Schema Refinement and Normal Forms

Chapter 19

Decomposition of a Relation Scheme

- 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 no attributes that do not appear in R), and
  - Every attribute of R appears as an attribute of at least one of the new relations.
- Intuitively, decomposing R means we will store instances of the relation schemes produced by the decomposition, instead of instances of R.
- E.g., Can decompose SNLRWH into SNLRH and RW.

Example Decomposition

- Decompositions should be used only when needed.
  - SNLRWH has FDs S → SNLRWH and R → W
  - Second FD causes violation of 3NF; W values repeatedly associated with R values. Easiest way to fix this is to create a relation RW to store these associations, and to remove W from the main schema:
    - i.e., we decompose SNLRWH into SNLRH and RW
- The information to be stored consists of SNLRWH tuples. If we just store the projections of these tuples onto SNLRH and RW, *are there any potential problems that we should be aware of?*

Problems with Decompositions

- There are three potential problems to consider:
  1. Some queries become more expensive.
     - e.g., How much did Joe earn? (salary = W*H)
  2. Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation!
     - Fortunately, not in the SNLRWH example.
  3. Checking some dependencies may require joining the instances of the decomposed relations.
     - Fortunately, not in the SNLRWH example.
- *Tradeoff*: Must consider these issues vs. redundancy.

Lossless Join Decompositions

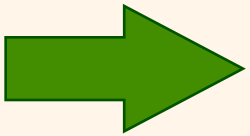
- Decomposition of R into X and Y is *lossless-join* w.r.t. a set of FDs F if, for every instance *r* that satisfies F:
  - $\pi_X(r) \bowtie \pi_Y(r) = r$
- It is always true that $r \subseteq \pi_X(r) \bowtie \pi_Y(r)$
  - In general, the other direction does not hold! If it does, the decomposition is lossless-join.
- 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).)*

More on Lossless Join

A	B	C
1	2	3
4	5	6
7	2	8

A	B
1	2
4	5
7	2

B	C
2	3
5	6
2	8



A	B	C
1	2	3
4	5	6
7	2	8
1	2	8
7	2	3

- 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
  - $X \cap Y \rightarrow Y$

Dependency Preserving Decomposition

- Consider CSJDPQV, C is key, JP → C and SD → P.
  - BCNF decomposition: CSJDQV and SDP (lossless-join)
  - Problem: Checking (Enforcing) JP → C requires a join!

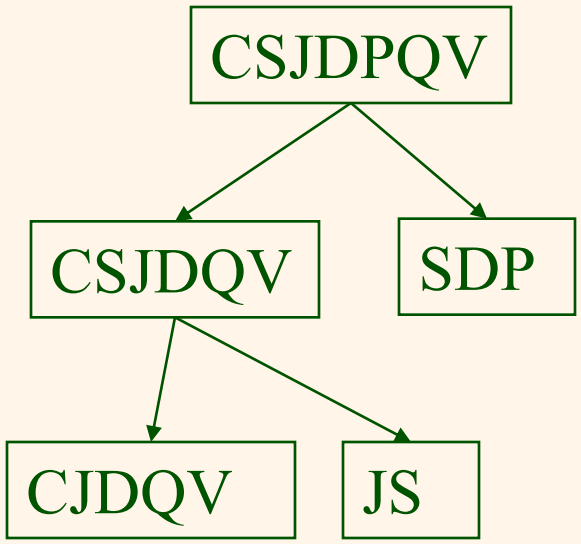
- Dependency preserving decomposition (Intuitive):
  - If R is decomposed into X, Y and Z, and we enforce the FDs on X, on Y and on Z, *(separately)* then all FDs that were given to hold on R must also hold. *(Avoids Problem (3).)*

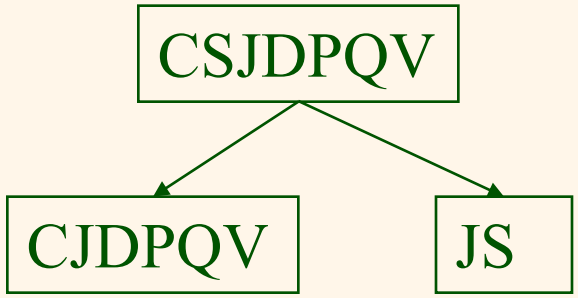
Decomposition into BCNF

- Consider relation R with FDs F. If X → Y violates BCNF, decompose R into R - Y and XY.
  - Repeated application of this idea will give us a collection of relations that are in BCNF; lossless join decomposition, and guaranteed to terminate.

Decomposition into BCNF

- Example: CSJDPQV, 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





- 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!

BCNF and Dependency Preservation

- In general, 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).
- To ensure dependency preservation, one idea:
  - 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' $JP \rightarrow C$. What if we also have $J \rightarrow C$?