ICS 321 Fall 2010
Normal Forms (ii)

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

Redundancies & Decompositions

Hourly_Emps

SSN	Name	Lot	Rating	Hourly_wages	Hours_worked
123-22-2366	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Hourly_Emps

SSN	Name	Lot	Rating	Hours_ worked
123-22-2366	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

RatingWages

Rating	Hourly_ wages
5	7
8	10

Decompositions

- Reduces redundancies and anomalies, but could have the following potential problems:
  1. Some queries become more expensive.
  2. Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation!
  3. 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.

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

Chase Test for Lossless Join

- R(A,B,C,D) is decomposed into S1={A,D}, S2={A,C}, S3={B,C,D}
- Construct a Tableau
  - One row for each decomposed relation
  - For each row i, subcript an attribute with i if it does not occur in Si.
- FDs: A→B, B →C, CD →A
- Rules for "equating two rows" using FDs:
  - If one is unsubscripted, make the other the same
  - If both are subscripted, make the subscripts the same
- Goal: one unsubscripted row

A	B	C	D	
a	b_1	c_1	d	S1
a	b_2	c	d_2	S2
a_3	b	c	d	S3

A	B	C	D
a	b_1	~~c_1~~ c	d
a	~~b_2~~ b_1	c	d_2
~~a_3~~ a	b	c	d

one unsubscripted row imply lossless join

Dependency-preserving Decomposition

Student	Course	Instructor
Smith	OS	Mark

→

Student	Instructor
Smith	Mark

Course	Instructor
OS	Mark

F= { SC → I, I→C }

Checking SC → I 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).)*
- *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 → V in F^+ (*closure of F*) such that U, V are in X.

Dependency-preserving Decomp. (Cont)

- Decomposition of R into X and Y is ***dependency preserving*** if $(F_X \text{ union } F_Y)^+ = F^+$
  - 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 $X \rightarrow Y$ violates BCNF, decompose R into R-Y and XY
  - Repeat until all relations are in BCNF.
- Example: CSJDPQV, key C, $JP \rightarrow C$, $SD \rightarrow P$, $J \rightarrow S$
  - To deal with $J \rightarrow S$, decompose CSJDPQV into JS and CJDPQV
  - To deal with $SD \rightarrow P$, 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' $JP \rightarrow C$. What if we also have $J \rightarrow C$?
- Refinement: Instead of the given set of FDs F, use a *minimal cover for F*.

Minimum Cover for a Set of FDs

- ***Minimal cover or basis*** 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., $A \rightarrow B$, $ABCD \rightarrow E$, $EF \rightarrow GH$, $ACDF \rightarrow EG$ has the following minimal cover:
  - $A \rightarrow B$, $ACD \rightarrow E$, $EF \rightarrow G$ and $EF \rightarrow H$

Computing the Minimal Cover

- Algorithm
  1. **Put the FDs into standard form X $\rightarrow$ 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 $\rightarrow$B, ABCD $\rightarrow$E, EF$\rightarrow$GH, ACDF $\rightarrow$EG has the following minimal cover:
  - A$\rightarrow$B, ACD$\rightarrow$E, EF$\rightarrow$ G and EF$\rightarrow$H

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.