals with multi-valued dependencies
  - *Fifth Normal Form addresses more complex (and rarer) situations where 4NF is not sufficient*

Normalization and Design

- Most organizations go to 3NF or better
- If a relation has only 2 attributes, it is automatically in 3NF and BCNF
- Our goal is to use lossless-join for all decompositions and preserve dependencies
- BCNF decomposition is always lossless, but may not preserve dependencies
- Good heuristic:
  - Try to ensure that all relations are in at least 3NF
  - Check for dependency preservation