Schema Refinement & Normalization Theory 4

Week 12 - 2
BCNF Decomposition Example 2

• Assume relation schema CSJDPQV:
  
  `Contracts(contract_id, supplier, project, dept, part, qty, value)`
  
  - key C, JP → C, SD → P, J → S

• To deal with SD → P, decompose into SDP, CSJDQV.
• To deal with J → S, decompose CSJDQV into JS and CJDQV
• A tree representation of the decomposition:
BCNF Decomposition

- In general, several dependencies may cause violation of BCNF. The order in which we “deal with” them could lead to very different sets of relations!
How do we know R is in BCNF?

• If R has only two attributes, then it is in BCNF

• If F only uses attributes in R, then:
  – R is in BCNF if and only if for each $X \rightarrow Y$ in $F$ (not $F^+$!), $X$ is a superkey of R, i.e., $X \rightarrow R$ is in $F^+$ (not $F$!).
Checking for BCNF Violations

- List all non-trivial FDs
- Ensure that left hand side of each FD is a superkey
- We have to first find all the keys!
Checking for BCNF Violations

• Is Courses(course_num, dept_name, course_name, classroom, enrollment, student_name, address) in BCNF?
• FDs are:
  – course_num, dept_name → course_name
  – course_num, dept_name → classroom
  – course_num, dept_name → enrollment
• What is (course_num, dept_name)+?
  – {course_num, dept_name, course_name, classroom, enrollment}
• Therefore, the key is
  {course_num, dept_name, course_name, classroom, enrollment, student_name, address}
• The relation is not in BCNF
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 → Z, Z → C

\[(city, street\_name) \rightarrow zip\_code\]
\[zip\_code \rightarrow city\]

• Can’t decompose while **preserving** CS → 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

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

• \( R_1 = (A, B), \ R_2 = (A, C) \)
  – Lossless-join decomposition:
    \[ R_1 \cap R_2 = \{ A \} \text{ and } A \rightarrow AB \]
  – 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 of set of FDs $F$:** If $R$ is decomposed into $X$, ... the projection 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 (Contd.)

• 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!
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 $X \rightarrow A$ ($X \in R$ and $A \in R$) in $F$, one of the following statements is true:
  
  - $A \in X$ (trivial FD), or
  - $X$ is a superkey, or
  - $A$ is part of some key 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 reln 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 decompression into BCNF can be used to obtain a lossless join decompression 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 minimal cover for \( F \).
Minimal Cover for a Set of FDs

- **Minimal cover** $G$ for a set of FDs $F$:
  - Closure of $F = \text{closure of } G$.
  - 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 $F$. 