CPSC 304
Introduction to Database Systems

Schema Refinement and Normal Forms

Textbook Reference
Database Management Systems
3rd edition19.1-19.6 (except 19.5.2)
2nd edition: 15.1-15.7

Hassan Khosravi
Borrowing many slides from Rachel Pottinger

Databases – the continuing saga

- We've learned that databases are wonderful things
- We've learned how to create a *conceptual design* using ER diagrams
- We've learned how to create a *logical design* by turning the ER diagrams into a relational schema including minimizing the data and relations created
- Now we need to *refine* that schema to reduce duplication of information

Learning Goals

- Debate the pros and cons of redundancy in a database.
- Provide examples of update, insertion, and deletion anomalies.
- Given a set of tables and a set of functional dependencies over them, determine all the keys for the tables.
- Show that a table is/isn't in 3NF or BCNF.
- Prove/disprove that a given table decomposition is a lossless join decomposition. Justify why lossless join decompositions are preferred decompositions.
- Decompose a table into a set of tables that are in 3NF, or BCNF.

Imagine that we've created a perfectly good entity for mailing addresses at UBC:

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

What would an instance look like?

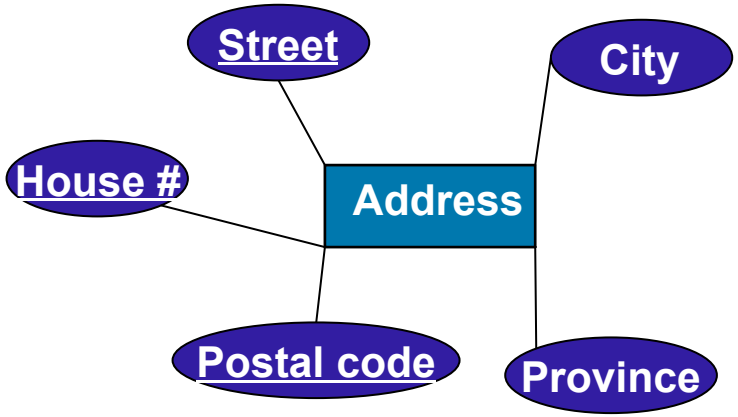
Name	Department	Mailing Location
Hassan Khosravi	Computer Science	201-2366 Main Mall
Steve Wolfman	Computer Science	201-2366 Main Mall
Rachel Pottinger	Computer Science	201-2366 Main Mall
Ed Knorr	Computer Science	201-2366 Main Mall
Joel Friedman	Computer Science	201-2366 Main Mall
Joel Friedman	Math	121-1984 Mathematics Rd
Brian Marcus	Math	121-1984 Mathematics Rd

Is this a good design?
Think about insert, deletion, and update examples that may be problematic

Okay, that's bad. But how do I know for sure if departments have only one address?

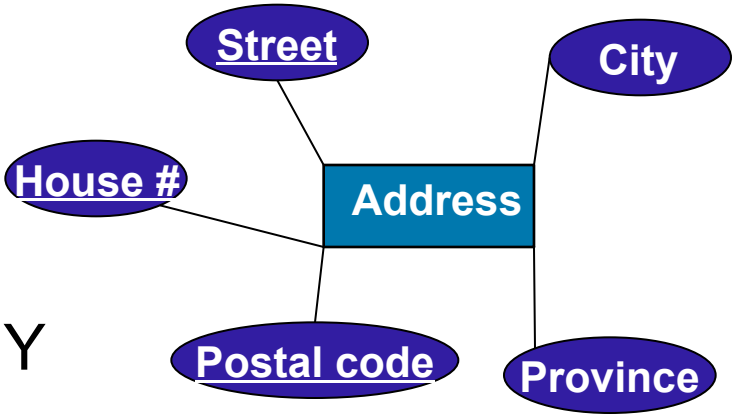
- Databases allow you to say that one attribute determines another through a *functional dependency*.
- So if Department determines Address but not Name, we say that there's a functional dependency from Department to Address. But Department is NOT a key.

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



Functional Dependencies (FDs) – technically speaking

- 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 if:
  for each possible t1, t2, if t1.PostalCode = t2.PostalCode then
  t1.{City,Province} = t2.{City,Province})
- 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

Let's see some more instances

House #	Street	City	Province	Postal Code
101	Main Street	Vancouver	BC	V6A 2S5
103	Main Street	Vancouver	BC	V6A 2S5
101	Cambie Street	Vancouver	BC	V6B 4R3
103	Cambie Street	Vancouver	BC	V6B 4R3
101	Main Street	Delta	BC	V4C 2N1
103	Main Street	Delta	BC	V4C 2N1

Does City determine Province?

Not necessarily. It does in this instance, but that doesn't mean anything. Similar to integrities!

Let's see some more instances

House #	Street	City	Province	Postal Code
101	Main Street	Vancouver	BC	V6A 2S5
103	Main Street	Vancouver	BC	V6A 2S5
101	Cambie Street	Vancouver	BC	V6B 4R3
103	Cambie Street	Vancouver	BC	V6B 4R3
101	Main Street	Delta	BC	V4C 2N1
103	Main Street	Delta	BC	V4C 2N1
800	Benvenuto Ave	Victoria	BC	V8M 1J8
165	Duckworth Street	Victoria	NF	A1B 4R5

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!
- 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

Clicker question: Possible FDs

- Consider the relation R with the following instance:

A	B	C	D
1	2	3	4
2	3	4	6
6	7	8	9
1	3	4	5

What FDs cannot be true given the instance above?

A. B→ C
B. B→ D
C. D→B
D. All of the above can be true
E. None of the above can be true

Clicker question: Possible FDs

- Consider the relation R with the following instance:

A	B	C	D
1	2	3	4
2	3	4	6
6	7	8	9
1	3	4	5

What FDs cannot be true given the instance above?

A. B→ C fine

B. B→ D not possible (2nd and 4th), so correct answer

C. D→B fine

D. All of the above can be true

E. None of the above can be true

Naming the Evils of Redundancy

- Let's consider Postal Code → City, Province

House #	Street	City	Province	Postal Code
101	Main Street	Vancouver	BC	V6A 2S5
103	Main Street	Vancouver	BC	V6A2S5
101	Cambie Street	Vancouver	BC	V6B 4R3
103	Cambie Street	Vancouver	BC	V6B 4R3
101	Main Street	Delta	BC	V4C 2N1
103	Main Street	Delta	BC	V4C 2N1

- Update anomaly: Can we change Delta's province?
- Insertion anomaly: What if we want to insert that V6T 1Z4 is in Vancouver?
- Deletion anomaly: If we delete all addresses with V6A 2S5, we lose that V6A 2S5 is in Vancouver!

Can we do better?

Once more, with feeling

House #	Street	Postal Code
101	Main Street	V6A 2S5
103	Main Street	V6A2S5
101	Cambie Street	V6B 4R3
103	Cambie Street	V6B 4R3
101	Main Street	V4C 2N1
103	Main Street	V4C 2N1

City	Province	Postal Code
Vancouver	BC	V6A 2S5
Vancouver	BC	V6B 4R3
Delta	BC	V4C 2N1

- Did we lose anything?
- Are our problems fixed?

Okay, that worked pretty well.
But what can we do to help us think about FDs in general?

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)

The Key's the thing

- 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
  - e.g., House#, Street, PostalCode is a key
- A superkey for a relation uniquely identifies the relation, but does not have to be minimal
  - i.e.,: key $\subseteq$ superkey
  - E.g.,:
    - House#, Street, PostalCode is a key and a superkey
    - House#, Street, PostalCode, city is a superkey, but not a key

Clicker question: Possible Keys

- Assume that the following FDs hold for a relation R(A,B,C,D):
  B→C
  C→B
  D→A,B,C

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

A. B
B. C
C. B,D
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→C
  C→B
  D→A,B,C

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

A. B — Does not determine all
B. C — Does not determine all
C. B,D — 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→C
  C→B
  D→A,B,C

Which of the following are not a superkey for the above relation?

A. D
B. B,D
C. B,C,D
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→A,B,C

Which of the following are not a superkey for the above relation?

A. D
B. B,D
C. B,C,D
D. All are superkeys
E. None are superkeys

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

Alan Keyes

Alicia Keys

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(House #, Street, City, Province, Postal Code)
- A superkey is a set of attributes determining the rest of the attributes in the relation, but does NOT have to be minimal (e.g., the key above, or adding in either of City and Province)
- Given a set of (explicit) functional dependencies, we can determine others…

Evelyn Keyes

Francis Scott Key

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 $X \rightarrow Y$
    e.g., city,major→city
  - *Augmentation*: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ 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* 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
- Example: Derive union rule from axioms (Reflexivity, Augmentation, and Transitivity)

1. X→Y	given
2. X→Z	given
3. X→X Y	1, augmentation
4. X Y→Z Y	2, augmentation
5. X→Z Y	3, 4, transitivity

1. X→YZ	given
2 YZ →Y	Reflexivity
3 YZ → Z	Reflexivity
4. X→Y	1, 2, transitivity
5. X→Z	1, 3, transitivity

All 5 Rules

- ***Reflexivity***: If $Y \subseteq X$, then $X \rightarrow Y$
  e.g., city,major→city
- ***Augmentation***: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ 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*
- ***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

Example: Supplier-Part DB

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

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

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

Supplier-Part Key: Part 1: Determining all attributes

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

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

Proof has two parts:

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

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

Supplier-Part Key: Part 2: Minimalness

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

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

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

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

sname → sname, city, status

p# → p#, pname

Do you, by any chance, have anything less painful?

- Scared you're going to mess up? *Closure* is a fool-proof method of checking FDs.
- Closure for a set of attributes X is denoted X^+
- 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
  Then add C to Closure

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

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

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

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

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

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

Clicker Exercise: Finding Keys

- Which of the following is a minimal key of the Relation R(ABCDE) with FD's:
  $D \rightarrow C$
  $CE \rightarrow A$,
  $D \rightarrow A$
  $AE \rightarrow D$.

A. ABDE
B. BCE
C. CDE
D. All of these are keys
E. None of these are keys

Clicker Exercise: Finding Keys

- Which of the following is a minimal key of the Relation R(ABCDE) with FD's:
  D → C
  CE → A,
  D → A
  AE → D.

A. ABDE — Superkey, since D→A
B. BCE — Key
C. CDE — CDE+=CDEA
D. All of these are keys
E. None of these are keys

Exercise: Finding all Keys

- Find all the minimal keys of the Relation R(ABCDE) with FD's:
  D → C
  CE → A,
  D → A
  AE → D.

Keys = ABE, BCE, BDE

Popping back up to our original question…

- Is this really a good design for a table?

Name	Department	Mailing Location
Rachel Pottinger	Computer Science	201-2366 Main Mall
Steve Wolfman	Computer Science	201-2366 Main Mall
Don Acton	Computer Science	201-2366 Main Mall
Ed Knorr	Computer Science	201-2366 Main Mall
Joel Friedman	Computer Science	201-2366 Main Mall
Joel Friedman	Math	121-1984 Mathematics Rd
Brian Marcus	Math	121-1984 Mathematics Rd

- Wouldn't it be nice if there was some rule that said if the amount of redundancy that we had was good?

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.

Remember those anomalies?

1NF

- Each attribute in a tuple has only one value
  - E.g., for “postal code” you can’t have both V6T 1Z4 and V6S 1W6
- Why do we need it? Codd’s original vision of the relational model allowed multi-valued attributes

Partial FDs and 2NF

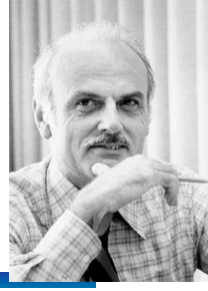
- Partial FDs:
  - A FD, A→ B is a partial FD, if some attribute of A can be removed and the FD still holds.
  - Formally, there is some proper subset of A, C ⊂ A, such that C → B
- Let us call attributes which are part of some candidate key, key attributes, and the rest non-key attributes.
- Second normal form:
  - A relation is in second normal form (2NF) if it is in 1NF and no non-key attribute is partially dependent on a candidate key.
- In other words, no C → B where C is a strict subset of a candidate key and B is a non-key attribute.

2NF with example

- No partial key dependency
- A relation is in 2NF, if for every FD X→Y where X is a minimal key and Y is a non-key attribute, then no proper subset of X determines Y
- 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

Raymond Boyce & Ted Codd



Boyce-Codd Normal Form (BCNF)

A relation R is in BCNF if:

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

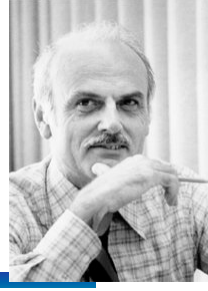
(Must be true for every such dependency)

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*.

Boyce-Codd Normal Form (BCNF)



Ex: Address(House#, Street, City, Province, PostalCode)

so

- House#, Street, PostalCode → City
- House#, Street, PostalCode → Province
- PostalCode → City
- PostalCode → Province

Is it in BCNF? Why or why not?

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

No. Postal Code is not a superkey

What do we want?
Guaranteed freedom from redundancy!

How do we get there?

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

Decomposing a Relation

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

- 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.
- Intuitively, decomposing R means storing instances of the relations produced by the decomposition, instead of instances of R.
- E.g., Address(House#,Street,City,Province,Postal Code)
  - Address(House#,Street#,PostalCode),
  - PC(City, Province, PostalCode)

How can we decompose correctly?

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

$R_1 \bowtie R_2$

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

Lossless-Join Decompositions: Definition

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

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

- If we JOIN the X-part of r with the Y-part of r the result is exactly r
- It is always true that r is a subset of the JOIN of its X-part and Y-part
- In general, the other direction does not hold! If it does, the decomposition is a lossless-join.

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".

Example Lossy-Join Decomposition

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

decompose

A	B
1	2
4	5
7	2

B	C
2	3
5	6
2	8

join

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

Last two rows are not in the original. Would this have happened if B→C?

All decompositions used to resolve redundancy *must* be lossless!

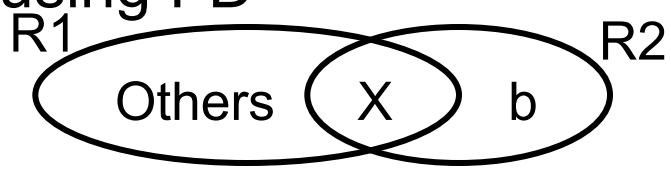
How do we decompose into BCNF losslessly?

- 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

1. Pick any f ∈ FD that violates BCNF of the form X→b
2. Decompose R into two relations: R_1(A-b) & R_2(X ∪ b)

- Recurse on R_1 and R_2 using FD

Pictorally:



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

BCNF Example (Let's do the first one together)

Remember BCNF def: For all non-trivial functional dependencies X→b, X must be a superkey for a relation to be in BCNF

Relation: R(ABCD) FD: B→C, D→A

Keys?

$A^+ = A$

$B^+ = BC$

$C^+ = C$

$D^+ = AD$

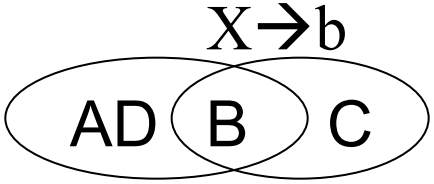
$BD^+ = BDCA$

BD is the only key

Look at FD B→ C. Is B a superkey?

No. Decompose

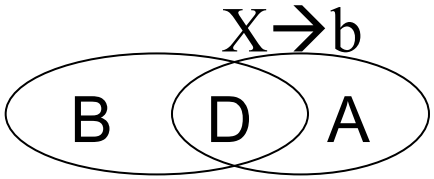
R1(B,C), R2(A,B,D)



Look at FD 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 answer mean?

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

- Take the closure of the attributes using *all* FDs
- For an FD X→b, if the decomposed relation S contains {X U 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 → C, BC → D, CD → E, DE → A, and AE → B. Project these FD's onto the relation S(A,B,C,D).
- Does AB→D hold?
  - First check if A, B and D are all in S? They are
  - Find AB^+= ABCDE
  - Then yes AB→ D does hold in S.
- Does CD→E hold? No

After you decompose, how do you know which FDs apply? The clicker version

- Consider relation R(A,B,C,D,E) with functional dependencies AB → C, BC → D, CD → E, DE → A, and AE → B. Project these FD's onto the relation S(A,B,C,D).
- Which of the following hold in S?

A. A→B
B. AB→E
C. AE→B
D. BCD→A
E. None of the above

After you decompose, how do you know which FDs apply? The clicker version

- Consider relation R(A,B,C,D,E) with functional dependencies AB → C, BC → D, CD → E, DE → A, and AE → B. Project these FD's onto the relation S(A,B,C,D).
- Which of the following hold in S?

A. A→B — A^+=A, so A→ B does not hold

B. AB→E — E is not in S, so AB→ E does not hold

C. AE→B — E is not in S, so AE→ B does not hold

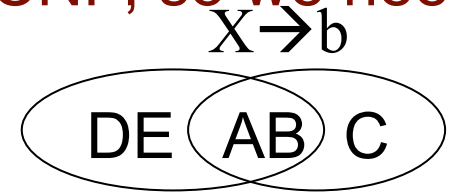
D. BCD→A — Yes. BCD+=ABCDE; all in S

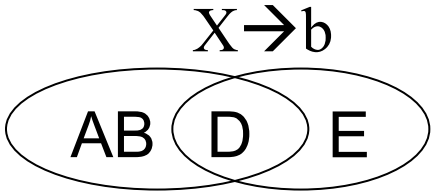
E. None of the above

Note that we use all FDs for finding closures, so for D we use DC→E Even though E is not present in S.

Another BCNF Example

- R(ABCDE)
- FD: AB→C, D→E
- Find closure of the following
  - AB^+ = ABC
  - D^+ = DE
- So the relation is not in BCNF, so we need to decompose
  - R_1(ABC), R_2(ABDE)
    - AB is a key for R_1 so it is in BCNF
    - D is not a key for R_2 so not in BCNF
  - R_3(ABD), R_4(DE)
- Final answer, R_1, R_3, R_4





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.

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

FD =

A→B

DE→ F,
B→C

Is it in BCNF? If so, why. If not, decompose into BCNF

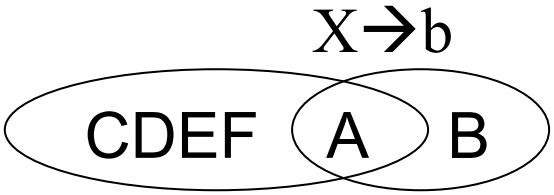
Yet Another BCNF Example: Do implicit FDs matter?

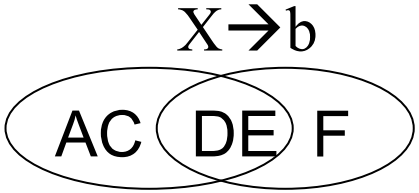
R(A,B,C,D,E,F)

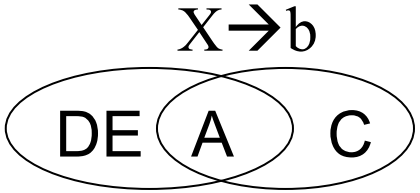
FD =
A→B
DE→ F,
B→C

- A^+ = ABC
- B^+ = BC
- DE^+ = DEF

- A→B is violating BCNF in R, so decompose
  - R_1(AB), R2(ACDEF)
  - R_1 is BCNF, but R2 is not in BCNF
- DE→ F is violating BCNF in R_2, so decompose
  - R_3(DEF), R_4(ACDE)
  - R_3 is in BCNF, is R_4 in BCNF?
- A^+ contains C so an implicit FD A→C holds in R4 which is violating BCNF
  - R5(AC), R6(ADE)
- Final answer R_1, R_3, R_5, R_6







Clicker exercise: More BCNF

- Let R(ABCD) be a relation with functional dependencies $A \rightarrow B$, $C \rightarrow D$, $AD \rightarrow C$, $BC \rightarrow A$
- Decompose into BCNF.

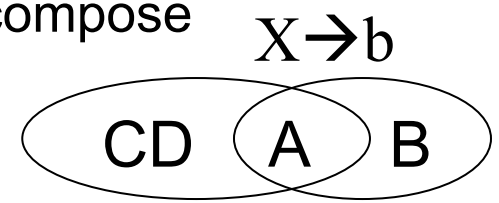
Clicker exercise: More BCNF

- Let R(ABCD) be a relation with functional dependencies $A \rightarrow B$, $C \rightarrow D$, $AD \rightarrow C$, $BC \rightarrow A$
- Decompose into BCNF. Which of the following is a lossless-join decomposition of R into Boyce-Codd Normal Form (BCNF)?

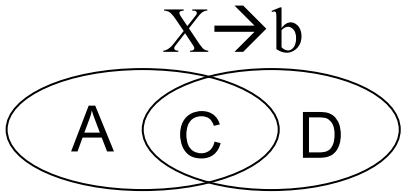
A. {AB, AC, BD}
B. {AB, AC, CD}
C. {AB, AC, BCD}
D. All are
E. None are

Clicker exercise: More BCNF

- Let R(ABCD) be a relation with functional dependencies $A \rightarrow B$, $C \rightarrow D$, $AD \rightarrow C$, $BC \rightarrow A$
- Decompose into BCNF.
  - $A^+ = AB$
  - $C^+ = CD$
  - $AD^+ = ADBC$
  - $BC^+ = BCAD$
- $A \rightarrow B$ is violating BCNF, so we need to decompose
  - $R_1(AB)$, $R_2(ACD)$



- $C \rightarrow D$ is violating BCNF so we decompose
  - R3(AC), R4(CD)



- Final answer $R_1(AB)$, R3(AC), R4(CD)

Clicker exercise: More BCNF

- Let R(ABCD) be a relation with functional dependencies $A \rightarrow B$, $C \rightarrow D$, $AD \rightarrow C$, $BC \rightarrow A$
- Decompose into BCNF. Which of the following is a lossless-join decomposition of R into Boyce-Codd Normal Form (BCNF)?

A. {AB, AC, BD} Let's see what happens if you randomly decompose
B. {AB, AC, CD} $R_1(\underline{A}B)$, R3($\underline{AC}$), R4($\underline{C}D$)
C. {AB, AC, BCD} C^+=CD, so C is not key for (BCD)
D. All are
E. None are

Clicker Exercise: Option 'A' exposed

- Let R(ABCD) be a relation with functional dependencies $A \rightarrow B$, $C \rightarrow D$, $AD \rightarrow C$, $BC \rightarrow A$

A. Is {AB, AC, BD} a lossless join?

Imagine tuples:

A	B	C	D
1	2	5	6
1	2	3	7
8	2	9	4

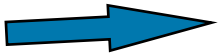
decompose

A	B
1	2
8	2

A	C
1	5
1	3
8	9

B	D
2	6
2	7
2	4

join



A	B	C	D
1	2	5	6
1	2	3	7
8	2	9	4
1	2	3	4

This BCNF stuff is great and easy!

- 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

An illustrative BCNF example

Unit	Company	Product

Unit → Company
Company, Product → Unit

Is unit a key?

Unit	Company

Unit → Company

Unit	Product

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

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
or b is part of a minimal key.

} BCNF

(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)

Example: R(Unit,Company, Product)

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

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

Minimal Cover for a Set of FDs

Goal: Transform FDs to be as small as possible

- *Minimal cover* G for a set of FDs F:
  - Closure of F = closure of G (i.e., imply the same FDs)
  - Right hand side of each FD in G is a single attribute
  - If we delete an FD in G or delete attributes from an FD in G, the closure changes
- Intuitively, every FD in G is needed, and is "*as small as possible*" in order to get the same closure as F
- e.g., A→B, ABCD→E, EF→GH, ACDF→EG has the following minimal cover:
  - A→B, ACD→E, EF→G and EF→H

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→B, ABCD→E, EF→G, EF→H, ACDF → EG

- Replace last rule with
  - 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

Example:

A→B, ABCD→E, EF→G, EF→H, ACDF → E, ACDF → G

- Can we take anything away from the LHS?
  - $ABCD^{+}$= ABCDE
  - ACD^{+}= ABCDE, so remove B from the FD

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→B, ACD→E, EF→G, EF→H, ACDF → E, ACDF → G

- Let's find $ACDF^+$ without considering the highlighted FDs
  - $ACDF^+$= ACDFEBGH, so I can remove the highlighted rules

- Final answer: A→B, ACD→E, EF→G, EF→H

Another minimal cover example

- Consider the relation R(CSJDPQV) with FDs C→SJDPQV, JP→C, SD→P, J→S

Find a minimal cover

Another minimal cover example

- Consider the relation R(CSJDPQV) with FDs C→SJDPQV, JP→C, SD→P, J→S

Find a minimal cover

- C→S, C→J, C→D, C→P, C→Q, C→V, JP→C, SD→P, J→S

Can we shorten any FDs?

Let's consider shortening JP→C

JP^+ = CSJDPQV

J^+ = JS

P^+ = P

Let's consider shortening SD→P

SD^+ = SDP

S^+ = S and D^+ = D

Another minimal cover example

- Consider the relation R(CSJDPQV) with FDs C→SJDPQV, JP→C, SD→P, J→S

Find a minimal cover

- C→S, C→J, C→D, C→P, C→Q, C→V, JP→C, SD→P, J→S

  Can we shorten any FDs? No

  Can we remove any FDs?

  - Let's consider C→S and find C^+ without considering this rule
    - C^+ = SJDPQV, so we can delete this FD
  - Let's consider C→P and find C^+ without considering this rule
    - C^+ = SJDQVP, so we can delete this FD

Make sure to minimize LHS before deleting redundant FDs

- If step 3 is done prior to step 2, the final set of FDs could still contain redundant FDs

1. One attribute on RHS)
2. Delete Redundant FDs
3. Minimize LHS of each FD

Does not work

- ABCD→E, E→D, A→B, AC→D
  - Let's delete redundant FDs.
  - None of the FDs are redundant.
- ABCD→E, E→D, A→B, AC→D
  - Now let's shorten FDs
  - ABCD→E can be replaced by AC→E
- However the current set of FDs are not minimal
  - AC→E, E→D, A→B, AC→D
  - The highlighted FD can be deleted

Decomposition into 3NF using Minimal Cover

- Decomposition into 3NF:
  - Given the FDs F, compute F': the *minimal cover for F*
  - Decompose using F' if violating 3NF similar to how it was done for BCNF
  - After each decomposition identify the set of dependencies N in F' that are not preserved by the decomposition.
  - For each $X \rightarrow a$ in N create a relation $R_n(X \cup a)$ and add it to the decomposition

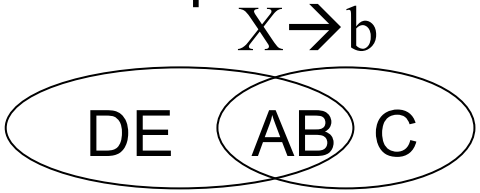
3NF example

- Example: R(ABCDE) FD: AB→C, C→D
  - Cover already minimal so find minimal keys

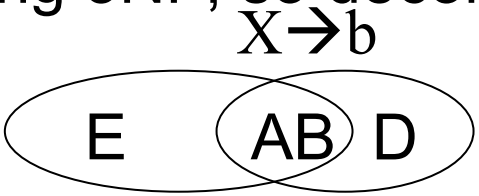
$ABE^+ = ABCDE$ only key

$AB^+ = ABCD$
$C^+ = CD$

  - AB→C is violating 3NF, so decompose
    - $R_1(ABC)$, $R_2(ABDE)$



  - AB→D (transitivity) is violating 3NF, so decompose
    - R3(ABD), R4(ABE)



  - $R_1(ABC)$, R3(ABD), R4(ABE) are now in 3NF
  - Are all FDs preserved? We lost C→D so add $R_5(CD)$

Another 3NF example

- Let R(CSJDPQV) be a relation with the following FDs
  - SD→P
  - JP→C
  - J→S
- Decompose R into 3NF

Another 3NF example

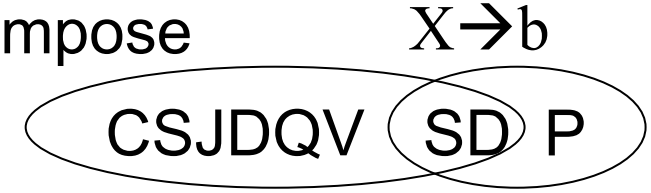
- Let R(CSJDPQV) be a relation with the following FDs
  - SD→P
  - JP→C
  - J→S
- Decompose R into 3NF
  - Already in minimal cover
  - SD^+=SDP
  - JP^+= JPSC
  - J^+=JS
  - $JDQV^+$ = CSJDPQV Key

Another 3NF example

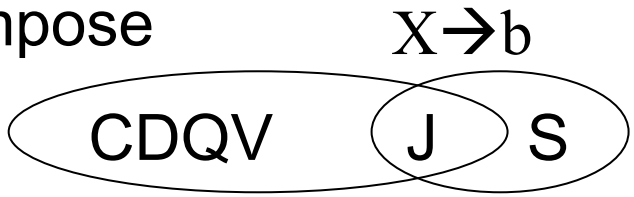
- Let R(CSJDPQV) be a relation with the following FDs
  - SD→P
  - JP→C
  - J→S

$JP^+ = JPSC$
$SD^+ = SDP$
$J^+ = JS$
$JDQV^+ = CSJDPQV$ Key

- SD→P violates 3NF in R, so decompose
  - $R_1(SDP)$, $R_2(CSJDQV)$



- J→S violates 3NF in R2, so decompose
  - $R_3(JS)$, $R_4(CJDQV)$



- No more violations! Are all FDs preserved? No so add $R_5(CJP)$
  - Final answer$R_1(SDP)$, $R_3(JS)$, $R_4(CJDQV)$, $R_5(CJP)$

Clicker Question: BCNF and 3NF

- Consider the following relation and functional dependencies:
  R(ABCD) FD's: ACD → B ; AC → D ; D → C ; AC → B
  Which of the following is true:

A. R is in neither BCNF nor 3NF
B. R is in BCNF but not 3NF
C. R is in 3NF but not in BCNF
D. R is in both BCNF and 3NF

Clicker Question: BCNF and 3NF

- Consider the following relation and functional dependencies:
  R(ABCD) FD's: ACD → B ; AC → D ; D → C ; AC → B
  Which of the following is true:

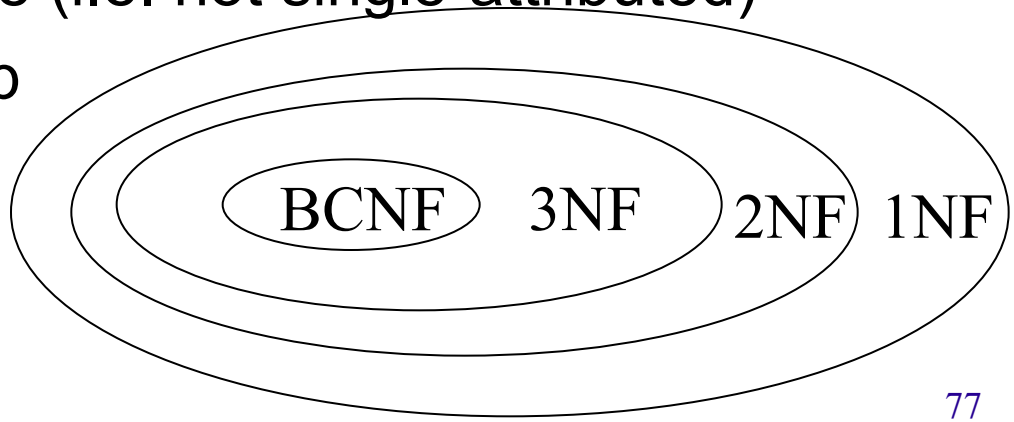
A. R is in neither BCNF nor 3NF
B. R is in BCNF but not 3NF
C. R is in 3NF but not in BCNF
D. R is in both BCNF and 3NF

ACD^+=ABCD
AC^+=ACDB
D^+=DC
AD^+=ADCB
Keys: AC, AD

D→C (D not key so NOT BCNF)
C is part of a minimal key so R is in 3NF

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.
- A 3NF relation R may not be in BCNF if all 3 of the following conditions are true:
  a. R has multiple keys
  b. Keys are composite (i.e. not single-attributed)
  c. These keys overlap



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

On the other hand… Denormalization

- Process of intentionally violating a normal form to gain performance improvements
- Performance improvements:
  - Fewer joins
  - Reduces number of foreign keys
    - Since FD's are often indexed, the number of indexes many be reduced
- Useful if certain queries often require (joined) results, and the queries are frequent enough

Learning Goals Revisited

- Debate the pros and cons of redundancy in a database.
- Provide examples of update, insertion, and deletion anomalies.
- Given a set of tables and a set of functional dependencies over them, determine all the keys for the tables.
- Show that a table is/isn't in 3NF or BCNF.
- Prove/disprove that a given table decomposition is a lossless join decomposition. Justify why lossless join decompositions are preferred decompositions.
- Decompose a table into a set of tables that are in 3NF, or BCNF.