Relational Design Theory

Carla Teixeira Lopes

Bases de Dados
Mestrado Integrado em Engenharia Informática e Computação, FEUP

Based on Jennifer Widom and Christopher Ré slides

Does BCNF guarantee a good decomposition?

Remove anomalies?

Yes

Can logically reconstruct original relation?

$R_1 \bowtie R_2 = R$?

Too few or too many tuples?

R

A	B	C
1	2	3
4	2	5

R_1

A	B
1	2
4	2

R_2

B	C
2	3
2	5

$R_1 \bowtie R_2 =$

123

125

423

425

Does BCNF guarantee a good decomposition?

R

A	B	C
1	2	3
4	2	5

R_1

A	B
1	2
4	2

R_2

B	C
2	3
2	5

$R_1 \bowtie R_2 =$ 123
125
423
425

What happened?

Not a BCNF decomposition

R1 and R2 would demand B $\rightarrow$ A or B $\rightarrow$ C and none hold

BCNF always lossless

BCNF decomposition is standard practice - very powerful & widely used!

The Chase Test for Lossless Join

S (A, B, C, D) decomposed in

S1 (A, D), S2 (A,C) and S3 (B, C, D)

DFs

A->B; A->C; CD->A

Does this decomposition ensure lossless joins?

The Chase Test for Lossless Join

S (A, B, C, D) decomposed in

- S1 (A, D), S2 (A,C) and S3 (B, C, D)
- A->B; A->C; CD->A

Build the tableau

- One line per decomposed relation
- A letter per each attribute in the decomposed relation
- Subscript the letter with *i*, if the attribute is not in S_i

A	B	C	D

A	B	C	D
a			d
a		c	
	b	c	d

A	B	C	D	
a	b_1	c_1	d	S1 (A, D)
a	b_2	c	d_2	S2 (A, C)
a_3	b	c	d	S3 (B, C, D)

The Chase Test for Lossless Join

S (A, B, C, D) decomposed in

- S1 (A, D), S2 (A,C) and S3 (B, C, D)
- A->B; B->C; CD->A

A	B	C	D
a	b_1	c_1	d
a	b_2	c	d_2
a_3	b	c	d

A->B

A	B	C	D
a	b_1	c_1	d
a	b_1	c	d_2
a_3	b	c	d

B->C

A	B	C	D
a	b_1	c	d
a	b_1	c	d_2
a_3	b	c	d

CD->A

A	B	C	D
a	b_1	c	d
a	b_1	c	d_2
a	b	c	d

Iterations

- A->B tells us that the first two rows must agree in the B attribute, that is, b_1=b_2
- From B->C, we know that c_1=c
- From CD->A, we know that a=a_3

The Chase Test for Lossless Join

S (A, B, C, D) decomposed in

- S1 (A, D), S2 (A,C) and S3 (B, C, D)
- A->B; B->C; CD->A

Conclusion

- If the final table has a line without subscripts, it is a lossless join decomposition

A	B	C	D
a	b_1	c	d
a	b_1	c	d_2
a	b	c	d

Exercise

Consider the following relation and FDs

Movie (title, year, studioName, president, presAddr)

title, year -> studioName

studioName -> president

president -> presAddr

Test if the following decomposition is lossless:

S1 (studioName, president)

S3 (studioName, presAddr)

S4 (studioName, title, year)

Exercise

Movie (title, year, studioName, president, presAddr) decomposed in

S1 (studioName, president); S3 (studioName, presAddr);

S4 (studioName, title, year)

title, year -> studioName

studioName -> president

president -> presAddr

Build the tableau

A (title)	B (year)	C (studioName)	D (president)	E (presAddr)	
a_1	b_1	c	d	e_1	S1
a_2	b_2	c	d_2	e	S3
a	b	c	d_3	e_3	S4

Exercise

Movie (title, year, studioName, president, presAddr) decomposed in

S1 (studioName, president); S3 (studioName, presAddr); S4 (studioName, title, year)

title, year -> studioName; studioName -> president; president -> presAddr

title	year	studio Name	presi dent	pres Addr
a_1	b_1	c	d	e_1
a_2	b_2	c	d_2	e
a	b	c	d_3	e_3

studioName->president

title	year	studio Name	presi dent	pres Addr
a_1	b_1	c	d	e_1
a_2	b_2	c	d	e
a	b	c	d	e_3

president->presAddr

title	year	studio Name	presi dent	pres Addr
a_1	b_1	c	d	e
a_2	b_2	c	d	e
a	b	c	d	e

lossless join decomposition

BCNF shortcomings – Example 1

Apply(SSN, cName, date, major)
- Can apply to each college once and for one major
- Colleges have non-overlapping application dates

FDs: SSN, cName -> date, major; date ->cName

Keys: {SSN, cName}

BCNF?
- No.
- A1 (date, cName)
- A2 (date, SSN, major)

Good design? Not necessarily.
- Student's application separated from the college
- Checking the first DF would require a join
- We might just prefer to keep everything together

Apply is in 3NF

BCNF shortcomings – Example 2

Unit	Company	Product
…	…	…

{Unit} → {Company}
{Company,Product} → {Unit}

Unit	Company
…	…

{Unit} → {Company}

Unit	Product
…	…

BCNF decomposition on:
{Unit} → {Company}

We lose the FD {Company, Product} -> {Unit}

BCNF shortcomings – Example 2

<u>Unit</u>	Company
Galaga99	UW
Bingo	UW

Unit	Product
Galaga99	Databases
Bingo	Databases

No problem so far. All *local* FDs are satisfied.

{Unit} → {Company}

Unit	Company	Product
Galaga99	UW	Databases
Bingo	UW	Databases

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

Violates the FD {Company, Product} -> {Unit}

BCNF shortcomings – Example 3

College (cName, state)
CollegeSize (cName, enrollment)
CollegeScores (cName, avgSAT)
CollegeGrades (cName, avgGPA)

Too decomposed

- We could capture all of the information in one relation or a couple and still be in BCNF

BCNF shortcomings

Dependency preservation is not guaranteed

- No guarantee that all original dependencies can be checked on decomposed relations
- This may require joins of those relations in order to check them

Various ways to handle so that decompositions are all lossless / no FDs lost

- For example 3NF

Usually a tradeoff between redundancy / data anomalies and FD preservation

BCNF still most common

- With additional steps to keep track of lost FDs

3NF Decomposition Algorithm

Input: relation R + set *F* of FDs for R

Output: decomposition of R into 3NF relations with "lossless join" and "dependency preservation"

1. Find a minimal basis for F, say G

   Right sides with only 1 attribute

   No redundant FDs

   For each DF $\bar{X} \to \bar{A}$, compute $\bar{X}^+$ using the other DFs. If $A \subseteq \bar{X}^+$, the DF $\bar{X} \to \bar{A}$ is redundant

   No redundant attributes on the left sides

   Remove 1 attribute from left side and compute closure of the remaining attributes with the **original** DFs. If closure includes the right side, the attribute can be removed

3NF Decomposition Algorithm

Input: relation R + set *F* of FDs for R

Output: decomposition of R into 3NF relations with “lossless join” and “dependency preservation”

2. For each DF $\bar{X} \rightarrow \bar{A}$ in G, create a relation R’ ($\bar{X}$, $\bar{A}$)

   Previously, merge DFs with equal left sides

3. If none of the relations of step 2 is a superkey for R, add another relation for a key for R

3NF Decomposition Example

R (A, B, C, D, E); $\boxed{\text{AB->C, C->B and A->D}}$ Minimal base

1. Find a minimal basis for DFs

   Right sides with only 1 attribute?

   No redundant DFs?

   - $\{A, B\}^+ = \{A, B, D\}$ → It does not contain C thus the DF is essential
   - $\{C\}^+ = \{C\}$ → It does not contain B thus the DF is essential
   - $\{A\}^+ = \{A\}$ → It does not contain D thus the DF is essential

   No redundant attributes on left side?

   On AB-> C, remove A, getting B->C. $\{B\}^+ = \{B\}$. Since it does not contain C, the attribute A is essential

   On AB->C, remove B, getting A->C. {A}+={AD}. Since it does not contain C, the attribute B is essential

3NF Decomposition Example

R (A, B, C, D, E); AB->C, C->B and A->D

2. For each DF $\bar{X} \rightarrow \bar{A}$ in G, create a relation R' (X, A)

   R_1 (A, B, C)

   R_2 (C, B)

   R_3 (A, D)

3. If none of the relations of step 2 is a superkey for R, add another relation for a key for R

   Keys: {A, B, E}, {A, C, E}

   R_4 should be one of them

Exercise

Consider the following relation and FDs

Movie (title, year, studioName, president, presAddr)
title, year -> studioName
studioName -> president
president -> presAddr

Decompose into 3NF relations.

Any advantages over the BCNF decomposition?

BCNF and 3NF decomposition

BCNF decomposition

- Assures lossless joins
- Dependency preservation is not always possible

3NF decomposition

- Assures lossless joins and dependency preservation

Summary

Designing a database schema

- Usually many designs possible
- Some are (much) better than others!
- How do we choose?

Very nice theory for relational database design

- Normal forms - “good” relations
- Design by decomposition
- Usually intuitive and works well
- Some shortcomings
  - Dependency enforcement
  - Query workload
  - Over-decomposition

Kahoot time!

Any doubts?

Readings

Jeffrey Ullman, Jennifer Widom, A first course in Database Systems 3rd Edition

- Section 3.1 – Functional Dependencies
- Section 3.2 – Rules About Functional Dependencies
- Section 3.3 – Design of Relational Database Schemas
- Section 3.4 – Decomposition: The Good, Bad, and Ugly
- Section 3.5 – Third Normal Form