Schema Refinement & Normalization Theory

Normal Forms 2

BCNF and Dependency Preservation

- In general, there may not be a dependency preserving decomposition into BCNF.
- Example: schema CSZ (city, street_name, zip_code) with FDs: CS $\rightarrow$ Z, Z $\rightarrow$ C

  $$(city,\ street\_name) \rightarrow zip\_code$$
  $$zip\_code \rightarrow city$$

- Can't decompose while ***preserving*** CS $\rightarrow$ Z, but CSZ is not in BCNF.

Example Regarding Dependency Preservation

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C)$
  - Can be decomposed in two different ways
- $R_1 = (A, B), \quad R_2 = (B, C)$
  - Lossless-join decomposition:

    $$R_1 \cap R_2 = \{B\} \text{ and } B \rightarrow BC$$
  - Dependency preserving
- $R_1 = (A, B), \quad R_2 = (A, C)$
  - Lossless-join decomposition:

    $$R_1 \cap R_2 = \{A\} \text{ and } A \rightarrow \mathrm{A}B$$
  - Not dependency preserving
    (cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

Dependency Preserving Decomposition

- Consider CSJDPQV, C is key, JP → C and SD → P.
  - BCNF decomposition: CSJDQV and SDP
  - Problem: Checking JP → C requires a join!
- Dependency preserving decomposition (Intuitive):
  - If R is decomposed into X, Y and Z, and we enforce the FDs that hold on X, on Y and on Z, then all FDs that were given to hold on R must also hold. *(Avoids Problem (3))*

What FD on a decomposition?

- *Projection (or **restriction**) of set of FDs F*: If R is decomposed into X, ... the projection (also referred to as restriction) of F onto X (denoted F_X) is the set of FDs $U \rightarrow V$ in F^+ (*closure of F*) such that U, V are in X.

Dependency Preserving Decompositions

- Decomposition of R into X and Y is *dependency preserving* if $(F_X \cup F_Y)^+ = F^+$
  - i.e., if we consider only dependencies in the closure F^+ that can be checked in X without considering Y, and in Y without considering X, these imply all dependencies in F^+.
- Important to consider F^+, not F, in this definition:
  - ABC, $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow A$, decomposed into AB and BC.
  - Is this dependency preserving? Is $C \rightarrow A$ preserved?????
- Dependency preserving does not imply lossless join:
  - ABC, $A \rightarrow B$, decomposed into AB and BC.
- And vice-versa!
- Expensive since we have to compute F^+ and $(F_1 \cup F_2 \cup \ldots \cup F_n)^+$

(Efficient) Testing for Dependency Preservation

- To check if a dependency $X \rightarrow Y$ is preserved in a decomposition of R into R_1, R_2, …, R_n we apply the following test (with attribute closure done with respect to F)
  - *result* = X
    while (changes to *result*) do
    for each R_i in the decomposition
    $t = (result \cap R_i)^+ \cap R_i$
    $result = result \cup t$
  - If *result* contains all attributes in Y, then the functional dependency $X \rightarrow Y$ is preserved.
- Apply the test on all dependencies in F to check if a decomposition is dependency preserving
- This procedure takes polynomial time.

- $R(A, B, C)$, $F = \{A \rightarrow B,\ B \rightarrow C,\ C \rightarrow A\}$, decomposed into AB and BC.
- The only FD we need to check is $C \rightarrow A$.
- Result = C
- Check AB:

  $T = (\text{Result} \cap AB)^{+} \cap AB$
  $= (C \cap AB)+ \cap AB = \{\}$
  Result = C
- Check BC:

  $T = (\text{Result} \cap BC)^{+} \cap BC$
  $= (C \cap BC)+ \cap AB = C+ \cap BC = ABC \cap BC = BC$
  Result = $BC \cup C = BC$
- Check AB again

  $T = (\text{Result} \cap AB)^{+} \cap AB$
  $= (BC \cap AB)+ \cap AB = (BC)+ \cap AB = ABC \cap AB = AB$
  Result = $BC \cup AB = ABC$
  (Result contains A, so dependency preserving!)

Another example

- Assume CSJDPQV is decomposed into

  SDP, JS, CJDQV

  It is not dependency preserving

  w.r.t. the FDs: $JP \rightarrow C$, $SD \rightarrow P$ and $J \rightarrow S$.
- However, it is a lossless join decomposition.
- In this case, adding JPC to the collection of relations gives us a dependency preserving decomposition.
- JPC tuples stored only for checking FD!

Summary of BCNF

- If a relation is in BCNF, it is free of redundancies that can be detected using FDs. Thus, trying to ensure that all relations are in BCNF is a good heuristic.
- If a relation is not in BCNF, we can try to decompose it into a collection of BCNF relations.
  - It is always possible to decompose a relation into a set of relations that are in BCNF such that:
    - the decomposition is lossless
    - it may not be possible to preserve dependencies.

Next: Third Normal Form

- There are some situations where
  - BCNF is not dependency preserving, and
  - efficient checking for FD violation on updates is important
- Solution: define a weaker normal form, called Third Normal Form (3NF)
  - Allows some redundancy (with resultant problems; we will see examples later)
  - But functional dependencies can be checked on individual relations without computing a join.
  - There is always a lossless-join, dependency-preserving decomposition into 3NF.

Third Normal Form (3NF)

- If R is in BCNF, obviously in 3NF.
- If R is in 3NF, some redundancy is possible. It is a compromise, used when BCNF not achievable (e.g., no "good" decomposition, or performance considerations).
  - *Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible.*

3NF

- Relation R with FDs F is in 3NF if, for each FD $X \rightarrow A$ ($X \subseteq R$ and $A \subseteq R$) in F, one of the following statements is true:
  - $A \subseteq X$ (trivial FD), <u>or</u>
  - X is a superkey, <u>or</u>
  - (A – X) is part of some <u>candidate key</u> for R

If one of these two is satisfied for ALL FDs, then R is in BCNF

Not just superkey! (why not?)

What Does 3NF Achieve?

- If 3NF is violated by $X \rightarrow A$, one of the following holds:
  - X is a subset of some key K (partial redundancy)
    - We store (X, A) pairs redundantly.
  - X is not a proper subset of any key.
    - There is a chain of FDs $K \rightarrow X \rightarrow A$, which means that we cannot associate an X value with a K value unless we also associate an A value with an X value.
- But: even if a relation is in 3NF, these problems could arise.
  - e.g., Reserves SBDC (sid, bid, date, credit_card). Keys are SBD, CBD. FD = $\{S \rightarrow C, \ C \rightarrow S\}$. R is in 3NF, but for each reservation of sailor S, same (S, C) pair is stored.
- Thus, 3NF is indeed a compromise relative to BCNF.

Decomposition into 3NF

- Obviously, the algorithm for lossless join decomp into BCNF can be used to obtain a lossless join decomp into 3NF (typically, can stop earlier).
- To ensure dependency preservation, one idea:
  - If $X \rightarrow Y$ is not preserved, add relation XY.
  - Problem is that XY may violate 3NF!
- Refinement: Instead of the given set of FDs F, use a *canonical cover* or a *minimal cover for F*.

Minimal Cover for a Set of FDs

- *Minimal cover* G for a set of FDs F:
  - Closure of F = closure of G.
  - Right hand side of each FD in G is a single attribute.
  - If we modify G by deleting an FD or by deleting attributes from an FD in G, the closure changes.
- Intuitively, every FD in G is needed, and "*as small as possible*" in order to get the same closure as F.

The textbook uses **canonical cover**, which does not have the second requirement. Instead, canonical cover requires that each left-hand-side of dependencies is unique.

Obtaining Minimal Cover

- Step 1: Put the FDs in a standard form (i.e. right-hand side should contain only single attribute)
- Step 2: Minimize the left side of each FD by eliminating any extraneous attributes.
- Step 3: Delete redundant FDs

- Find minimal cover for F = {ABH $\rightarrow$ CK, A $\rightarrow$ D, C $\rightarrow$ E, BGH $\rightarrow$ L, L $\rightarrow$ AD, E $\rightarrow$ L, BH $\rightarrow$ E}

- Step 1: Make RHS of each FD into a single attribute:

$F = \{ABH \rightarrow C, ABH \rightarrow K, A \rightarrow D, C \rightarrow E,$
$BGH \rightarrow L, L \rightarrow A, L \rightarrow D, E \rightarrow L, BH \rightarrow E\}$

- $F = \{ABH \rightarrow C, ABH \rightarrow K, A \rightarrow D, C \rightarrow E, BGH \rightarrow L, L \rightarrow A, L \rightarrow D, E \rightarrow L, BH \rightarrow E\}$

- Step 2: Eliminate extraneous (redundant) attributes from LHS, e.g. Can an attribute be deleted from $ABH \rightarrow C$?
  - Compute (AB)+, (BH)+, (AH)+ and see if any of them contains C. (Why?)
  - (AB)+ = ABD, (BH)+ = ABCDEHKL, (AH)+ = ADH. Since $C \in$ (BH)+, $BH \rightarrow C$ is entailed by F. So A is redundant in $ABH \rightarrow C$. Similarly, A is also redundant in $ABH \rightarrow K$. Check further to see if B or H is redundant as well.
  - Similarly, for $BGH \rightarrow L$, G is redundant since $L \in$ (BH)+.
  - $F = \{BH \rightarrow C, BH \rightarrow K, A \rightarrow D, C \rightarrow E, BH \rightarrow L, L \rightarrow A, L \rightarrow D, E \rightarrow L, BH \rightarrow E\}$

- F = {BH → C, BH → K, A → D, C → E, BH → L, L → A, L → D, E → L, BH → E}

- Step 3: Delete redundant FDs from F.

  – If F – {*f*} infers *f*, then *f* is redundant, i.e. if *f* is X → A, then check if X+ using F – *f* still contains A. If it does, then it means X → A can be inferred by other FDs.
  – e.g. For BH → L, (BH)+ (not using BH → L) = ACDEKL, which contains L. This means BH → L can be inferred by other FDs, so it's a redundant FD.
  – In fact, BH → L can be inferred by BH → E, E → L.
  – Check other FDs using the same algorithm.

- Note: the order of Step 2 and Step 3 should not be exchanged.

What to do with Minimal Cover?

- After obtaining the minimal cover, for each FD *X*→ *A* in the minimal cover that is not preserved, create a table consisting of XA (so we can check dependency in this new table, i.e. dependency is preserved).
- Why is this new table guaranteed to be in 3NF (whereas if we created the new table from F, it might not?)
  – Since X → A is in the minimal cover, Y → A does not hold for any Y that is a strict subset of X.
    - So X is a key for XA (satisfies condition #2)
    - If any other dependencies hold over XA, the right side can involve only attributes in X because A is a single attribute (satisfies condition #3).

Comparison of BCNF and 3NF

- It is always possible to decompose a relation into a set of relations that are in 3NF such that:
  - the decomposition is lossless
  - the dependencies are preserved
- It is always possible to decompose a relation into a set of relations that are in BCNF such that:
  - the decomposition is lossless
  - it may not be possible to preserve dependencies.

Normalization Review

- Identify all FD's in F^+
- Identify candidate keys
- Identify (strongest, or specific) normal forms
  - BCNF, 3NF
- Schema decomposition
  - When to decompose
  - How to check if a decomposition is lossless-join and/or dependency preserving
    - Use projection of F^+ to check for dependency preservation
  - Decompose into:
    - Lossless-join
    - Dependency preserving
      - Use minimal cover

Normalization Theory - Practice Questions

Example

A	B	C
1	1	2
1	1	3
2	2	3
2	2	2

FDs with A as the left side:	**Satisfied by the relation?**
$A \rightarrow A$	Yes (trivial FD)
$A \rightarrow B$	Yes
$A \rightarrow C$	No: tuples 1&2
$AB \rightarrow A$	Yes (trivial FD)
$AC \rightarrow B$	Yes

Example

Let F={ A $\rightarrow$ BC, B $\rightarrow$C }. Is C $\rightarrow$AB in F^+?

Answer: No. Either of the following 2 reasons is ok:

Reason 1) C^+=C, and does not include AB.

Reason 2) We can find a relation instance such that it satisfies F but does not satisfy C $\rightarrow$ AB.

A	B	C
1	1	2
2	1	2

List all the <u>non-trivial</u> FDs in F^+

- Given $F=\{A \rightarrow B, B \rightarrow C\}$. Compute F^+ (with attributes A, B, C).

	A	B	C	AB	AC	BC	ABC
A	√	√	√	√	√	√	√
B		√	√			√	
C			√				
AB	√	√	√	√	√	√	√
AC	√	√	√	√	√	√	√
BC		√	√			√	
ABC	√	√	√	√	√	√	√

Attribute closure
$A^+=ABC$
$B^+=BC$
$C^+=C$
$AB^+=ABC$
$AC^+=ABC$
$BC^+=BC$
$ABC^+=ABC$

Example

- Given F={ A → B, B → C}. Find a relation that satisfies F:

A	B	C
1	1	2
2	1	2

- Given F={ A → B, B → C}. Find a relation that satisfies F but does not satisfy B → A. The above example suffices.
- Can you find an instance that satisfies F but not A → C? No. Because A → C is in F^+

Examples

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

Candidate key: ACE. How do we know?

Intuitively,
-A is not determined by any other attributes (like E), and A has to be in a candidate key (because a candidate key has to determine all the attributes).
- Now if A is in a candidate key, B cannot be in the same candidate key, since we can drop B from the candidate without losing the property of being a “key”.
- So B cannot be in a candidate key
- Same reasoning apply to others attributes.

Example

$R(A, B, C, D, E)$,
$F = \{A \rightarrow B, C \rightarrow D\}$ [Same as previous]

Which normal form?

Not in BCNF. This is the case where all attributes in the FDs appear in R. We consider A, and C to see if either is a superkey of not. Obviously, neither A nor C is a superkey, and hence R is not in BCNF. More precisely, we have $A \rightarrow B$ is in F^+ and non-trivial, but A is not a superkey of R.

Example

R(A, B, C, D, E)
F = {A → B, C → D} [Same as previous]

Which normal form?

We already know that it's not in BCNF.
Not in 3NF either. We have A → B is in F^+ and non-trivial, but A is not a superkey of R. Furthermore, B is not in any candidate key (since the only candidate key is ACE).

Example

- $R(A,B,F)$, $F = \{AC \rightarrow E, B \rightarrow F\}$.
- Candidate key? AB
- BCNF? No, because of $B \rightarrow F$ (B is not a superkey).
- 3NF? No, because of $B \rightarrow F$ (F is not part of a candidate key).

Example

- R(D, C, H, G), F = {A → I, I → A}
- Candidate key? DCHG
- BCNF? Yes
- 3NF? Yes

Example

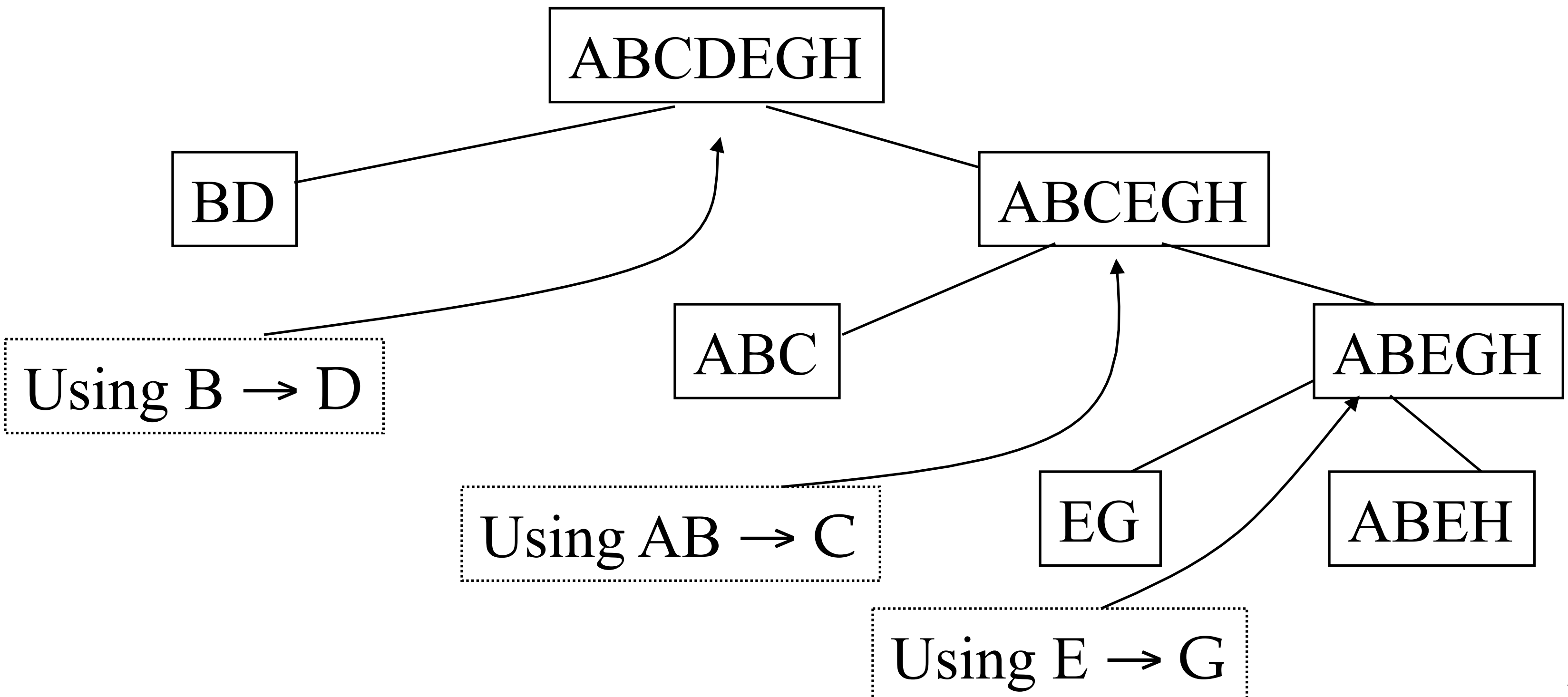
- R(A, B, C, D, E, G, H)
  F={AB → C, AC → B, B → D, BC → A, E → G}

- Candidate keys?
  - H has to be in all candidate keys
  - E has to be in all candidate keys
  - G cannot be in any candidate key (since E is in all candidate keys already).
  - Since AB → C, AC → B and BC → A, we know no candidate key can have ABC together.
  - AEH, BEH, CEH are not superkeys.
  - Try ABEH, ACEH, BCEH. They are all superkeys. And we know they are all candidate keys (since above properties)
  - These are the only candidate keys: (1) each candidate key either contains A, or B, or C since no attributes other than A,B,C determine A, B, C, and (2) if a candidate key contains A, then it must contain either B, or C, and so on.

Example

- Same as previous
- Not in BCNF, not in 3NF
- Decomposition:

R(A, B, C, D, E, G, H)
F={AB → C, AC → B, B → D, BC → A, E → G}



Example

- R(A, B, C, D, E, G, H)

  F={AB → C, AC → B, B → D, BC → A, E → G}
- Decomposition: BD, ABC, EG, ABEH
- Why good decomposition?
  - They are all in BCNF
  - Lossless-join decomposition
    - How do you know this if you don't know how R was decomposed?
  - All dependencies are preserved.

Example

- R(A, B, D, E) decomposed into R1(A, B, D), R2(A, B, E)
- F={AB → DE}
- It is a dependency preserving decomposition!
  - AB → D can be checked in R1
  - AB → E can be checked in R2
  - {AB → DE} is equivalent to {AB → D, AB → E}