CZ2007 Introduction to Database Systems (Week 4)

Topic 4: Third Normal Form (1)

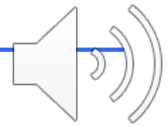
Dr. Ng Wee Keong
Associate Professor

This Lecture

- 3NF 
- 3NF Decomposition

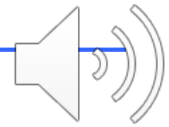
Third Normal Form (3NF)

- A relaxation of BCNF that
  - Is less strict
  - Allows decompositions that always preserve functional dependencies



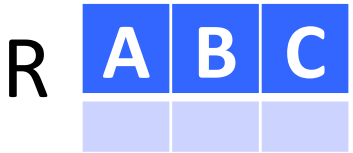
First Normal Form (1NF)

- Key-attribute: An attribute in a multi-attribute key
  - ABC is key, then A, B, or C is key attribute
  - Key-attribute(s) = Partial key or part of a key; AB is partial key
- 1NF: All attributes have atomic values
- 2NF: Every non-key attribute is dependent on the whole of every candidate key
  - But, may still have (non-key-attribute X) → (non-key-attribute Y) in the relation



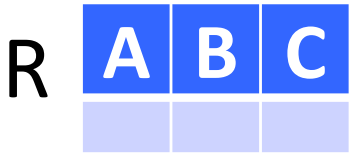
Second Normal Form (2NF)

- 2NF: Every non-key attribute is dependent on the whole of **EVERY** candidate key
  - But, may still have (non-key-attribute X) → (non-key-attribute Y) in the relation
- Example:
  - Given FDs: A→B, A→C, B→C
  - Key: {A}
  - Non-key attributes: B, C
  - A→B means B depends on key
  - A→C means C depends on key
  - But B→C means non-key C depends on non-key B
  - So R is in 2NF



Third Normal Form (3NF)

- Definition: A table satisfies 3NF, if and only if for every non-trivial X→Y
  - Either X contains a key (Y depends on key)
  - Or each attribute in Y is contained in a key (Y depends on key)
- Example:
  - Given FDs: C→B, AB→C, BC→C
  - Keys: {AB}, {AC}
  - AB→C is OK, since AB is a key of R
  - C→B is OK, since B is part of AB
  - We ignore BC→C since it is trivial
  - So R is in 3NF

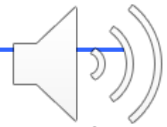


Third Normal Form (3NF)

- Definition: A table satisfies 3NF, if and only if for every non-trivial X→Y
  - Either X contains a key (X depends on key)
  - Or each attribute in Y is contained in a key (Y depends on key)
- Another Example:
  - Given FDs: A→B, B→C
  - Keys: {A}
  - A→B is OK, since A is a key of R
  - B→C is not OK, since B doesn't contain a key, and C is NOT in a key of R
  - So R is NOT in 3NF

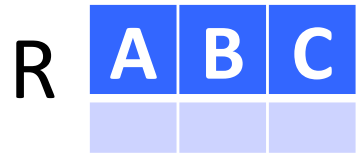
3NF, BCNF

- 3NF: All attributes (key-attributes or non-key-attributes) depends (directly or transitively) on candidate keys
  - But candidate keys may have **overlapping** attributes
  - May result in key-attribute(s) of one key depends on key-attribute(s) of another key
- BCNF: In all dependencies (FDs), LHS must contain key (cannot depend on partial key)



3NF, BCNF

- 3NF: All attributes (key-attributes or non-key-attributes) depends (directly or transitively) on candidate keys
  - But candidate keys may have **overlapping** attributes
  - May result in key-attribute(s) of one key depends on key-attribute(s) of another key
- Example:
  - Given FDs: C→B, AB→C, BC→C
  - Keys: {AB}, {AC}
  - Key-attributes: A, B, C
  - R is in 3NF but C→B means partial key B depends on partial key C



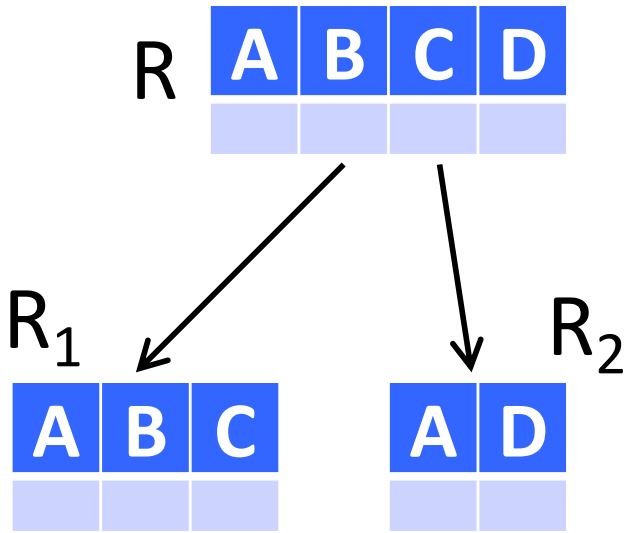
- BCNF: In all dependencies (FDs), LHS must contain key (cannot depend on partial key)

This Lecture

- 3NF
- 3NF Decomposition

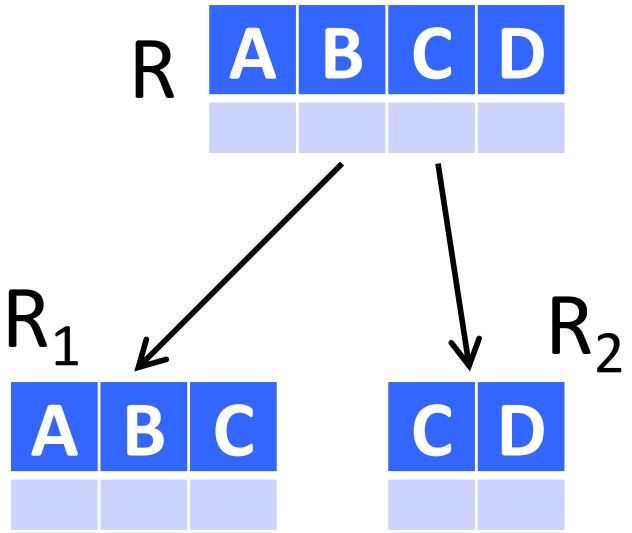
3NF Decomposition

- Given: A table NOT in 3NF
- Objective: Decompose it into smaller tables that are in 3NF
- Example
  - Given: R(A, B, C, D)
  - FDs: AB→C, C→B, A→D
  - Keys: {AB}, {AC}
  - R is not in 3NF, due to A→D
  - 3NF decomposition of R: R_1(A, B, C), R_2(A, D)



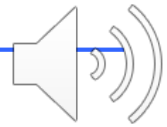
3NF Decomposition Algorithm

- Given: A table R, and a set S of FDs
  - e.g., R(A, B, C, D)
    S = {A→BD, AB→C, C→D, BC→D}
- Step 1: Derive a minimal basis of S
  - e.g., a minimal basis of S is
    {A→B, A→C, C→D}
- Step 2: In the minimal basis, combine the FDs whose left hand sides are the same
  - e.g., after combining A→B and A→C,
    we have {A→BC, C→D}
- Step 3: Create a table for each FD remained
  - $R_1(A, B, C)$, $R_2(C, D)$
- Step 4: If none of the tables contain a key of the original table R, create a table that contains a key of R
- Step 5: Remove redundant tables



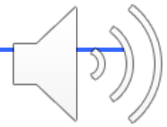
Minimal Basis

- Given a set S of FDs, the minimal basis of S is a simplified version of S
- Previous example:
  - S = {A→BD, AB→C, C→D, BC→D}
  - A minimal basis: {A→B, A→C, C→D}
- How simplified?
- Three conditions.
- Condition 1: For any FD in the minimal basis, its right hand side has only one attribute.
- Example in S: A→BD does not satisfy this condition
- That is why A→BD is not in the minimal basis



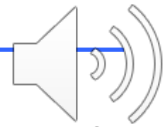
Minimal Basis

- Previous example:
  - S = {A→BD, AB→C, C→D, BC→D}
  - A minimal basis: {A→B, A→C, C→D}
- Condition 2: No FD in the minimal basis is redundant.
- That is, no FD in the minimal basis can be derived from the other FDs.
- Example in S: BC→D can be derived from C→D
- That is why BC→D is not in the minimal basis



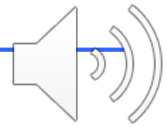
Minimal Basis

- Previous example:
  - S = {A→BD, AB→C, C→D, BC→D}
  - A minimal basis: {A→B, A→C, C→D}
- Condition 3: For each FD in the minimal basis, none of the attributes on the left hand side is redundant
- That is, for any FD in the minimal basis, if we remove an attribute from the left hand side, then the resulting FD is a new FD that cannot be derived from the original set of FDs
- Example:
  - S contains AB→C
  - If we remove B from the left hand side, we have A→C
  - A→C can be derived from S, as $\{A\}^+ = \{ABDC\}$
  - This indicates that A→C is "hidden" in S
  - Hence, we can replace AB→C with A→C, as A→C is "simpler"
  - This is why AB→C is not in the minimal basis



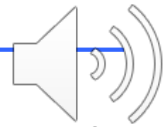
Algorithm for Minimal Basis

- Given: a set S of FDs
- Example: S = {A→BD, AB→C, C→D, BC→D}
- Step 1: Transform the FDs, so that each right hand side contains only one attribute
- Result: S = {A→B, A→D, AB→C, C→D, BC→D}
- Reason:
  - Condition 1 for minimal basis:
    The right hand side of each FD contains only one attribute



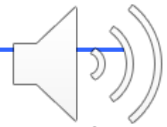
Algorithm for Minimal Basis

- Result of Step 1:
  - S = {A→B, A→D, AB→C, C→D, BC→D}
- Step 2: Remove redundant FDs
- Is A→B redundant?
- i.e., is A→B implied by other FDs in S?
- Let's check
- Without A→B, we have {A→D, AB→C, C→D, BC→D}
- Given those FDs, we have $\{A\}^+ = \{AD\}$, which does not contain B
- Therefore, A→B is not implied by the other FDs



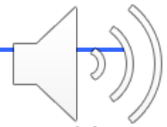
Algorithm for Minimal Basis

- Result of Step 1:
  - S = {A→B, A→D, AB→C, C→D, BC→D}
- Continue Step 2: Remove redundant FDs
- Is A→D redundant?
- i.e., is A→D implied by other FDs in S?
- Let's check
- Without A→D, we have {A→B, AB→C, C→D, BC→D}
- Given those FDs, we have $\{A\}^+ = \{ABCD\}$, which contains D
- Therefore, A→D is implied by the other FDs
- Hence, A→D is redundant and should be removed
- Result: S = {A→B, AB→C, C→D, BC→D}



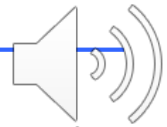
Algorithm for Minimal Basis

- Result of the last step:
  - S = {A→B, AB→C, C→D, BC→D}
- Continue Step 2: Remove redundant FDs
- Is AB→C redundant?
- i.e., is AB→C implied by other FDs in S?
- Let's check
- Without AB→C, we have {A→B, C→D, BC→D}
- Given those FDs, we have $\{AB\}^+ = \{AB\}$, which does not contain C
- Therefore, AB→C is NOT implied by the other FDs
- Hence, AB→C is not redundant and should not be removed



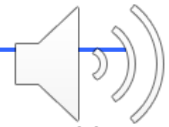
Algorithm for Minimal Basis

- Result of the last step:
  - S = {A→B, AB→C, C→D, BC→D}
- Continue Step 2: Remove redundant FDs
- Is C→D redundant?
- i.e., is C→D implied by other FDs in S?
- Let's check
- Without C→D, we have {A→B, AB→C, BC→D}
- Given those FDs, we have $\{C\}^+ = \{C\}$, which does not contain D
- Therefore, C→D is NOT implied by the other FDs and should not be removed



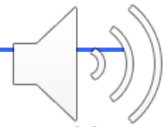
Algorithm for Minimal Basis

- Result of the last step:
  - S = {A→B, AB→C, C→D, BC→D}
- Continue Step 2: Remove redundant FDs
- Is BC→D redundant?
- i.e., is BC→D implied by other FDs in S?
- Let's check
- Without BC→D, we have {A→B, AB→C, C→D}
- Given those FDs, we have $\{BC\}^+ = \{BCD\}$, which contains D
- Therefore, BC→D is implied by the other FDs
- Hence, BC→D is redundant and should be removed
- Result: S = {A→B, AB→C, C→D}



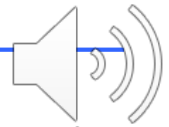
Algorithm for Minimal Basis

- Result of Step 2:
  - S = {A→B, AB→C, C→D}
- Step 3: Remove redundant attributes on the left hand side (lhs) of each FD
- Only AB→C has more than one attribute on the lhs
- Let's check
- Is A redundant?
- If we remove A, then AB→C becomes B→C
- Whether this removal is OK depends on whether B→C is "hidden" in S already
- If B→C is "hidden" in S, then the removal of A is OK, (since the removal does not add extraneous information into S)
- Is B→C "hidden" in S?
- Check: Given S, we have $\{B\}^+ = \{B\}$, which does NOT contain C
- Therefore, B→C is not "hidden" in S, and hence, A is NOT redundant



Algorithm for Minimal Basis

- Result of Step 2:
  - S = {A→B, AB→C, C→D}
- Step 3: Remove redundant attributes on the left hand side (lhs) of each FD
- Only AB→C has more than one attribute on the lhs
- Let's check
- Is B redundant?
- If we remove B, then AB→C becomes A→C
- Whether this is OK depends on whether A→C is "hidden" in S
- Is A→C "hidden" in S?
- Check: Given S, we have $\{A\}^+ = \{ABCD\}$, which contains C
- Therefore, A→C is "hidden" in S
- Hence, we can simplify AB→C to A→C
- Final minimal basis: S = {A→B, A→C, C→D}



To continue in

Topic 4: Third Normal Form (2)