Decomposition & Schema Normalization

CS 564- Fall 2018

ACKs: *Dan Suciu, Jignesh Patel, AnHai Doan*

WHAT IS THIS LECTURE ABOUT?

- Bad schemas lead to redundancy
- To "correct" bad schemas: decompose relations
  - lossless-join
  - dependency preserving
- Desired normal forms
  - **BCNF**
  - **3NF**

DB DESIGN THEORY

- Helps us identify the “bad” schemas and improve them
    1. express constraints on the data: functional dependencies (**FDs**)
    2. use the FDs to decompose the relations

- The process, called normalization, obtains a schema in a “normal form” that guarantees certain properties
    - examples of normal forms: **BCNF**, **3NF**, …

SCHEMA DECOMPOSITION

WHAT IS A DECOMPOSITION?

We decompose a relation **R**(A_1, …, A_n) by creating

- $\mathbf{R_1}$(B_1, .., B_m)
- $\mathbf{R_2}$(C_1,…, C_l)
- where $\{B_1, \dots, B_m\} \cup \{C_1, \dots, C_l\} = \{A_1, \dots A_n\}$

- The instance of $\mathbf{R_1}$ is the projection of **R** onto B_1, .., B_m
- The instance of $\mathbf{R_2}$ is the projection of **R** onto C_1, .., C_l

EXAMPLE: DECOMPOSITION

SSN	name	age	phoneNumber
934729837	Paris	24	608-374-8422
934729837	Paris	24	603-534-8399
123123645	John	30	608-321-1163
384475687	Arun	20	206-473-8221

SSN	name	age
934729837	Paris	24
123123645	John	30
384475687	Arun	20

SSN	phoneNumber
934729837	608-374-8422
934729837	603-534-8399
123123645	608-321-1163
384475687	206-473-8221

DECOMPOSITION DESIDERATA

What should a good decomposition achieve?

1. minimize redundancy
2. avoid information loss (lossless-join)
3. preserve the FDs (dependency preserving)
4. ensure good query performance

EXAMPLE: INFORMATION LOSS

name	age	phoneNumber
Paris	24	608-374-8422
John	24	608-321-1163
Arun	20	206-473-8221

Decompose into:

$\mathbf{R_1}$(name, age)

$\mathbf{R_2}$(age, phoneNumber)

name	age
Paris	24
John	24
Arun	20

age	phoneNumber
24	608-374-8422
24	608-321-1163
20	206-473-8221

We can't figure out which phoneNumber corresponds to which person!

LOSSLESS-JOIN DECOMPOSITION

$$
\begin{tikzcd}
& \mathbf{R}(A, B, C) \arrow[dl] \arrow[dr] & \\
\mathbf{R_1}(A, B) \arrow[dr] & & \mathbf{R_2}(B, C) \arrow[dl] \\
& \mathbf{R'}(A, B, C) &
\end{tikzcd}
$$

decompose (projection)

recover (natural join)

A <u>natural join</u> is a join on the same attribute names

A schema decomposition is **<u>lossless-join</u>** if for any initial instance **R**, **R = R'**

A LOSSLESS-JOIN CRITERION

Starting with:

- a relation $\mathbf{R}(\mathbf{A})$ + set F of FDs
- a decomposition of $\mathbf{R}$ into $\mathbf{R_1}(\mathbf{A_1})$ and $\mathbf{R_2}(\mathbf{A_2})$

we say that a decomposition is lossless-join **if and only if** $\boldsymbol{A_1} \cap \boldsymbol{A_2}$ is a superkey either in $\mathbf{R_1}$ or in $\mathbf{R_2}$

EXAMPLE

- relation **R**(A, B, C, D)
- FD $A \longrightarrow B, C$

Lossless-join

- decomposition into $\mathbf{R_1}$(A, B, C) and $\mathbf{R_2}$(A, D)
- $\{A, B, C\} \cap \{A, D\} = \{A\}$
- For $\mathbf{R_1}$ we have indeed $A \longrightarrow B, C$

Not lossless-join

- decomposition into $\mathbf{R_1}$(A, B, C) and $\mathbf{R_2}$(D)

DEPENDENCY PRESERVING

Given **R** and a set of FDs F, we decompose **R** into $\mathbf{R_1}$ and $\mathbf{R_2}$. Suppose:

- $\mathbf{R_1}$ has a set of FDs F_1
- $\mathbf{R_2}$ has a set of FDs F_2
- F_1 and F_2 are computed from F

A decomposition is **dependency preserving** if by enforcing F_1 over $\mathbf{R_1}$ and F_2 over $\mathbf{R_2}$, we can enforce F over **R**

GOOD EXAMPLE

Person(SSN, name, age, canDrink)

- $SSN \longrightarrow name, age$
- $age \longrightarrow canDrink$

decomposes into

- $\mathbf{R_1}$(SSN, name, age)
  - $SSN \longrightarrow name, age$
- $\mathbf{R_2}$(age, canDrink)
  - $age \longrightarrow canDrink$

BAD EXAMPLE

R(A, B, C)

- $A \longrightarrow B$
- $B, C \longrightarrow A$

Decomposes into:

- **$\mathbf{R_1}$**(A, B)
  - $A \longrightarrow B$
- **$\mathbf{R_2}$**(A, C)
  - no FDs here!!

$\mathbf{R_1}$

A	**B**
a_1	b
a_2	b

$\mathbf{R_2}$

A	**C**
a_1	c
a_2	c

recover

A	**B**	**C**
a_1	b	c
a_2	b	c

The recovered table violates $B, C \longrightarrow A$

NORMAL FORMS

A **normal form** represents a "good" schema design:

- 1NF (flat tables/atomic values)
- 2NF
- **3NF**
- **BCNF**
- 4NF
- …

more restrictive

BCNF Decomposition

BOYCE-CODD NORMAL FORM (BCNF)

A relation **R** is in **BCNF** if whenever $X \longrightarrow B$ is a non-trivial FD, then X is a superkey in **R**

Equivalent definition: for every attribute set X

- either $X^+ = X$
- or $X^+ = all\ attributes$

BCNF EXAMPLE 1

SSN	name	age	phoneNumber
934729837	Paris	24	608-374-8422
934729837	Paris	24	603-534-8399
123123645	John	30	608-321-1163
384475687	Arun	20	206-473-8221

$$SSN \longrightarrow name, age$$

- **key** = $\{SSN, phoneNumber\}$
- $SSN \longrightarrow name, age$ is a "bad" FD
- The above relation is **not** in BCNF!

BCNF EXAMPLE 2

SSN	name	age
934729837	Paris	24
123123645	John	30
384475687	Arun	20

$$SSN \longrightarrow name, age$$

- **key** = $\{SSN\}$
- The above relation is in BCNF!

BCNF EXAMPLE 3

SSN	phoneNumber
934729837	608-374-8422
934729837	603-534-8399
123123645	608-321-1163
384475687	206-473-8221

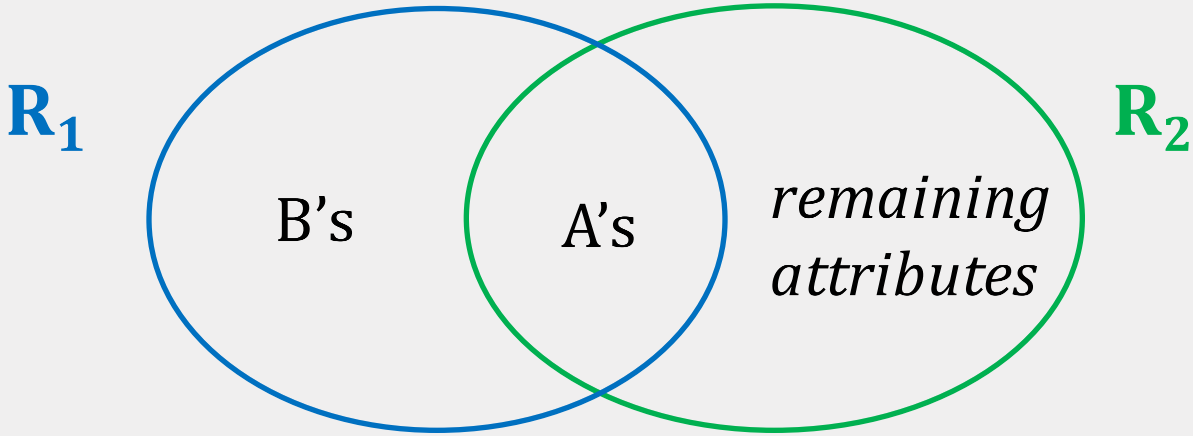
- **key** = $\{SSN, phoneNumber\}$
- The above relation is in BCNF!
- Is it possible that a binary relation is not in BCNF?

BCNF DECOMPOSITION

- Find an FD that violates the BCNF condition

$$A_1, A_2, \ldots, A_n \longrightarrow B_1, B_2, \ldots, B_m$$

- Decompose **R** to **R_1** and **R_2**:

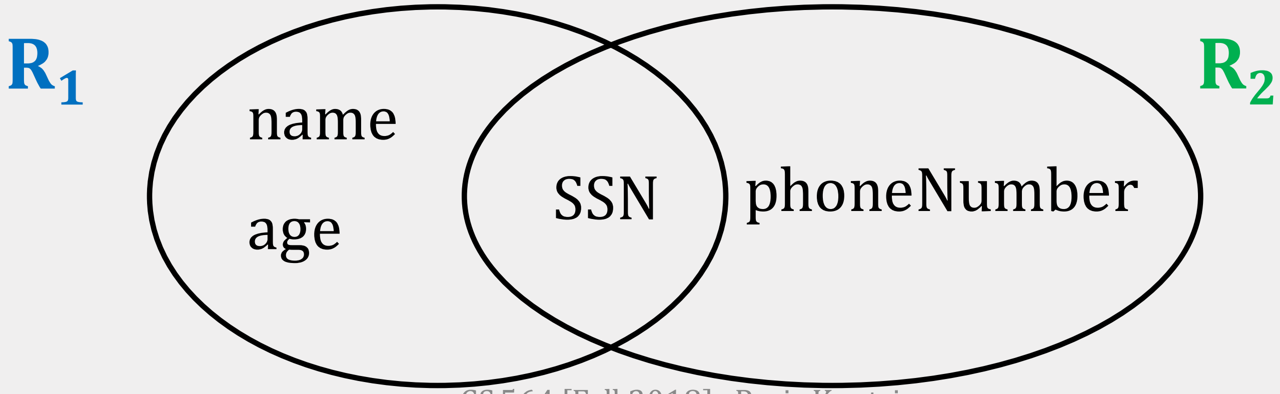


- Continue until no BCNF violations are left

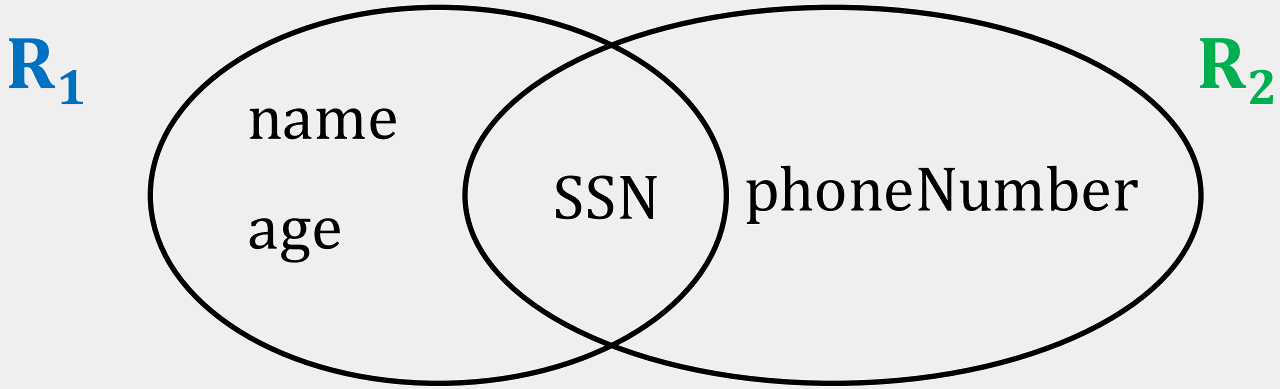
EXAMPLE

SSN	name	age	phoneNumber
934729837	Paris	24	608-374-8422
934729837	Paris	24	603-534-8399
123123645	John	30	608-321-1163
384475687	Arun	20	206-473-8221

- The FD $SSN \longrightarrow name, age$ violates BCNF
- Split into two relations $\mathbf{R_1}$, $\mathbf{R_2}$ as follows:



EXAMPLE CONT'D



$$SSN \longrightarrow name, age$$

SSN	name	age
934729837	Paris	24
123123645	John	30
384475687	Arun	20

SSN	phoneNumber
934729837	608-374-8422
934729837	603-534-8399
123123645	608-321-1163
384475687	206-473-8221

BCNF DECOMPOSITION PROPERTIES

The BCNF decomposition:

- removes certain types of redundancy
- is lossless-join
- is not always dependency preserving

BCNF IS LOSSLESS-JOIN

Example:

R(A, B, C) with $A \longrightarrow B$ decomposes into:

$\mathbf{R_1}$(A, B) and $\mathbf{R_2}$(A, C)

- The BCNF decomposition always satisfies the lossless-join criterion!

BCNF IS NOT DEPENDENCY PRESERVING

R(A, B, C)

- $A \longrightarrow B$
- $B, C \longrightarrow A$

There may not exist any BCNF decomposition that is FD preserving!

The BCNF decomposition is:

- $\mathbf{R_1}$(A, B) with FD $A \longrightarrow B$
- $\mathbf{R_2}$(A, C) with no FDs

BCNF EXAMPLE (1)

Books (author, gender, booktitle, genre, price)

- $author \longrightarrow gender$
- $booktitle \longrightarrow genre, price$

What is the candidate key?

- *(author, booktitle)* is the only one!

Is is in BCNF?

- **No**, because the left hand side of both (not trivial) FDs is not a superkey!

BCNF EXAMPLE (2)

Books (author, gender, booktitle, genre, price)

- $author \longrightarrow gender$
- $booktitle \longrightarrow genre, price$

Splitting **Books** using the FD $author \longrightarrow gender$:

- **Author** (author, gender)
  FD: $author \longrightarrow gender$ in BCNF!
- **Books2** (authos, booktitle, genre, price)
  FD: $booktitle \longrightarrow genre, price$ not in BCNF!

BCNF EXAMPLE (3)

Books (author, gender, booktitle, genre, price)

- $author \longrightarrow gender$
- $booktitle \longrightarrow genre, price$

Splitting **Books** using the FD $author \longrightarrow gender$:

- **Author** (author, gender)
  FD: $author \longrightarrow gender$ in BCNF!

- Splitting **Books2** (author, booktitle, genre, price):
  - **BookInfo** (booktitle, genre, price)
    FD: $booktitle \longrightarrow genre, price$ in BCNF!
  - **BookAuthor** (author, booktitle) in BCNF!

Third Normal Form (3NF)

3NF DEFINITION

A relation **R** is in **3NF** if whenever $X \longrightarrow A$, one of the following is true:

- $A \in X$ (trivial FD)
- X is a superkey
- A is part of some key of **R** (prime attribute)

BCNF implies 3NF !!

3NF CONT'D

- Example: **R**(A, B, C) with $A, B \longrightarrow C$ and $C \longrightarrow A$
  - is in 3NF. Why?
  - is not in BCNF. Why?

- Compromise used when BCNF not achievable: *aim for BCNF and settle for 3NF*
- Lossless-join and dependency preserving decomposition into a collection of 3NF relations is always possible!

3NF ALGORITHM

1. Apply the algorithm for BCNF decomposition until all relations are in 3NF (we can stop earlier than BCNF)
2. Compute a minimal basis *F'* of *F*
3. For each non-preserved FD $X \longrightarrow A$ in *F'*, add a new relation R(X, A)

3NF EXAMPLE (1)

Start with relation **R** (A, B, C, D) with FDs:

- $A \longrightarrow D$
- $A, B \longrightarrow C$
- $A, D \longrightarrow C$
- $B \longrightarrow C$
- $D \longrightarrow A, B$

Step 1: find a BCNF decomposition

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

3NF EXAMPLE (2)

Start with relation **R** (A, B, C, D) with FDs:

- $A \longrightarrow D$
- $A, B \longrightarrow C$
- $A, D \longrightarrow C$
- $B \longrightarrow C$
- $D \longrightarrow A, B$

Step 2: compute a minimal basis of the original set of FDs:

- $A \longrightarrow D$
- $B \longrightarrow C$
- $D \longrightarrow A$
- $D \longrightarrow B$

3NF EXAMPLE (3)

Start with relation **R** (A, B, C, D) with FDs:

- $A \longrightarrow D$
- $A, B \longrightarrow C$
- $A, D \longrightarrow C$
- $B \longrightarrow C$
- $D \longrightarrow A, B$

Step 3: add a new relation for any FD in the basis that is not satisfied:

- all the dependencies in *F'* are satisfied!
- the resulting decomposition **R1**, **R2** is also BCNF!

IS NORMALIZATION ALWAYS GOOD?

- Example: suppose A and B are always used together, but normalization says they should be in different tables
  - decomposition might produce unacceptable performance loss
- Example: data warehouses
  - huge historical DBs, rarely updated after creation
  - joins expensive or impractical