By Marina Barsky

Desired properties of decompositions

Lecture 16

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

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

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 → B

B → C

CD → 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

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

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

A → B

B → C

CD → A

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

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

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

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**

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)

Chase test: conclusion

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

$A \rightarrow B$

$B \rightarrow C$

$CD \rightarrow A$

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* → *C and ***C* → *B***

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

- The difference here that a violating FD ***C* → *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

No FD's

R2	
client	banker
A	X
A	Y
B	X

The only requirement is that banker uniquely identifies bank

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

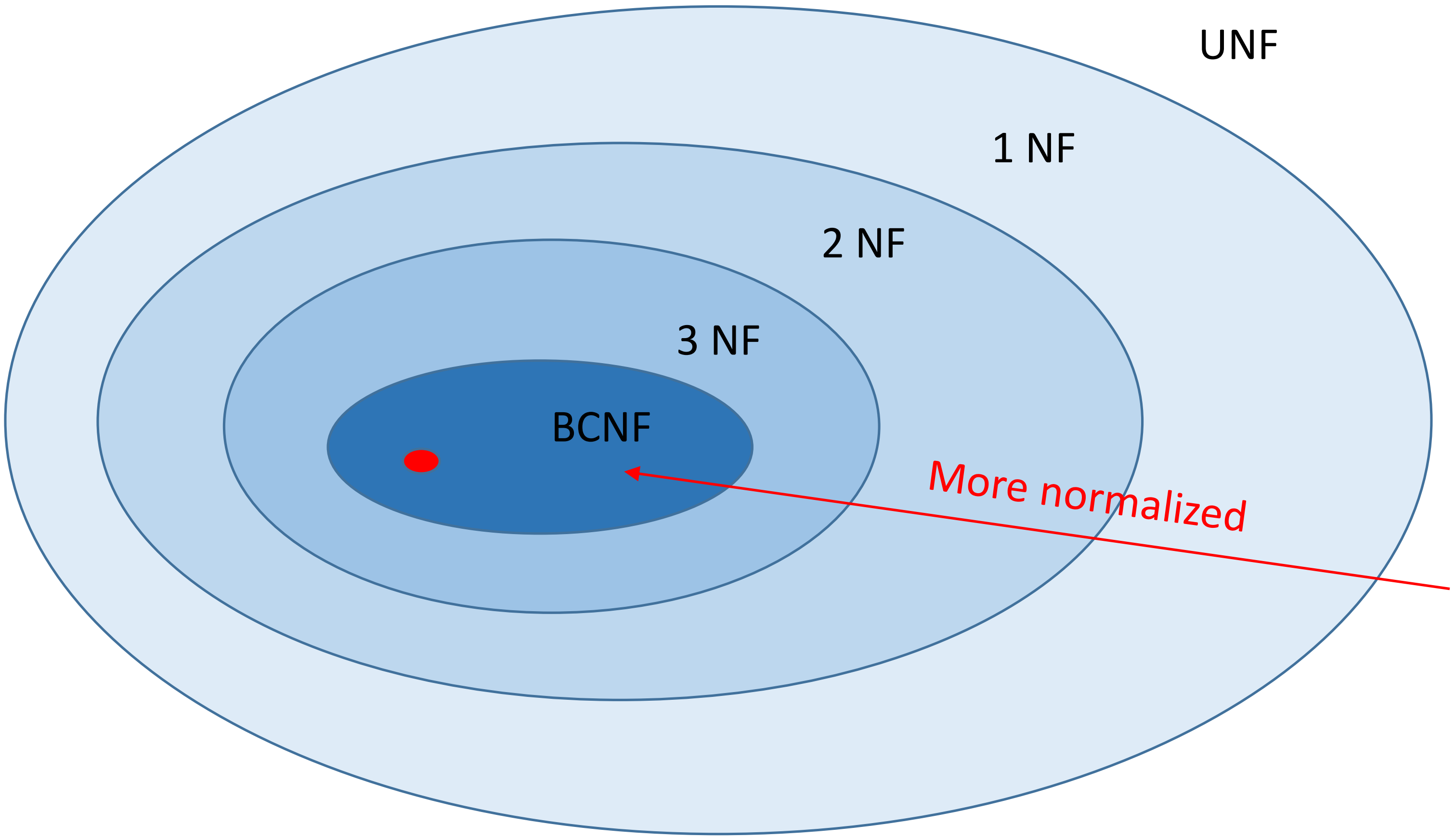
⋈

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

- 3rd 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 ✓

Decomposition into 3NF

- We can always perform a decomposition into 3NF relations with a lossless join and dependency preservation.

- Need to compute ***minimal basis*** for the FD's:
    1. Right sides are single attributes.
    2. No FD can be removed.
    3. No attribute can be removed from a left side.

Constructing a Minimal Basis

1. Split right sides.
2. Repeatedly try to remove an FD and see if the remaining FD’s are equivalent to the original.
3. Repeatedly try to remove an attribute from a left side and see if the resulting FD’s are equivalent to the original.

3NF Synthesis algorithm

- Compute minimal basis
- Split into one relation per FD in the minimal basis.
  - Schema is the union of the left and right sides.
- If no key is contained in an FD, then add one relation whose schema is some key.

Example: 3NF Synthesis

- Relation R = ABCD.
- FD's $A \rightarrow B$ and $A \rightarrow C$
- These FD's form minimal basis

- Decomposition:

AB and AC from the FD's, plus AD for a key.

Why 3NF Synthesis Works

- Preserves dependencies: each FD from a minimal basis is contained in a relation, thus preserved.
- Lossless Join: use the chase to show that the row for the relation that contains a key can be made all-unsubscripted variables.

- hard algorithmically – finding minimal bases.