Decomposition, 3NF, BCNF

Dr. Bassam Hammo

Decomposition of a Relation Schema

- If a relation is not in a desired normal form, it can be *decomposed* into multiple relations that each are in that normal form.
- Suppose that relation R contains attributes *A1 …An.* A *decomposition* of R consists of replacing R by two or more relations such that:
  - Each new relation scheme contains a subset of the attributes of R, and
  - Every attribute of R appears as an attribute of at least one of the new relations.

Normalization Using Functional Dependencies

- When we decompose a relation schema *R* with a set of functional dependencies *F* into R_1, R_2,.., R_n we want

  - <u>Lossless-join Decomposition</u> (complete reproduction)

  - <u>No Redundancy</u> (BCNF or 3NF)

  - <u>Dependency Preservation</u>

Lossless-join Decomposition

- All attributes of an original schema (R) must appear in the decomposition (R_1, R_2):

  $$R = R_1 \cup R_2$$

- For all possible relations R_i on schema R

  $$R = \prod_{R1}(R) \bowtie \prod_{R2}(R)$$

- We Want to be able to reconstruct big (e.g. universal) relation by joining smaller ones (using natural joins)

  (i.e. $R1 \bowtie R2 = R$)

Example (Lossless-Join)

A	B	C
a_1	b_1	c_1
a_1	b_2	c_2

decompose

A	C
a_1	c_1
a_1	c_2

B	C
b_1	c_1
b_2	c_2

join

A	B	C
a_1	b_1	c_1
a_1	b_2	c_2

Example (Lossy-Join)

A	B	C
a_1	b_1	c_1
a_1	b_2	c_2

decompose

A	B
a_1	b_1
a_1	b_2

A	C
a_1	c_1
a_1	c_2

join

A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_1	b_2	c_1
a_1	b_2	c_2

Testing for Lossless-Join Decomposition

- Rule: A decomposition of *R* into *(R1, R2)* is *lossless,* iff:

$$\boldsymbol{R1 \cap R2 \rightarrow R1} \qquad \textit{or}$$

$$\boldsymbol{R1 \cap R2 \rightarrow R2}$$

in *F+*.

Exercise: Lossless-join Decomposition

$R = \{A,B,C,D,E\}$.

$F = \{A \rightarrow BC,\ CD \rightarrow E,\ B \rightarrow D,\ E \rightarrow A\}$.

Is the following decomposition a lossless join?

1. $R1 = \{A,B,C\}$, $R2 = \{A,D,E\}$

   Since $R1 \cap R2 = A$, and A is a key for R1, the decomposition is lossless join.

2. $R1 = \{A,B,C\}$, $R2 = \{C,D,E\}$

   Since $R1 \cap R2 = C$, and C is not a key for R1 or R2, the decomposition is not lossless join.

Dependency Preserving Decomposition

- The decomposition of a relation scheme R with FDs F is a set of tables (fragments) R_i with FDs F_i
- F_i is the subset of dependencies in F+ (the closure of F) that include only attributes in R_i.
- The decomposition is **dependency preserving** iff

$$(\cup_i F_i)+ = F+$$

- In other words: we want to minimize the cost of global integrity constraints based on FD's (i.e. avoid big joins in assertions)

$$(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$$

Exercise: Non-Dependency Preserving Decomposition

R = (A, B, C), F = {$A \rightarrow B$, $B \rightarrow C$, $\mathbf{A \rightarrow C}$}

Key: **A**

Assume there is a dependency $\mathbf{B \rightarrow C}$, where the LHS is not the key, meaning that there can be considerable redundancy in R.

Solution: Break it in two tables **R1**(A,B), **R2**(A,C)

Exercise: Non-Dependency Preserving Decomposition

The decomposition is **lossless** because the common attribute **A** is a key for R1 (and R2)

The decomposition is not dependency preserving because:

F1=$\{A \rightarrow B\}$,

F2=$\{A \rightarrow C\}$ and $(F1 \cup F2)+ \neq F+$

But, we lost the FD $\{B \rightarrow C\}$

- In practical terms, each FD is implemented as a constraint or assertion, which it is checked when there are updates. In the above example, in order to find violations, we have to join R1 and R2. Which can be very expensive.

Exercise: Dependency Preserving Decomposition

R = (A, B, C), F = {A→B, B→C, **A→C**} Key: **A**

Solution: Break it in two tables **R1**(A,B), **R2**(B,C)

- The decomposition is **lossless** because the common attribute **B** is a key for **R2**
- The decomposition is **dependency preserving** because F1={A→B}, F2={B→C} and (F1∪F2)+=F+
- Violations can be found by inspecting the individual tables, without performing a join.
- What about *A* → *C ?*

  If we can check *A* → *B*, and *B* → *C*, *A* → *C* is implied.

Exercise 2 : FD-Preserving Decomposition

$R = \{A,B,C,D,E\}$.

$F = \{A \rightarrow BC,\ CD \rightarrow E,\ B \rightarrow D,\ E \rightarrow A\}$

$R1 = \{A,B,C\}$, $R2 = \{A,D,E\}$

Is the above decomposition dependency-preserving?

No.

$CD \rightarrow E$ and **$B \rightarrow D$** are lost.

Third Normal Form Decomposition

Third Normal Form

3NF: A schema *R* is in third normal form (3NF) if

for all FD $\alpha \rightarrow \beta$ in F^+, at least one of the following holds:

(1) $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$).

(2) α is a superkey for *R*.

(3) Each attribute *A* in $\beta - \alpha$ is contained in a candidate key for *R* *(prime)*.

- **The decomposition is both lossless-join and dependency-preserving**

Third Normal Form

- A relational schema **R** is in 3NF if for every FD $X \rightarrow A$ associated with **R** either:
  - $A \subseteq X$ (i.e., the FD is trivial) or
  - X is a superkey of **R** or
  - A is part of some key (not just superkey!)
- 3NF weaker than BCNF (every schema that is in BCNF is also in 3NF)

Third Normal Form

- Compromise - Not all redundancy removed, but dependency-preserving decompositions are always possible
- 3NF decomposition is based on the concept of *minimal cover of a set of FDs*

Decomposition into 3NF

- **Decomposition**
  - Given: relation R, set F of functional dependencies
  - Find: decomposition of R into a set of 3NF relation R_i
  - Algorithm:

(1) Eliminate redundant FDs, resulting in a canonical cover Fc of F
(2) Create a relation $R_i = XY$ for each FD $X \rightarrow Y$ in Fc
(3) If the key K of R does not occur in any relation R_i, create one more relation R_i=K

Computing Minimal Cover

- **step 1**: RHS of each FD is a single attribute.
- **step 2**: Eliminate unnecessary attributes from LHS.
  - Algorithm: If FD $XB \rightarrow A \in T$ (where B and A are single attributes) and $X \rightarrow A$ is entailed by T, then B was unnecessary
- **step 3**: Delete unnecessary FDs from T
  - Algorithm: If T - $\{f\}$ entails f, then f is unnecessary.
    - If f is $X \rightarrow A$ then check if A $\in X^{+}_{T-\{f\}}$

Example

- $\{A \to B, ABCD \to E, EF \to GH, ACDF \to EG\}$
- Make RHS a single attribute: $\{A \to B, ABCD \to E, EF \to G, EF \to H, ACDF \to E, ACDF \to G\}$
- Minimize LHS: $ACD \to E$ instead of $ABCD \to E$
- Eliminate redundant FDs
  - Can $ACDF \to G$ be removed?
  - Can $ACDF \to E$ be removed?
- Final answer: $\{A \to B, ACD \to E, EF \to G, EF \to H\}$

Example

- Relation: R=CSJDPQV
- FDs: C→CSJDPQV, SD→P, JP→C,J→S
- Find minimal cover: {C→J, C→D, C→Q, C→V, JP→C, J→S, SD→P}
- Combine LHS: {C→JDQV, JP→C, J→S, SD→P}
- New relations: CJDQV, JPC, JS, SDP
- Since CJDQV is a superkey we are done!

BCNF Normal Form Decomposition

Boyce-Codd Normal Form

- **BCNF:** A schema R is in BCNF with respect to a set F of functional dependencies, if for all functional dependencies in
- F^+ of the form $\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

  (1) $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)

  (2) α is a superkey for R

- In other words, the left part of any non-trivial dependency must be a superkey.
- If we do not have redundancy in F, then for each $\alpha \rightarrow \beta$, α must be a candidate key.
- **The decomposition is lossless-join but may not be dependency-preserving**

Decomposing into BCNF Schemas

- For all dependencies *A* → *B* in *F+*, check if *A* is a superkey
  - By using attribute closure
- If not, then
  - Choose a dependency in F+ that breaks the BCNF rules, say A → B
  - Create R1 = A B
  - Create R2 = A (R – B – A)
  - Note that: R1 ∩ R2 = A and A → AB (= R1), so this is lossless decomposition
- Repeat for *R1, and R2*
  - By defining F1+ to be all dependencies in F that contain only attributes in R1
  - Similarly F2+

BCNF Decomposition

- Suppose **R** = ($R; F$) is not in BCNF

- In general: Let $X \rightarrow Y \in F$ be a violating FD
  - Decompose into XY and $(R - Y) \cup X$

If either R-A or XA is not in BCNF, decompose them further recursively

BCNF Example #1

R = (A, B, C)

F = {A → B, B → C}

Candidate keys = {A}

BCNF? = No. **B → C** violates.

B → C

R1 = (B, C)

F1 = {B → C}

Candidate keys = {B}

BCNF? = true

R2 = (A, B)

F2 = {A → B}

Candidate keys = {A}

BCNF? = true

BCNF Example #2

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

F = {A → B, BC → D}

Candidate keys = {ACE}

BCNF = Violated by {A → B, BC → D} etc…

From A → B and BC → D by pseudo-transitivity

A → B

R1 = (A, B)

F1 = {A → B}

Candidate keys = {A}

BCNF = true

R2 = (A, C, D, E)

F2 = {AC → D}

Candidate keys = {ACE}

BCNF = false (AC → D)

AC → D

R3 = (A, C, D)

F3 = {AC → D}

Candidate keys = {AC}

BCNF = true

R4 = (A, C, E)

F4 = {} [[only trivial]]

Candidate keys = {ACE}

BCNF = true

Dependency preservation ???

We can check:

A → B (R1), AC → D (R3),

but we lost BC → D

So this is not a dependency -preserving decomposition

Example #3

R = (A, B, C, D, E)
F = {A $\rightarrow$ B, BC $\rightarrow$ D}
Candidate keys = {ACE}
BCNF = Violated by {A $\rightarrow$ B, BC $\rightarrow$ D} etc…

BC $\rightarrow$ D

R1 = (B, C, D)
F1 = {BC $\rightarrow$ D}
Candidate keys = {BC}
BCNF = true

R2 = (B, C, A, E)
F2 = {A $\rightarrow$ B}
Candidate keys = {ACE}
BCNF = false (A $\rightarrow$ B)

A $\rightarrow$ B

R3 = (A, B)
F3 = {A $\rightarrow$ B}
Candidate keys = {A}
BCNF = true

R4 = (A, C, E)
F4 = {} [[only trivial]]
Candidate keys = {ACE}
BCNF = true

Dependency preservation ???
We can check:
BC $\rightarrow$ D (R1), A $\rightarrow$ B (R3),
Dependency-preserving decomposition

Example #4

R = (A, B, C, D, E, H)
F = {A → BC, E → HA}
Candidate keys = {DE}
BCNF = Violated by {A → BC} etc…

A → BC

R1 = (A, B, C)
F1 = {A → BC}
Candidate keys = {A}
BCNF = true

R2 = (A, D, E, H)
F2 = {E → HA}
Candidate keys = {DE}
BCNF = false (E → HA)

E → HA

R3 = (E, H, A)
F3 = {E → HA}
Candidate keys = {E}
BCNF = true

R4 = (ED)
F4 = {} [[only trivial]]
Candidate keys = {DE}
BCNF = true

Dependency preservation ???
We can check:
A → BC (R1), E → HA (R3),
Dependency-preserving decomposition

More Examples

Example #5: BCNF Decomposition

- Relation: R=CSJDPQV
- FDs: C→CSJDPQV, SD→P, JP→C, J→S
- JP→C is OK, since JP is a superkey
- SD→P is a violating FD
- Decompose into R1=CSJDQV and R2=SDP
- J→S is still a violation in R1
- Decompose R1: CJDQV and JS
- Final set: CJDQV, JS, SDP
- Order matters: what happens if we use J→S first?

Exercise 3

R =(A, B, C, D).

F = $\{C \rightarrow D, C \rightarrow A, B \rightarrow C\}$.

Question 1: Identify all candidate keys for R.

Question 2: Identify the best normal form that R satisfies.

Question 3: Decompose R into a set of BCNF relations.

Question 4: Decompose R into a set of 3NF relations.

Exercise 3 Solution

R =(A, B, C, D).

F = {C→D, C→A, B→C}.

Question 1: Identify all candidate keys for R.

$$
\begin{aligned}
B^+ &= B &\quad (B \to B)\\
&= BC &\quad (B \to C)\\
&= BCD &\quad (C \to D)\\
&= ABCD &\quad (C \to A)
\end{aligned}
$$

so the candidate key is B.

B is the ONLY candidate key, because nothing determines B:

There is no rule that can produce B, except B →B.

Exercise 3 Solution

R =(A, B, C, D).

F = $\{C \rightarrow D, C \rightarrow A, B \rightarrow C\}$.

Question 2: Identify the best normal form that R satisfies.

R is not 3NF, because:

$C \rightarrow D$ causes a violation,

- $C \rightarrow D$ is non-trivial ($\{D\} \not\subset \{C\}$).
- C is not a superkey.
- D is not part of any candidate key.

$C \rightarrow A$ causes a violation

- Similar to above

$B \rightarrow C$ causes no violation

Since R is not 3NF, it is not BCNF either.

Exercise 3 Solution

R =(A, B, C, D).

F = {C→D, C→A, B→C}.

Question 3: Decompose R into a set of BCNF relations

(1)C→D and C→A both cause violations of BCNF.

Take C→D: decompose R to R_1= {A, B, C} , R_2={C, D}.

(2)Now check for violations in R_1 and R_2. (Actually, using F^+)

R_1 still violates BCNF because of C→A.

Decompose R_1 to R_{11} = {B, C} R_{12} = {C, A}.

Final decomposition: R_2 = {C, D}, R_{11} = {B, C}, R_{12} = {C, A}.

No more violations: Done!

Exercise 3 Solution

$R = (A, B, C, D)$.

$F = \{C \rightarrow D, C \rightarrow A, B \rightarrow C\}$.

Question 4: Decompose R into a set of 3NF relations.

The canonical cover is $F_c = \{C \rightarrow DA, B \rightarrow C\}$.

For each functional dependency in F_c we create a table:

$R_1 = \{C, D, A\}$, $R_2 = \{B, C\}$.

The table R_2 contains the candidate key for R – we done.

Exercise 4

R = (A, B, C, D)

F = {AB→C, AB→D, C→A, D→B}

1. Is R in 3NF, why? If it is not, decompose it into 3NF

2. Is R in BCNF, why? If it is not, decompose it into BCNF

Exercise 4 Solution

R = (A, B, C, D)

F = {AB→C, AB→D, C→A, D→B}

1. Is R in 3NF, why? If it is not, decompose it into 3NF

Yes.

Find all the Candidate Keys:

AB, BC, CD, AD

Check all FDs in F for 3NF condition

2. Is R in BCNF, why? If it is not, decompose it into BCNF

No. Because for C→A, C is not a superkey. Similar for D→B

R1 = {C, D}, R2 = {A, C}, R3 = {B, D}

Summary

- Step 1: BCNF is a good form for relation
  - If a relation is in BCNF, it is free of redundancies that can be detected using FDs.
- Step 2 : If a relation is not in BCNF, we can try to decompose it into a collection of BCNF relations.
- Step 3: If a lossless-join dependency-preserving decomposition into BCNF is not possible (or unsuitable given typical queries), consider decomposition into 3NF.
- Note: Decompositions should be carried out while keeping *performance requirements* in mind.