ICS 321 Fall 2009

Schema Refinement & Normal Forms (iii)

Asst. Prof. Lipyeow Lim
Information & Computer Science Department
University of Hawaii at Manoa
Decompositions

• Reduces redundancies and anomalies, but could have the following potential problems:
  – Some queries become more expensive.
  – Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation!
  – Checking some dependencies may require joining the instances of the decomposed relations.

• Two desirable properties:
  – Lossless-join decomposition
  – Dependency-preserving decomposition
Lossless-join Decomposition

• Decomposition of R into X and Y is *lossless-join* w.r.t. a set of FDs F if, for every instance \( r \) that satisfies F:

\[
\pi_X(r) \text{ join } \pi_Y(r) = r
\]

• In general one direction \( \pi_X(r) \text{ join } \pi_Y(r) \supseteq r \) is always true, but the other may not hold.

• Definition extended to decomposition into 3 or more relations in a straightforward way.

• *It is essential that all decompositions used to deal with redundancy be lossless! (Avoids Problem (2).)*
Conditions for Lossless Join

- The decomposition of R into X and Y is lossless-join wrt F if and only if the closure of F contains:
  - \( X \cap Y \rightarrow X \), or
  - \( X \cap Y \rightarrow Y \)
- In particular, the decomposition of R into UV and R - V is lossless-join if \( U \rightarrow V \) holds over R.
Dependency-preserving Decomposition

- 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)).*

- **Projection of set of FDs F:** If R is decomposed into X, ... 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 $X$. 

\[
F = \{ \text{SC} \rightarrow \text{I}, \text{I} \rightarrow \text{C} \}
\]

Checking $\text{SC} \rightarrow \text{I}$ requires a join!
Dependency-preserving Decomp. (Cont)

• 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! (Example?)
Decomposition into BCNF

• Consider relation R with FDs F. How do we decompose R into a set of small relations that are in BCNF?

• Algorithm:
  – If X → Y violates BCNF, decompose R into R-Y and XY
  – Repeat until all relations are in BCNF.

• Example: CSJDPQV, key C, JP → C, SD → P, J → S
  – To deal with J → S, decompose CSJDPQV into JS and CJDPQV
  – To deal with SD → P, decompose into SDP, CSJDQV

• Order in which we deal with the violating FD can lead to different relations!
BCNF & Dependency Preservation

• BCNF decomposition is lossless join, but there may not be a dependency preserving decomposition into BCNF
  – e.g., CSZ, CS→Z, Z→C
  – Can’t decompose while preserving 1st FD; not in BCNF.
• Similarly, decomposition of CSJDQV into SDP, JS and CJDQV is not dependency preserving (w.r.t. the FDs JP C, SD→P and J →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! *Redundancy!*
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).

• How can we ensure dependency preservation?
  – If $X \rightarrow Y$ is not preserved, add relation $XY$.
  – Problem is that $XY$ may violate 3NF! e.g., consider the addition of CJP to `preserve’ JP→C. What if we also have J→C?

• Refinement: Instead of the given set of FDs $F$, use a *minimal cover for $F$*. 
Minimum 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.

• e.g., A → B, ABCD → E, EF → GH, ACDF → EG has the following minimal cover:
  – A → B, ACD → E, EF → G and EF → H
Computing the Minimal Cover

• Algorithm
  1. **Put the FDs into standard form X → A.** RHS is a single attribute.
  2. **Minimize the LHS of each FD.** For each FD, check if we can delete an attribute from LHS while preserving F+.
  3. **Delete redundant FDs.**

• Minimal covers are not unique. Different order of computation can give different covers.
• e.g., A → B, ABCD → E, EF → GH, ACDF → EG has the following minimal cover:
  – A → B, ACD → E, EF → G and EF → H
Refining an ER Diagram

• 1st diagram translated:
  Workers(S,N,L,D,S)
  Departments(D,M,B)
  – Lots associated with workers

• Suppose all workers in a dept are assigned the same lot: D→L

• Redundancy; fixed by:
  Workers2(S,N,D,S)
  Dept_Lots(D,L)

• Can fine-tune this:
  Workers2(S,N,D,S)
  Departments(D,M,B,L)
Summary of Schema Refinement

• 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.
  – Must consider whether all FDs are preserved. If a lossless-join, dependency preserving decomposition into BCNF is not possible (or unsuitable, given typical queries), should consider decomposition into 3NF.
  – Decompositions should be carried out and/or re-examined while keeping performance requirements in mind.