INFS7901
Database Principles

Functional Dependencies and Normal Forms

Rocky Chen

Design Guidelines

Functional Dependencies

Normalization

Informal Design Guidelines

- Informal measures of relational database schema quality and design guidelines
  - Semantics of the attributes
  - Reducing the redundant values in tuples
  - Reducing the null values in tuples
  - Disallowing fake tuples

Guideline 1

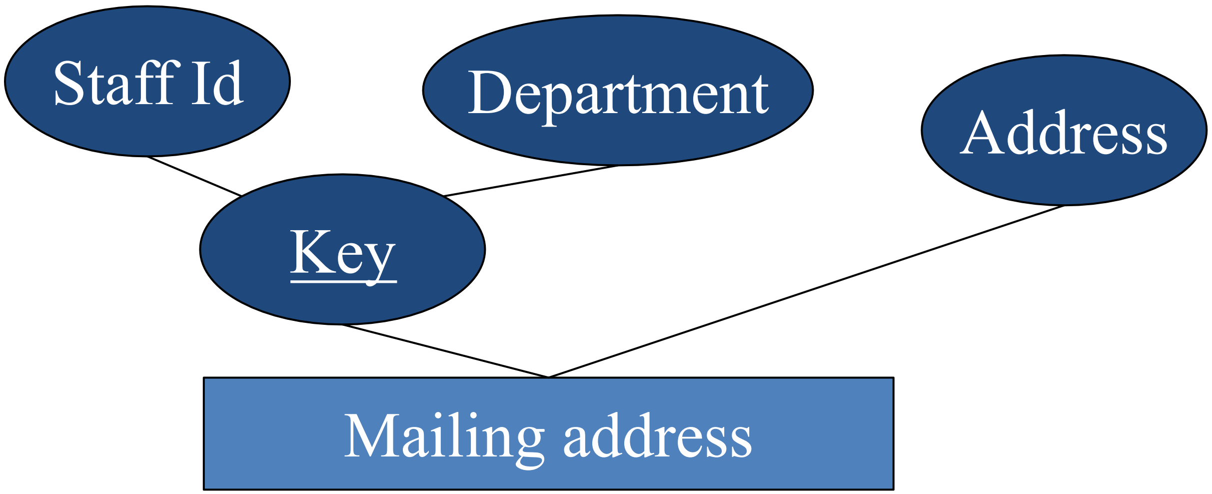
- Design each relation so that it is easy to explain its meaning
- Do not combine attributes from multiple entity types and relationship types into a single relation

Redundant Values in Tuples

- One design goal is to minimize the storage space that base relations occupy
- The way the grouping of attributes into relation schemas is done, has a significant effect on storage space
- In addition, an incorrect grouping may cause *update anomalies* which may result in inconsistent data or even loss of data

A Motivating Example

- Imagine that we have created the following entity for mailing addresses at a university



Meets all the criteria that we have discussed for an entity so far

What Potential Problems You See?

Staff Id	Department	Address
100	Computer Science	78-101 Main Mall
104	Computer Science	78-101 Main Mall
104	Math	69-201 Main Mall
105	Physics	50-205 Main Mall

- **Modification anomaly:** data inconsistency that results from data redundancy.
  - Example: Updating the mailing address of a department.
- **Deletion anomaly:** loss of certain attributes because of the deletion of other attributes.
  - Example: Deleting 105 would lead to deletion of the address of Physics.
- **Insertion anomaly:** Lack of ability to insert some attributes without the presence of other attributes.
  - Example: Storing the mailing address of a department that has no faculty members.

Guideline 2

- Design the base relation schema so that no insertion, deletion, or modification anomalies occur in the relations

- If any do occur, ensure that all applications that access the database update the relations in such a way as to not compromise the integrity of the database

Guideline 3

- As far as possible, avoid placing attributes in a base relation whose values may be null

- If nulls are unavoidable, make sure that they apply in exceptional cases only and that they do not apply to a majority of tuples in the relation

Decomposing a Relation

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

Mailing Address (staff Id, Department, Address)

Academics(staff Id, **Department**)

Mailing Address(Department, Address)

How should we decompose tables to remove anomalies?

The Join

- Definition: $R_1 \bowtie R_2$ is the (natural) join of the two relations
  - 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

$R_1 \bowtie R_2$

A	B	C
1	2	3
4	5	6
7	2	8
1	2	8
7	2	3

Lossless-Join Decompositions: Definition

Decomposition of R into R_1 and R_2 is a lossless-join w.r.t. a set of FDs F if, for every instance r that satisfies F:

$$R = R_1 \bowtie R_2$$

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

Example

R

A	B	C
1	2	3
4	5	6
7	2	8

Decompose

R_1

A	B
1	2
4	5
7	2

R_2

B	C
2	3
5	6
2	8

Join

R′

A	B	C
1	2	3
4	5	6
7	2	8
1	2	8
7	2	3

- The word loss in lossless refers to loss of information, not to loss of tuples. Maybe a better term here is "noisiness" or "addition of fake information".
- Last two rows are not in the original R. Would this have happened if B determined C?

Guideline 4

- Design the relation schemas so that they can be (relationally) *joined* with equality conditions on attributes that are either primary keys or foreign keys in a way that guarantees that no fake tuples are generated

Fixing Anomalies

Staff Id	Department	Address
100	Computer Science	78-101 Main Mall
104	Computer Science	78-101 Main Mall
104	Math	69-201
105	Physics	50-205

Staff Id	Department
100	Computer Science
104	Computer Science
104	Math
105	Physics

Department	Address
Computer Science	78-101 Main Mall
Math	69-201
Physics	50-205

Fixing Anomalies

Staff Id	Department
100	Computer Science
104	Computer Science
104	Math
105	Physics

Department	Address
Computer Science	78-101 Main Mall
Math	69-201
Physics	50-205

Modification anomaly fixed: Updating the mailing address only in one place does not lead to inconsistencies in the database.

Deletion anomaly fixed: Deleting the row with id 105 does not delete the mailing address of Physics.

Insertion anomaly fixed: It is now possible to store the mailing address of a department that has no faculty members.

Fixing Anomalies Beyond our Example

- We were able to fix the anomalies by splitting the Mailing address table.

- In the rest of this module, we will discuss how anomalies can be addressed formally.

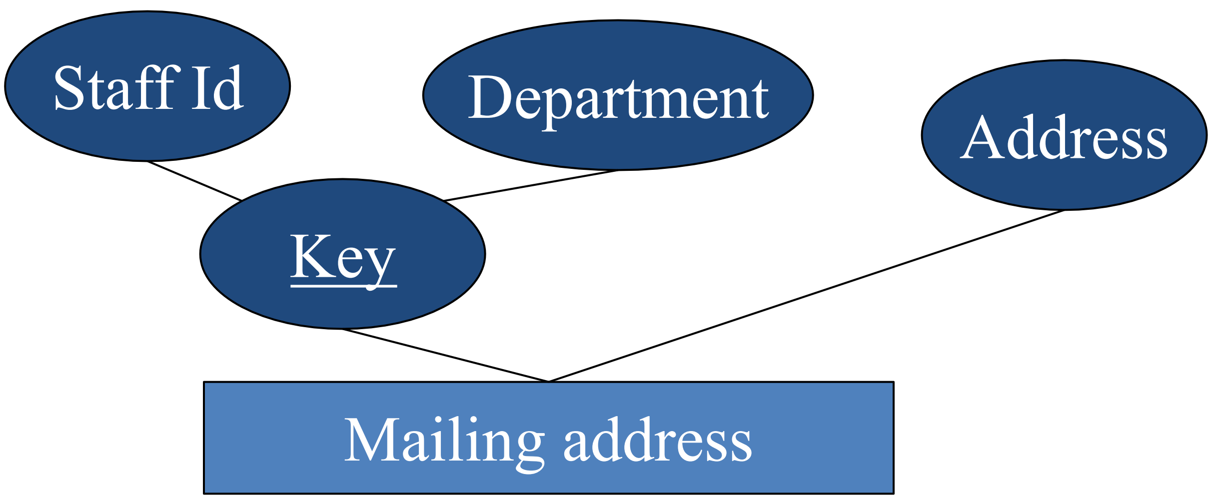
Design Guidelines

Functional Dependencies

Normalization

Functional Dependencies, Informally

- How do I know for sure if departments only have one address?
- Databases allow you to say that one attribute determines another through a functional dependency.
- So if Department determines Address, we say that there is a functional dependency from Department to Address (Dep→ Address), But Department is NOT a key (why?).



Functional Dependencies, Formally

- A **functional dependency** **X→Y** holds if for every possible legal instance, for all tuples *t1*, *t2*:

$$\text{if} \quad t1.X = t2.X \rightarrow t1.Y = t2.Y$$

- Which means given two tuples in *r*, if the X values agree, then the Y values must also agree.
- Example:

Department → Address
if t1.Department = t2.Department→ t1.Adress= t2.Adress

Identifying Functional Dependencies

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

Staff Id	Department	Address
100	Computer Science	78-101 Main Mall
104	Computer Science	78-101 Main Mall
104	Math	69-201 Main Mall
105	Physics	50-205 Main Mall

- Based on this instance alone, we cannot conclude that Department → Address.

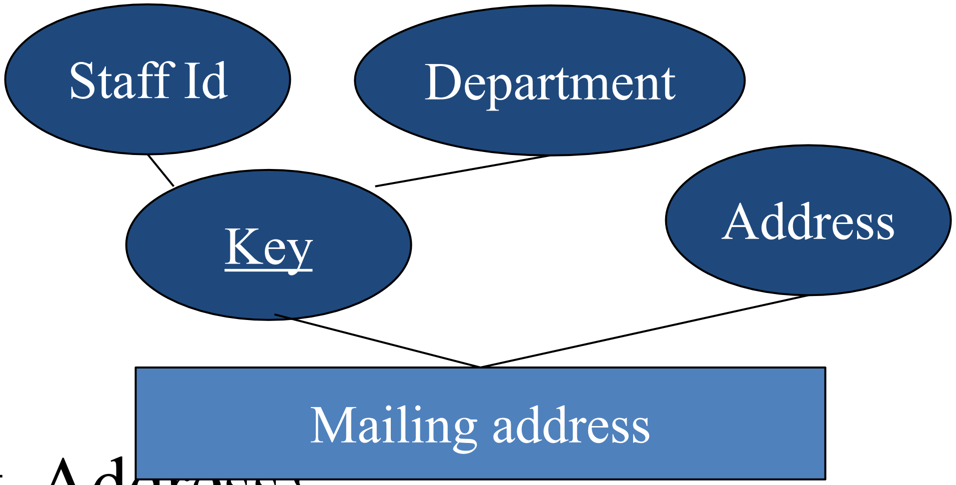
Question

- Given a table with a million rows can you tell if a functional dependency exists from data?
- Given a table can you check if a functional dependency doesn’t hold from data?
- Where do functional dependencies come from?

Keys

- As a reminder, a key is a minimal set of attributes that uniquely identify a relation
  - i.e., a key is a minimal set of attributes that *functionally determines* all the attributes
- A superkey for a relation uniquely identifies the relation, but does not have to be minimal
  - i.e.,: key $\subseteq$ superkey

  key is a subset of superkey



- Example
  - key {Id, Department}
  - superkey {Id, Department, Address}

Clicker question: Possible Keys

- Assume that the following FDs hold for a relation R(A,B,C,D):
  $B \rightarrow C$
  $C \rightarrow B$
  $D \rightarrow ABC$

Which of the following is a key for the above relation?

A. B
B. C
C. BD
D. All of the above
E. None of the above

Clicker question: Possible Keys

- Assume that the following FDs hold for a relation R(A,B,C,D):
  $B \rightarrow C$
  $C \rightarrow B$
  $D \rightarrow ABC$

Which of the following is a key for the above relation?

A. B Does not determine all

B. C Does not determine all

C. BD Not minimal

D. All of the above

E. None of the above The right answer

Clicker question: Possible Superkeys

- Assume the same relation R(A,B,C,D) and FDs
  $B \rightarrow C$
  $C \rightarrow B$
  $D \rightarrow ABC$

Which of the following is a superkey for the above relation?

A. D

B. BD

C. BCD

D. All are superkeys

E. None are superkeys

Clicker question: Possible Superkeys

- Assume the same relation R(A,B,C,D) and FDs
  B→C
  C→B
  D→ABC

Which of the following is a superkey for the above relation?

A. D

B. BD

C. BCD

D. All are superkeys

E. None are superkeys

D is a key. Therefore, all of the answers are superkeys

Explicit and Implicit FDs

- Given a set of (explicit) functional dependencies, we can determine implicit ones

studentid → *city*, *city* → *acode* implies *studentid* → *acode*

- A functional dependency fd is implied by a set F of functional dependencies if fd holds whenever all FDs in F hold.

fd= {studentid → *acode}*

F	
fd1:	studentid → city
fd2:	city → acode

- **Closure of F**: the set of all FDs implied by F.

Armstrong's Axioms

- Armstrong's Axioms (X, Y, Z are sets of attributes):
  - ***Reflexivity***: If $Y \subseteq X$, then $X \rightarrow Y$
    e.g., city,major→city Y={city} X={city, major}
  - ***Augmentation***: If $X \rightarrow Y$, then $X Z \rightarrow Y Z$ for any Z
    e.g., if sid→city, then sid,major → city,major
  - ***Transitivity***: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
    sid→*city*, *city*→*areacode* implies *sid*→*areacode*
- These are *sound (every rule is legit)* and *complete (all legit rules are produced)* inference rules.

1. studentid→acode fd1, fd2, transitivity

F

fd1: studentid→ city

fd2: city→ acode

Additional Rules

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

1. $X \rightarrow X Y$	fd1, augmentation
2. $X Y \rightarrow Z Y$	fd2, augmentation
3. $X \rightarrow Z Y$	1, 2, transitivity

F	
fd1:	$X \rightarrow Y$
fd2:	$X \rightarrow Z$

  - ***Decomposition***: If $X \rightarrow Y Z$, then $X \rightarrow Y$ and $X \rightarrow Z$
    e.g., if sid→acode,city then sid→acode, and sid→city

1 $YZ \rightarrow Y$	Reflexivity
2 $YZ \rightarrow Z$	Reflexivity
3. $X \rightarrow Y$	fd1, 1, transitivity
5. $X \rightarrow Z$	fd1, 2, transitivity

F	
fd1:	$X \rightarrow YZ$

Example: Supplier-Part DB

- Suppliers supply parts to projects.
  - SupplierPart(sname, city, state, p#, pname, qty)
    - supplier attributes: sname, city, state
    - part attributes: p#, pname
    - supplier-part attributes: qty

- Functional dependencies:
  - fd1: sname → city
  - fd2: city → state
  - fd3: p# → pname
  - fd4: sname, p# → qty

Exercise: Show that (sname, p#) is a superkey

Supplier-Part Key: Part 1:

Exercise: Show that (sname, p#) is a superkey of SupplierPart(sname,city,state,p#,pname,qty)

Proof has two parts:

fd1: sname → city
fd2: city → state
fd3: p# → pname
fd4: sname, p# → qty

a. Show: sname, p# is a (super)key

1.	sname, p# → sname, p#	reflex
2.	sname → city	fd1
3.	sname → state	2, fd2, trans
4.	sname, p# → city, p#	2, augm
5.	sname, p# → state, p#	3, augm
6.	sname, p# → sname, p#, state	1, 5, union
7.	sname, p# → sname, p#, state, city	4, 6, union
8.	sname, p# → sname, p#, state, city, qty	7, fd4, union
9.	sname, p# → sname, p#, state, city, qty, pname	8, fd3, union

Example: Supplier-Part DB

- Suppliers supply parts to projects.
  - SupplierPart(sname, city, state, p#, pname, qty)
    - supplier attributes: sname, city, state
    - part attributes: p#, pname
    - supplier-part attributes: qty

- Functional dependencies:
  - fd1: sname → city
  - fd2: city → state
  - fd3: p# → pname
  - fd4: sname, p# → qty

Exercise: Show that (sname, p#) is a key

Supplier-Part Key: Part 2

b. Show: (sname, p#) is a *minimal* key of
SupplierPart(sname,city,state, p#,pname,qty)

1. p# does not appear on the RHS of any FD therefore except for p# itself, nothing else determines p#
2. specifically, sname → p# does not hold
3. therefore, sname is not a key
4. similarly, p# is not a key

fd1: sname → city
fd2: city → state
fd3: p# → pname
fd4: sname, p# → qty

sname,p# →sname, p#, city, state, pname, qty

sname → sname, city, state

p# → p#, pname

Computing the Closure of Attributes

- Closure for <u>a set of attributes</u> X is denoted by X^+

> 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
>   Then add C to Closure

- Example
  - $Studentid^+ = \{Studentid\}$
  - $Studentid^+ = \{Studentid, city\}$
  - $Studentid^+ = \{Studentid, city, acode\}$

F	
fd1:	studentid → city
fd2:	city → acode

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

Supplier-Part Key: Closure of Attributes

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
      Then add C to Closure

fd1: sname → city
fd2: city → status
fd3: p# → pname
fd4: sname, p# → qty

Ex: SupplierPart(sname,city,status,p#,pname,qty)

$\{sname, p\#\}^+ =$ sname,p#, city, status, pname, qty

$\{sname\}^+ =$ sname, city, status

$\{p\#\}^+ =$ p#, pname

So seeing if a set of attributes is a superkey means checking to see if it's closure is all the attributes – pretty simple

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: If 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
- We were able to fix the anomalies by splitting (decomposing) the Mailing address table, but how should we do this more formally? And how is it related to functional dependencies?

Design Guidelines

Functional Dependencies

Normalization

Normalization

- Normalization is a process that aims at achieving better designed relational database schemas using
  - Functional Dependencies
  - Primary Keys

- The normalization process takes a relational schema through a series of tests to certify whether it satisfies certain conditions
  - The schemas that satisfy certain conditions are said to be in a given *'Normal Form'*.
  - Unsatisfactory schemas are decomposed by breaking up their attributes into smaller relations that possess desirable properties (e.g., no anomalies)

Review of "Key" Concepts

- **Superkey** - A set of attributes such that no two tuples have the same values for these attributes
- **Key** - A minimal Superkey, called a Candidate Key if more than one:
  - **Primary key** - A selected candidate key
  - **Secondary key** - Remaining candidate keys
- **Prime Attribute** - An attribute that is a member of any candidate key
- **Non-prime attribute** - An attribute that is not a member of any candidate key

First Normal Form (1NF)

- A relation schema is in 1NF if domains of attributes include only atomic (simple, indivisible) values and the value of an attribute is a single value from the domain of that attribute
- 1NF disallows
  - having a set of values, a tuple of values, or a combination of both as an attribute value for a single tuple
  - “relations within relations” and “relations as attributes of tuples

Customer	Order	Items
Jones	123	Basket, Football
Gupta	876	Hat, Glass, Pencil

A non-1nf relation

Relations in 1NF still have problems

Customer	Order	Item
Jones	123	Basket
Jones	123	Football
Gupta	876	Hat
Gupta	876	Glass
Gupta	876	Pencil

Problem with this design:
Redundancy (Jones, 123)

Customer	Order	Item1	Item2	Item3	Item4
Jones	123	Basket	Football	Null	Null
Gupta	876	Hat	Glass	Pencil	Null

Problem with this design:

Too many Null values
What if an order has >max (4) items

Second Normal Form (2NF)

- A relation is in 2NF, if for every FD $X \rightarrow Y$ where X is a minimal key and Y is a non-prime attribute, then no proper subset of X determines Y.

Not part of any key

- e.g., the address relation is not in 2NF:
  - House#, street, postal_code is a minimal key
  - House#, street, postal_code → Province
  - Postal_code → province

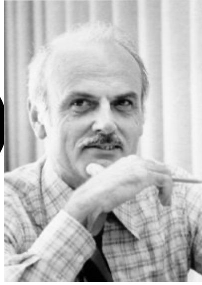
X = House#, street, postal code
Y = province

Redundancy in 2NF

- In 2NF, a member of key shouldn't determine a non-prime member.

- But still, a non-prime attribute can determine another attribute, which results in redundancy.

Boyce-Codd Normal Form (BCNF)



Raymond Boyce

Ted Codd

A relation R is in BCNF if for all non-trivial dependencies in R:

If X → b then X is a superkey for R

- A dependency is **trivial** if the LHS contains the RHS, e.g., City, Province→ City is a trivial dependency (A, B → A)
- Informally: Whenever a set of attributes of R determine another attribute, it should determine all the attributes of R.

BCNF Example

- Address(<u>House#, Street, </u>City, Province, <u>PostalCode</u>)

House#, Street, PostalCode → City

House#, Street, PostalCode → Province

PostalCode → City

PostalCode → Province

Is Address in BCNF?

$\{PostalCode\}^{+} = \{PostalCode, City, Province\}$

No. PostalCode is not a superkey

PostalCode → City is violating BCNF

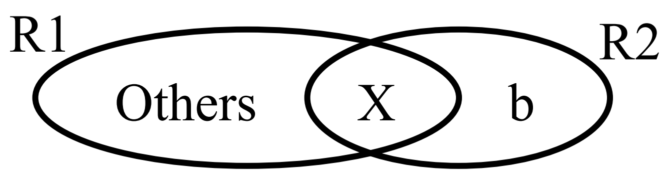
Decomposing into BCNF Losslessly

1. Add implicit FDs to your list of FDs

e.g. If FDs contain A→B and B→C, add A→C to list of FDs

2. Pick any f ∈ FD that violates BCNF of the form X→b

3. Decompose R into two relations: $R_1(\text{All} \setminus b)$ & $R_2(X \cup b)$



4. Repeat on R_1 and R_2 using FD

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

How do we decompose into BCNF losslessly?

- Ultimately, all relations with two attributes 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

BCNF Definition: A relation R is in BCNF if for all non-trivial dependencies in R: If X → b then X is a superkey for R

BCNF Decomposition Example

- Relation: R(ABCD)
- Closure and keys

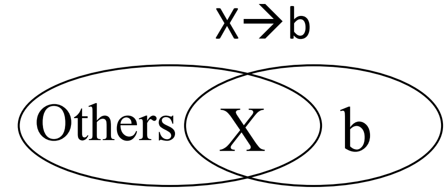
$A^+ = A$ $\qquad$ $B^+ = BC$

$C^+ = C$ $\qquad$ $D^+ = AD$

BD is the only key as $BD^+ = BDCA$

fd1: $B \rightarrow C$

fd2: $D \rightarrow A$



Considering B→C, is B a superkey in R?

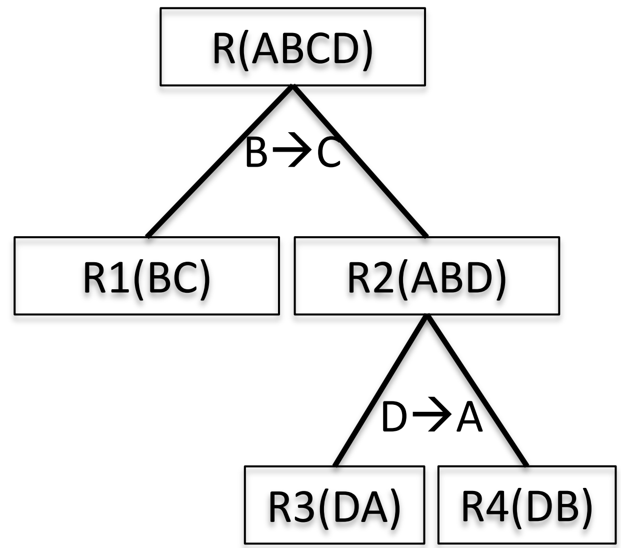
No. Decompose

R1(B,C), R2(A,B,D)

Considering D→A, is D a superkey for R2?

No. Decompose

R3(D,A), R4(D,B)



Final answer: R1(B,C), R3(D,A), R4(D,B).
What does this mean?

Test if a given FD applies

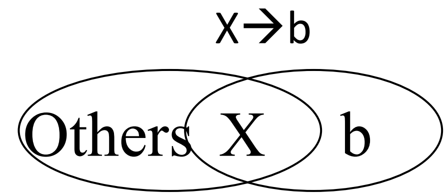
For an FD $X \rightarrow b$, if the decomposed relation S contains $X \cup b$, and $b \in X^+$ then the FD holds for S.

- 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 FDs 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.

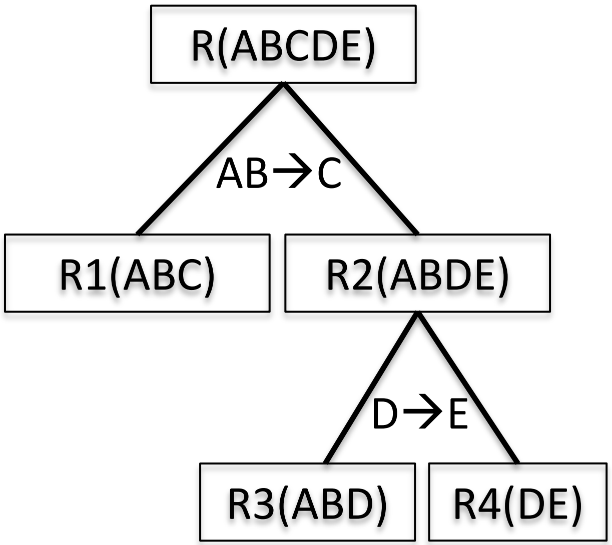
- Does $CD \rightarrow E$ hold?
  - No (Why? Because S doesn't contain E)

Another BCNF Example

fd1: $AB \rightarrow C$
fd2: $D \rightarrow E$



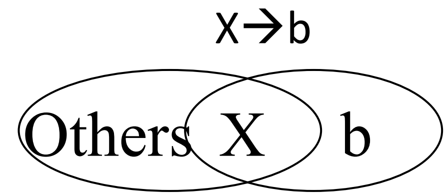
- R(ABCDE)
- Find closure of the following
  - $AB^{+} = ABC$
  - $D^{+} = DE$
- $AB \rightarrow C$ violates BCNF in R
  - R1(ABC), R2(ABDE)
- $D \rightarrow E$ violates BCNF in R2
  - R3(ABD), R4(DE)



Final answer: R1(ABC), R3(ABD), R4(DE)

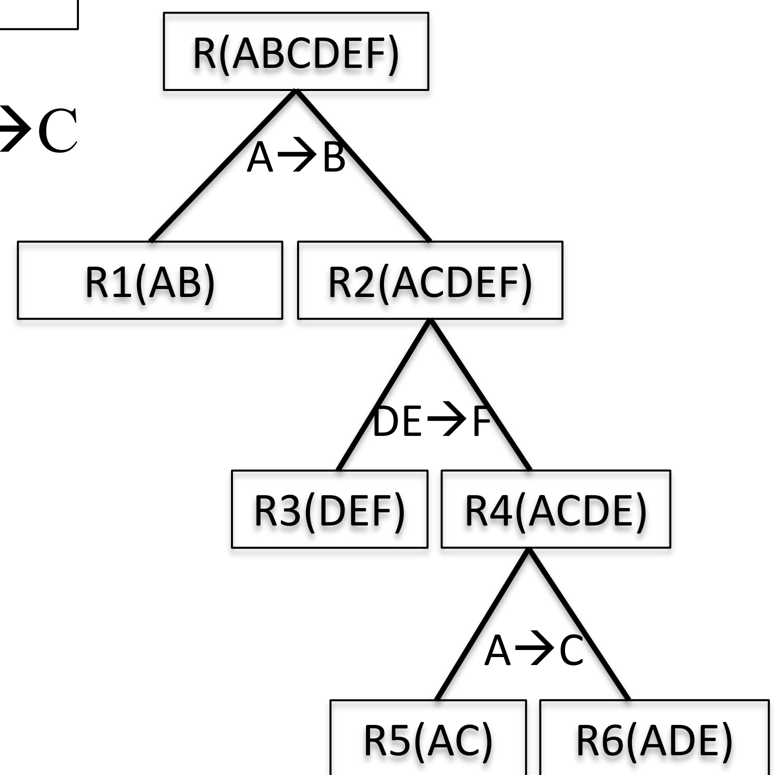
Example: Implicit FDs matter

fd1	A→B
fd2	DE→ F
fd3	B→C



- R(A,B,C,D,E,F)
  - $A^+ = ABC$
  - $B^+ = BC$
  - $DE^+ = DEF$
  - A^+ contains C so an implicit FD A→C holds. We add fd4 as A→C

- A→B is violating BCNF in R
  - R1(AB), R2(ACDEF)
  - R1 is BCNF, but R2 is not in BCNF
- DE→ F is violating BCNF in R2
  - R3(DEF), R4(ACDE)
  - R3 is in BCNF, is R4 in BCNF?
- A→C is violating BCNF in R4
  - R5(AC), R6(ADE)



Final answer R1 (AB), R3(DEF), R5(AC), R6(ADE)

BCNF is great, and…

- Guaranteed that there will be no redundancy of data
- Easy to understand (just look for superkeys)
- Easy to do.
- So what is the main problem with BCNF?
  - For one thing, BCNF may not preserve all dependencies

Dependency Preservation

A functional dependency X→Y is preserved in a relation R, if R contains all of the attributes of X and Y.

Example:

- R(A,B,C) fd1 A→B , fd2 A→D

fd1 is preserved in R and fd2 is not preserved in R

An Illustrative BCNF example

Unit	Company	Product

Unit $\rightarrow$ Company
Company, Product $\rightarrow$ Unit

Is unit a key?

Unit	Company

Unit $\rightarrow$ Company

Unit	Product

The second functional dependency is no longer preserved.

So What's the Problem?

Unit	Company
SKYWill	ABC
Team Meat	ABC

Unit	Product
SKYWill	Databases
Team Meat	Databases

Unit $\rightarrow$ Company

No problem so far. All *local* FD's are satisfied.
Let's join the relations back into a single table again:

Unit	Company	Product
SKYWill	ABC	Databases
Team Meat	ABC	Databases

Violates the FD: Company, Product $\rightarrow$ Unit

Denormalization

- Process of intentionally violating a normal form to gain performance improvements
- Performance improvements:
    1. Fewer joins
    2. Reduces number of foreign keys

- Useful in data analysis or if certain queries often require (joined) results, and the queries are frequent enough.