CMPT 321
FALL 2017

Design Theory for Relational Databases

Self-study material

By Marina Barsky

Kent, W. (1983) [A Simple Guide to Five Normal Forms in Relational Database Theory]

BCNF decomposition: formally

Textbook: 3.1 – 3.3

Functional dependencies: formal definition

- $X \rightarrow Y$ is an assertion about a relation R that whenever two tuples of R agree on all the attributes X, then they must also agree on all attributes in set Y.

- Say "$X \rightarrow Y$ holds in R."
- Convention: …, X, Y, Z represent sets of attributes; A, B, C,… represent single attributes.
- Convention: no set formers in sets of attributes, just ABC, rather than $\{A,B,C\}$.

Formal example of FDs

- AC → B

A	B	C
5	3	2
5	4	3
5	5	2

Does this instance violate AC → B?

Formal example of FDs

- AC → B

A	B	C
5	3	**2**
5	4	3
5	5	**2**

Does this instance violate AC → B?

Keys: formal definition

- *K* is a ***superkey*** for relation *R* if *K* functionally determines all of *R*

- *K* is a ***key*** for *R* if *K* is a superkey, but no proper subset of *K* is a superkey

Formal example of keys

- Suppose R is a relation with attributes A, B, C
- Tell how many superkeys R has if the only key is A?

Formal example of keys

- Suppose R is a relation with attributes A, B, C
- Tell how many superkeys R has if the only key is A?
- Superkeys:
  - A
  - AB
  - ABC
  - AC

Inferring FD's

- We are given FD's $X_1 \rightarrow A_1, X_2 \rightarrow A_2, \ldots, X_n \rightarrow A_n$, and we want to know whether an FD $Y \rightarrow B$ must hold in any relation that satisfies the given FD's.

- Example: If $A \rightarrow B$ and $B \rightarrow C$ hold, does $A \rightarrow C$ hold?

Inference rules

- Splitting rule
- Transitive rule
- Trivial FDs
- Closure

Splitting (and combining) rule

- Splitting right sides of FD's:
  - $X \rightarrow A_1A_2 \ldots A_n$ holds for R precisely when each of $X \rightarrow A_1$, $X \rightarrow A_2, \ldots, X \rightarrow A_n$ hold for R.

- Combining right sides of FD's:
  - when $X \rightarrow A_1$, $X \rightarrow A_2, \ldots, X \rightarrow A_n$ hold then $X \rightarrow A_1A_2 \ldots A_n$ holds

- **There is no splitting (combining) rule for left sides!**
- We'll generally express FD's with singleton right sides

Splitting rule reasoning

- Suppose we have $A \rightarrow BC$
- This is an assertion that if 2 tuples agree on A, they also agree in all B and C
- That means that they agree in B and they agree in C: $A \rightarrow B$, $A \rightarrow C$

$A \rightarrow BC$			$A \rightarrow B$	$A \rightarrow C$
(a, b, c)	(a, b, c)		(a, b, c)	(a, b, c)
(a, b1, c1)	(a, b, c)	⇒	(a, b, c1)	(a, b1, c)
(a, b2, c2)	(a, b, c)		(a, b, c2)	(a, b2, c)

Inference rules

- Splitting rule
- Transitive rule
- Trivial FDs
- Closure

Transitive rule

- If X → Y and Y → Z then X → Z

A → B

(a, b, c)
(a, b1, c1)
(a, b2, c2)

A → B

(a, b, c)
(a, b, c1)
(a, b, c2)

B → C

(a, b, c)
(a, b, c1)
(a, b, c2)

B → C

(a, b, c)
(a, b, c)
(a, b, c)

A → C

(a, b, c)
(a, b, c)
(a, b, c)

Inference rules

- Splitting rule
- Transitive rule
- Trivial FDs
- Closure

Trivial FD's

- If $X \rightarrow Y$ and $Y \subseteq X$ then $X \rightarrow Y$ is called a trivial dependency
- Explanation: All tuples that agree in all of X surely agree in a subset of them
- Example: $AB \rightarrow B$ is a trivial dependency

Inference Test

- To test if $Y \rightarrow B$, start by assuming two tuples agree in all attributes of Y
- Use the given FD's to infer that these tuples must also agree in certain other attributes.
  - If B is one of these attributes, then $Y \rightarrow B$ is true.
  - Otherwise, the two tuples, with any forced equalities, form a two-tuple relation that proves $Y \rightarrow B$ does not follow from the given FD's.

Inference rules

- Splitting rule
- Transitive rule
- Trivial FDs
- Closure

Closure for a set of attributes Y

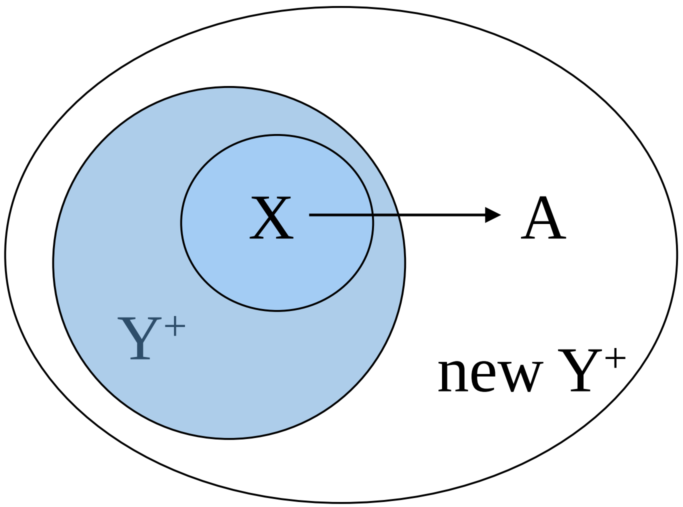
- The *closure* of a set Φ of functional dependencies is the set of all functional dependencies logically implied by Φ

- The closure for an attribute set Y is a set of all implied dependencies with Y in the left-hand side

- The *closure* of Y is denoted Y^+.

Computing closure for a set of attributes Y

- Convert all FDs to LHS-singleton FD's using splitting rule
- Basis: $Y^+ = Y$.
- Induction: Look for an FD's left side X that is a subset of the current Y^+. If the FD is $X \rightarrow A$, add A to Y^+.



Example: computing closure

- Given:

R(A,B,C,D) with FD's AB $\rightarrow$ C, B $\rightarrow$ D, CD $\rightarrow$ A, AD $\rightarrow$ B.

- Computing closure for **AB**:

$\{AB\}^+ = \{ABC\}$ (from AB $\rightarrow$ C)

$\{ABC\}^+ = \{ABCD\}$ (from B $\rightarrow$ D)

- Answer:

$\{AB\}^+ = \{ABCD\}$

Example: computing closure

- Given:

R(A,B,C,D) with FD’s AB $\rightarrow$ C, B $\rightarrow$ D, CD $\rightarrow$ A, AD $\rightarrow$ B.

- Computing closure for **B**:

$\{B\}^+ = \{BD\}$ (from B $\rightarrow$ D)

- Answer:

$\{B\}^+ = \{BD\}$

Example: computing closure

- Given:

R(A,B,C,D) with FD's AB → C, B → D, CD → A, AD → B.

- Computing closure for CD:

$\{CD\}^+ = \{CDA\}$ (from CD → A)

$\{CDA\}+=\{CDAB\}$ (from AD → B)

- Answer:

$\{CD\}^+ = \{ABCD\}$

Example: computing closure

- Given:

R(A,B,C,D) with FD's AB → C, B → D, CD → A, AD → B.

- Computing closure for **AD**:

$\{AD\}^+ = \{ADB\}$ (from AD → B)

{ADB}+={ADBC} (from AB → C)

- Answer:

$\{AD\}^+ = \{ABCD\}$

Why do we need to compute closure

- By computing closure for every possible set of attributes we obtain a full exhaustive set of FD’s – both declared and implied

- Closure has multiple applications

Using closure to test for an FD

- Suppose R(A,B,C,D,E,F) and the the FD's are

AB→C, BC→AD, D→E, and CF→B

- We wish to test whether AB→D follows from the set of FD's?

- We compute $\{A,B\}^+$ which is {A,B,C,D,E}.
- Since D is a member of the closure, we imply AB→D

Using closure to test for an FD

- Consider the relation R(A, B, C, D, E) and the set of FD's S1 = {AB->C, AE->D, D->B}

- Which of the following assumptions does not follow from S1

1. S2={AD->C}
2. S2={AD->C, AE->B}
3. S2 = {ABC->D, D->B}
4. S2 = {ADE->BC}

Using closure to test for a key

One way of testing if a set of attributes, let's say A, is a key, is:

1. Find it's closure A^+.
2. Make sure that it contains all attributes of R.
3. Make sure that you cannot create a smaller set, let's say A', by removing one or more attributes from A, that has the property 2.

Using closure to compute all superkeys

- Given:

R(A,B,C,D) with FD’s AB $\rightarrow$ C, B $\rightarrow$ D, CD $\rightarrow$ A, AD $\rightarrow$ B.

$\{AB\}^+ = \{ABCD\}$

$\{B\}^+ = \{BD\}$

$\{CD\}^+ = \{ABCD\}$

$\{AD\}^+ = \{ABCD\}$

{AB}, {CD}, {AD} are superkeys

Using superkeys for identifying candidate keys

R(A,B,C,D) with FD's AB → C, B → D, CD → A, AD → B.

{AB}, {CD}, {AD} are superkeys

Can A be a key?

$\{A\}^+ = \{A\}$ – no

Can B be a key?

$\{B\}^+ = \{BD\}$ – no

{AB} is a key – minimal superkey

Analogous tests show that {CD} and {AD} are also keys

Boyce-Codd Normal Form: formal definition

- Boyce-Codd Normal Form (BCNF): simple condition under which all the anomalies of 2NF, 3NF and BCNF can be guaranteed not to exist.

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

  Whenever there is a *nontrivial* dependency

  $A_1A_2\ldots A_n \rightarrow B_1B_2\ldots B_m$

  for **R**, it must be the case that

  $\{A_1, A_2, \ldots, A_n\}$ is a **superkey** for **R**.

One more time: relation is in BCNF when

whenever ***X*** $\rightarrow$ ***Y*** is a nontrivial FD that holds in *R*,

X is a **superkey**

- Remember: *nontrivial* means *Y* is not contained in *X*.
- Remember, a *superkey* is any superset of a key (not necessarily a proper superset).

Example BBD

Beers(<u>name</u>, manf, manfAddr)

- FD's: name → manf,   manf → manfAddr
- Only key is {name} .

- name → manf - does not violate BCNF
- manf → manfAddr - violation

Decomposition into BCNF

- Find a non-trivial FD $A_1A_2 \ldots A_n \rightarrow B_1B_2 \ldots B_m$ that violates BCNF, i.e. $A_1A_2 \ldots A_n$ isn't a superkey.
- Decompose the relation into two overlapping relations:
  - One is all the attributes involved in the violating dependency and
  - the other is the **left side of the violating FD** and all the other attributes not involved in the violating FD

- By repeatedly, choosing suitable decompositions, we can break any relation schema into a collection of smaller relations, each in BCNF.

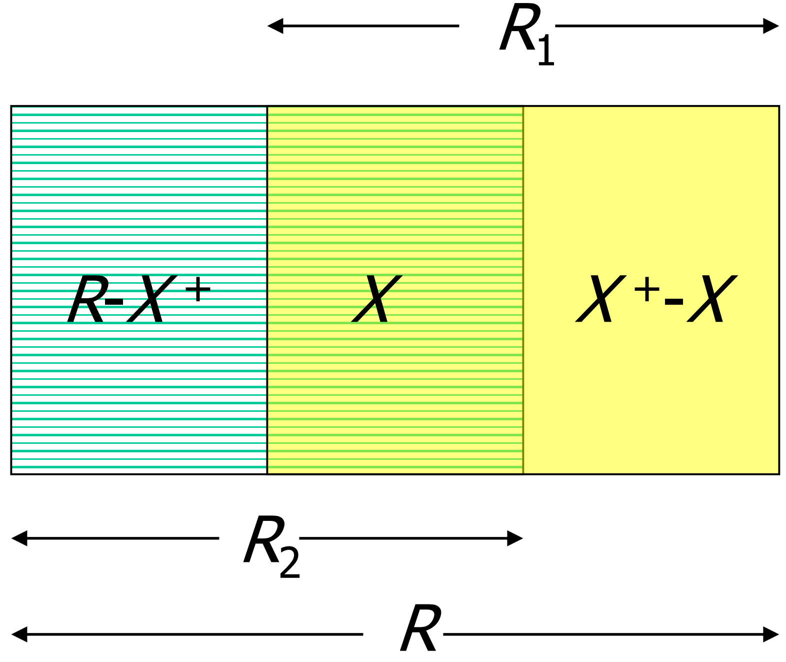
BCNF decomposition algorithm: step 1

- Given: relation *R* with FD's *F*

- Look among the given FD's for a BCNF violation $X \rightarrow Y$
- Compute X^+.
  - Not all attributes, or else X is a superkey

BCNF decomposition algorithm: step 2

- Replace R by relations with schemas:
  1. $R_1 = X^+$
  2. $R_2 = R - (X^+ - X)$



BCNF decomposition algorithm: step 3

- Identify all new FD’s in R1 and R2
- For each R1 and R2 – if any dependency violates BCNF - go to step 1

- Until no more BCNF violations

Formal Example 1/5

- Given R(A,B,C,D) with AB → C, C → D, and D → A
- Indicate all BCNF violations

$\{AB\}^{+}=\{ABCD\}$ - not a violation, {AB} is (super)key

$C^{+} = \{CDA\}$ – violation

$D^{+} = \{DA\}$ - violation

Formal Example 2/5

- Given R(A,B,C,D) with AB → C, C → D, and D → A

C^+ = {CDA} – violation

D^+ = {DA} – violation

- Decompose into relations that are in BCNF

- Variant 1:

R1 (C, D, A}

R2 (B, C)

Formal Example 3/5

- Given R(A,B,C,D) with AB → C, C → D, and D → A

C^+ = {CDA} – violation

D^+ = {DA} – violation

- Decompose into relations that are in BCNF

- Variant 2:

R1 (D, A}

R2 (B, C, D)

Formal example 4/5

- R(A,B,C,D) with AB → C, C → D, and D → A

R1 (C, D, A}

R2 (B, C)

- Should we stop? No, we need to test R1 and R2 for BCNF violations

- Which FD's do we have in R1?

C → D, and D → A

C^+ = {CDA} – not a violation

D^+ = {DA} - violation

Formal example 5/5

- R(A,B,C,D) with AB $\rightarrow$ C, C $\rightarrow$ D, and D $\rightarrow$ A

R1 (C, D, A}

R2 (B, C)

- Decomposing R1 with C $\rightarrow$ D, and D $\rightarrow$ A

D^{+} = {DA} – violation

R1.1 (D,A)

R1.2 (C, D)

Final result

- R(A,B,C,D) with AB → C, C → D, and D → A

- Decomposed into:

R2 (B, C)

R1.1 (D,A)

R1.2 (C, D)

- Should we decompose any further?
- No, because every relation with 2 attributes is automatically in BCNF

Every relation with 2 attributes is in BCNF

- R (A, B)

3 cases:

- There are no non-trivial FD’s

No violations

- A → B holds

A is the key – no violations

- B → A holds

B is the key no violations

■

Desired properties of decompositions

Textbook: 3.4 – 3.5

We expect that after decomposition

- No anomalies and redundancies
- We can recover the original relation from the tuples in its decompositions
- We can ensure that after reconstructing the original relation from the decompositions, the original FD's hold

Desired properties of normalization: after decomposition

- No redundancies and anomalies
- Recoverability of information
- Preservation of original FD's

Recovering Information from a decomposition by join

- We have the relation R(A, B, C) and B→ C holds

A	B	C
a	b	c

Then we decompose R into R1 and R2 as follows:

A	B
a	b

B	C
b	c

Joining the two would get the R back.

Recovering Information from a decomposition by join: lossless join

- Getting the tuples we started back is not enough to show that the original relation R is truly represented by the decomposition.

A	B	C
a	b	c
a1	b	c1

We have the relation R(A, B, C) and B→ C holds

Then we decompose R into R1 and R2 as follows:

A	B
a	b
a1	b

B	C
b	c
b	c1

Recovering Information from a decomposition by join: lossless join

- Getting the tuples we started back is not enough to show that the original relation R is truly represented by the decomposition.

A	B	C
a	b	c
a1	b	c1

We have the relation R(A, B, C) and B→ C holds

Then we decompose R into R1 and R2 as follows:

A	B
a	b
a1	b

B	C
b	c

Because we decomposed along B→ C, we can conclude that c1=c are the same so really there is only one tuple in R2

Recovering Information from a non-BCNF decomposition: lossy join

- Note that the FD should exist, otherwise the join wouldn't reconstruct the original relation
- Example: we have the relation R(A, B, C) but neither B → A nor B→ C holds.

A	B	C
a	b	c
a1	b	c1

Then we decompose R into R1 and R2 as follows:

A	B
a	b
a1	b

B	C
b	c
b	c1

Recovering Information from a non-BCNF decomposition: lossy join

- Since both R1 and R2 share the same attribute B, if we natural join them, we'll get:

A	B
a	b
a1	b

$\bowtie$

B	C
b	c
b	c1

=

A	B	C
a	b	c
a	b	c1
a1	b	c
a1	b	c1

- We got two bogus tuples, (a, b, c1) and (a1, b, c), which were not in the original relation

A	B	C
a	b	c
a1	b	c1

Testing for a lossless Join

- If we project R onto R_1, R_2,…, R_k , can we recover R by rejoining?
- Any original tuple in R surely can be recovered from its projected fragments.
- So the only question is: **when we rejoin, do we ever get back something we didn’t have originally**?

Chase test for lossless join

- An organized way of proving that any tuple t in $R_1 \bowtie R_2 \bowtie \ldots R_k$ is in the original relation R
- We construct an example of the original relation in a special way, representing the decompositions by leaving the corresponding values unsubscribed
- This representation is called a **Tableau** (example on the next page)

Example: Tableau

- Relation R(A, B, C, D)
- Decomposed into:

R1 (A,D)

R2 (A, C)

R3 (B, C, D)

Tuple t = (a, b, c, d)

A	B	C	D
a	b1	c1	**d**
a	b2	**c**	d2
a3	**b**	**c**	**d**

This row is a test case for R1(A,D). So we leave a and d unsubscribed, and label b1 and c1 as arbitrary values in row 1

Example: Tableau

- Relation R(A, B, C, D)
- Decomposed into:

R1 (A,D)

R2 (A, C)

R3 (B, C, D)

Tuple *t* = (a, b, c, d)

A	B	C	D
a	b1	c1	**d**
a	b2	**c**	d2
a3	**b**	**c**	**d**

This row is a test case for R2(A,C). So we leave a and c unsubscribed, and label b2 and d2 as arbitrary values in row 2

Example: Tableau

- Relation R(A, B, C, D)
- Decomposed into:

R1 (A,D)

R2 (A, C)

R3 (B, C, D)

Tuple t = (a, b, c, d)

A	B	C	D
a	b1	c1	**d**
a	b2	**c**	d2
a3	**b**	**c**	**d**

This row is a test case for R3(B,C,D). So we leave b, c, and d unsubscribed, and label a3 as arbitrary value in row 3

Goal: show that after project and join no new bogus tuples

- We **“chase”** the tableau applying FD’s one-by-one
- Relation R(A, B, C, D)
- FD’s:

A $\rightarrow$ B

B $\rightarrow$ C

CD $\rightarrow$ A

A	B	C	D
a	b1	c1	d
a	b2	c	d2
a3	b	c	d

Tableau

?

Project and join

A	B	C	D
a	b1	c1	d
a	b2	c	d2
a3	b	c	d

Chase test 1/4

- Relation R(A, B, C, D)
- FD's:

A → B

B → C

CD → A

A	B	C	D
a	b1	c1	d
a	b2	c	d2
a3	b	c	d

A	B	C	D
a	b1	c1	d
a	**b1**	c	d2
a3	b	c	d

Chase test 2/4

- Relation R(A, B, C, D)
- FD's:

A → B

B → C

CD → A

A	B	C	D
a	b1	c1	d
a	b2	c	d2
a3	b	c	d

A	B	C	D
a	b1	c1	d
a	b1	c	d2
a3	b	c	d

A	B	C	D
a	b1	**c**	d
a	b1	c	d2
a3	b	c	d

Chase test 3/4

- Relation R(A, B, C, D)
- FD's:

A → B

B → C

CD → A

A	B	C	D
a	b1	c1	d
a	b2	c	d2
a3	b	c	d

A	B	C	D
a	b1	c1	d
a	b1	c	d2
a3	b	c	d

A	B	C	D
a	b1	c	d
a	b1	c	d2
a3	b	c	d

A	B	C	D
a	b1	c	d
a	b1	c	d2
a	b	c	d

Chase test: conclusion

- Relation R(A, B, C, D)
- FD's:

A → B

B → C

CD → A

Once we have an entire row unsubscribed, we know that the decomposition is lossless – chase test is complete

A	B	C	D
a	b1	c1	d
a	b2	c	d2
a3	b	c	d

A	B	C	D
a	b1	c1	d
a	b1	c	d2
a3	b	c	d

A	B	C	D
a	b1	c	d
a	b1	c	d2
a3	b	c	d

A	B	C	D
a	b1	c	d
a	b1	c	d2
a	**b**	**c**	**d**

Chase test: conclusion

- Relation R(A, B, C, D)
- FD's:

A → B

B → C

CD → A

If you project this relation onto R1 (A,D), R2 (A, C), and R3 (B, C, D), and then join, you will get exactly the same original relation (you can check)

A	B	C	D
a	b1	c1	d
a	b2	c	d2
a3	b	c	d

A	B	C	D
a	b1	c1	d
a	b1	c	d2
a3	b	c	d

A	B	C	D
a	b1	c	d
a	b1	c	d2
a3	b	c	d

A	B	C	D
a	b1	c	d
a	b1	c	d2
a	**b**	**c**	**d**

Chase test: conclusion

- Relation R(A, B, C, D)
- FD's:

A → B

B → C

CD → A

The decomposition into R1 (A,D), R2 (A, C), R3 (B, C, D) is a **lossless** decomposition

A	B	C	D
a	b1	c1	d
a	b2	c	d2
a3	b	c	d

A	B	C	D
a	b1	c1	d
a	b1	c	d2
a3	b	c	d

A	B	C	D
a	b1	c	d
a	b1	c	d2
a3	b	c	d

A	B	C	D
a	b1	c	d
a	b1	c	d2
a	**b**	**c**	**d**

■

Chase test: another example

- Suppose we have relation R(A,B,C,D) with FD B→AD
- We have decomposed into

R1(A,B), R2(B,C), R3(C,D)

A	B	C	D
a	b	c1	d1
a2	b	c	d2
a3	b3	c	d

A	B	C	D
a	b	c1	d1
a	b	c	**d1**
a3	b3	c	d

The decomposition into R1{A,B}, R2{B,C}, R3{C,D} is a **lossy** decomposition

If you now project and join back, you will get bogus tuples, for example (a3, b3, c, d1) which was not in the original relation

Summary of the “Chase”

1. If two rows agree in the left side of a FD, make their right sides agree too.
2. Always replace a subscripted symbol by the corresponding unsubscripted one, if possible.
3. If we ever get an unsubscripted row, we know any tuple in the project-join is in the original (the join is lossless).
4. Otherwise, the final tableau is a counterexample.

Desired properties of normalization: after decomposition

- No redundancies and anomalies
- Recoverability of information
- Preservation of original FD's

Preservation of original FD’s

- Most BCNF decompositions preserve original FD’s
- There are special cases when the original relation cannot be decomposed into BCNF and preserve original FD’s

BCNF decomposition which does not preserve FD's

- There is one structure of FD's that causes trouble when we decompose.

AB $\rightarrow$ ***C*** and ***C*** $\rightarrow$ ***B***

- There are two keys, {*A*,*B*} and {*A*,*C*}
- ***C*** $\rightarrow$ ***B*** is a BCNF violation, so we must decompose into *AC*, *BC*

- The difference here that a violating FD ***C*** $\rightarrow$ ***B*** has B in RHS, and **B is a part of a primary key**
- An attribute that is a part of some key is called a ***prime***

Example: BCNF gone wrong

- Given R (client, bank, banker) with FD's:

{client, bank} → banker - {client, bank} is the key

banker → bank – violation

- We decompose into

R1 (banker, bank)

R2 (client, banker)

- However the original FD {client, bank} → banker is lost in this decomposition!

Example continued: at the moment of decomposition

- R (client, bank, banker)
- FD's:

{client, bank} → banker

banker → bank

{client, bank} → banker
banker → bank

R		
client	bank	banker
A	1	X
A	2	Y
B	1	X

- Decomposition:

R1 (banker, bank)

R2 (client, banker)

banker → bank

R1	
banker	bank
X	1
Y	2

No FD's

R1	
client	banker
A	X
A	Y
B	X

Example continued: lossless decomposition

banker → bank

R1	
banker	bank
X	1
Y	2

⋈

No FD's

R2	
client	banker
A	X
A	Y
B	X

{client, bank} → banker
banker → bank

R		
client	bank	banker
A	1	X
A	2	Y
B	1	X

The decomposition is lossless – requirement 2 is satisfied

Example continued: no original constraint {client, bank} → banker

banker → bank

R1	
banker	bank
X	1
Y	1

The only requirement is that banker uniquely identifies bank

No FD's

R2	
client	banker
A	X
A	Y
B	X

Now we can insert into R1 and R2 without the original constraints, and that will allow to insert invalid values

Example continued: no original constraint {client, bank} → banker

banker → bank

R1	
banker	bank
X	1
Y	1

⋈

No FD's

R2	
client	banker
A	X
A	Y
B	X

{client, bank} → banker
banker → bank

R		
client	bank	banker
A	1	X
A	1	Y
B	1	X

Invalid join! Tuple (A, 1, Y) should have been prevented by the original FD {client, bank} → banker

Another example – zip code

R (city, street, zipcode)

- FD's:

{city, street} → zipcode

zipcode → city

R		
city	street	zipcode
A	X	10
B	X	20
A	Y	11
B	Y	20

R1	
zipcode	city
10	A
20	B
11	A

R2	
street	zipcode
X	10
X	20
Y	11
Y	20

It seems that we can still recover the original by join

Another example – concluded

R1	
zipcode	city
10	A
20	A
11	A

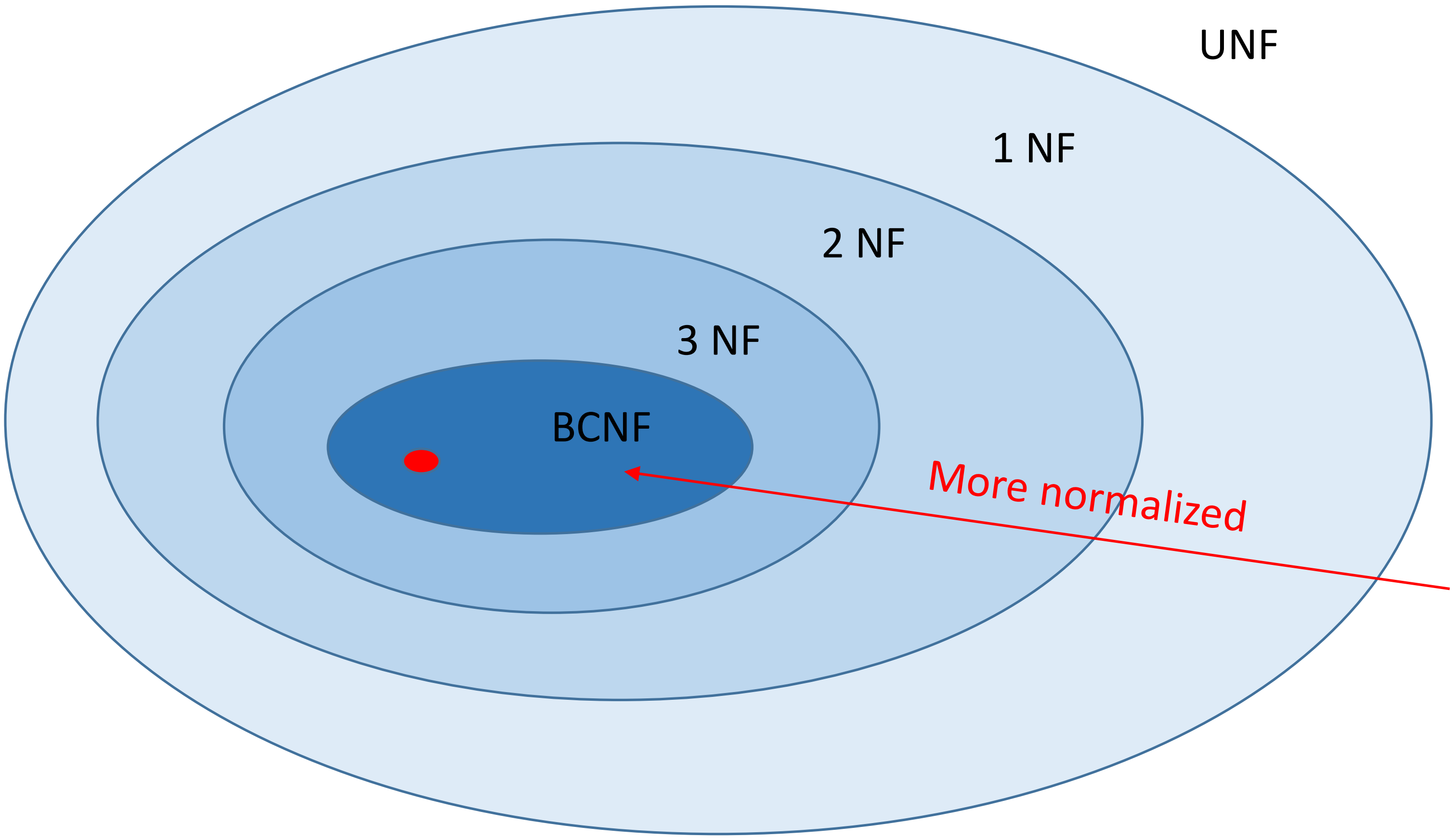
$\bowtie$

R2	
street	zipcode
X	10
X	20
Y	11
Y	20

R		
city	street	zipcode
A	X	10
A	X	20
A	Y	11
B	Y	20

But we are now free to enter invalid values into R1 and R2 because the original FD {city, street} → zipcode is lost!

Relationship between normal forms



Relaxing normalization requirements: 3NF

- 3^{rd} Normal Form (3NF) modifies the BCNF condition so we do not have to decompose in this problematic situation

- An attribute is ***prime*** if it is a member of any key.
- $X \rightarrow A$ violates 3NF if and only if X is not a superkey, and also A is not prime

Example: 3NF

- In our situation with FD's $AB \rightarrow C$ and $C \rightarrow B$, we have key *AB*
- Thus *A* and *B* are each prime.
- Although $C \rightarrow B$ violates BCNF, it **does not violate 3NF**
- So no decomposition is performed, and all the original FD's are preserved

Desired properties of normalization: after decomposition

- No redundancies and anomalies
- Recoverability of information
- Preservation of original FD’s

Desired properties of normalization: after decomposition: BCNF

- No redundancies and anomalies
- Recoverability of information
- Preservation of original FD's

Desired properties of normalization: after decomposition: 3NF

- No redundancies and anomalies
- Recoverability of information
- Preservation of original FD's

Multivalued Dependencies & Fourth Normal Form (4NF)

Textbook: 3.6 – 3.7

A New Form of Redundancy

- Multivalued dependencies (MVD's) express a condition among tuples of a relation that exists when the relation is trying to represent more than one many-many relationship.
- Then certain attributes become independent of one another, and their values must appear in all combinations.

Example

Drinkers (name, addr, phones, beersLiked)

- A drinker's phones are independent of the beers they like.
- Thus, each of a drinker's phones appears with each of the beers they like in all combinations.
  - If a drinker has 3 phones and likes 10 beers, then the drinker has 30 tuples
  - where each phone is repeated 10 times and each beer 3 times
- This repetition is unlike redundancy due to FD's, of which name->addr is the only one.

Tuples Implied by Independence

If we have tuples:

name	addr	phones	beersLiked
sue	a	p1	b1
sue	a	p2	b2

Then these tuples must also be in the relation:

name	addr	phones	beersLiked
sue	a	p2	b1
sue	a	p1	b2

Definition of MVD

- A *multivalued dependency* (MVD) *X* ->->*Y* is an assertion that if two tuples of a relation agree on all the attributes of *X*, then their components in the set of attributes *Y* may be swapped, and the result will be two tuples that are also in the relation.

Example

Drinkers (<u>name</u>, addr, <u>phones</u>, <u>beersLiked</u>)

FD: name -> addr

MVD's: name ->-> phones

name ->-> beersLiked

- Key is
  - {name, phones, beersLiked}.
- Which dependencies violate 4NF ?
  - All

Example, Continued

- Decompose using name -> addr:

1. Drinkers1 (name, addr)
   - In 4NF, only dependency is name -> addr.
2. Drinkers2(name, phones, beersLiked)
   - Not in 4NF. MVD's name ->-> phones and name ->-> beersLiked apply.
   - Key?
     - No FDs, so all three attributes form the key.

Example: Decompose Drinkers2

- Either MVD name ->-> phones or name ->-> beersLiked tells us to decompose to:
  - Drinkers3(name, phones)
  - Drinkers4(name, beersLiked)

Fourth Normal Form

- The redundancy that comes from MVD's is not removable by putting the database schema in BCNF.
- There is a stronger normal form, called 4NF, that (intuitively) treats MVD's as FD's when it comes to decomposition, but not when determining keys of the relation.

4NF Definition

- A relation *R* is in 4NF if whenever *X* ->->*Y* is a nontrivial MVD, then *X* is a superkey.
  - Nontrivial means that:
    1. *Y* is not a subset of *X*, and
    2. *X* and *Y* are not, together, all the attributes.
  - Note that the definition of "superkey" still depends on FD's only.

BCNF Versus 4NF

- Remember that every FD *X* ->*Y* is also an MVD, *X* ->->*Y*.
- Thus, if *R* is in 4NF, it is certainly in BCNF.
  - Because any BCNF violation is a 4NF violation.
- But *R* could be in BCNF and not 4NF, because MVD's are "invisible" to BCNF.

Decomposition and 4NF

- If $X \twoheadrightarrow Y$ is a 4NF violation for relation R, we can decompose R using the same technique as for BCNF.
  1. XY is one of the decomposed relations.
  2. All but $Y - X$ is the other.

Example

Drinkers (name, areaCode, phone, beersLiked, manf)

- A drinker can have several phones, with the number divided between areaCode and phone (last 7 digits).
- A drinker can like several beers, each with its own manufacturer.

Example, Continued

- Since the areaCode-phone combinations for a drinker are independent of the beersLiked-manf combinations, we expect that the following MVD's hold:

  name ->-> areaCode phone

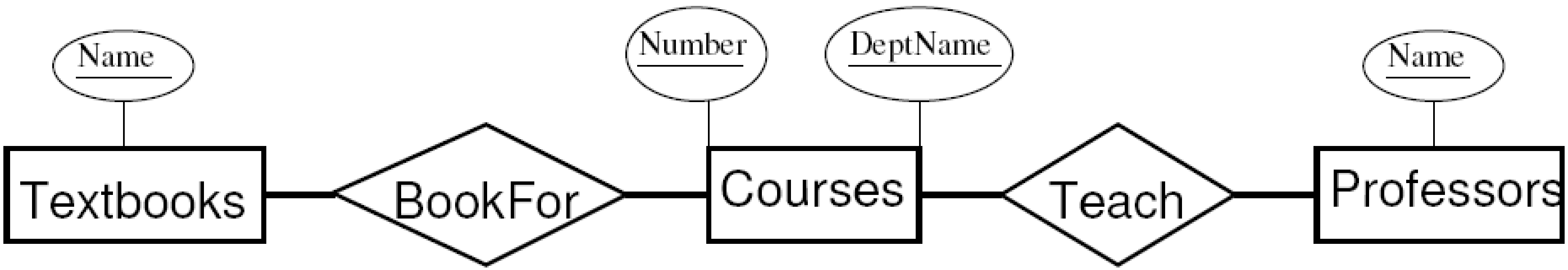
  name ->-> beersLiked manf

Example Data

Here is possible data satisfying these MVD's:

name	areaCode	phone	beersLiked	manf
Sue	650	555-1111	Bud	A.B.
Sue	650	555-1111	WickedAle	Pete's
Sue	415	555-9999	Bud	A.B.
Sue	415	555-9999	WickedAle	Pete's

Another Example



- The relation is Courses(Number, DeptName, Textbook, Professor).
  - Each Course can have multiple required Textbooks.
  - Each Course can have multiple Professors.
  - Professors uses every required textbook while teaching a Course.

Number	DeptName	Textbook	Professor
4604	CS	FCDB	Ullman
4604	CS	SQL Made Easy	Ullman
4604	CS	FCDB	Widom
4604	CS	SQL Made Easy	Widom

- The relation is in BCNF since there are no non-trivial FDs.
- Is there any redundancy?

Relationships Among Normal Forms

Property	3NF	BCNF	4NF
Eliminates redundancy due to FDs	Maybe	Yes	Yes
Eliminates redundancy due to MDs	No	No	Yes
Preserves FDs	Yes	Maybe	Maybe
Preserves MDs	Maybe	Maybe	Maybe