Schema Refinement and Normal Forms (continued)
Is a Relation Schema “Good”?

- “Normal forms” determine which relation schemas are “good,” in the sense of avoiding certain kinds of redundancy
- **Boyce-Codd Normal Form (BCNF):** Consider a relation R and a set of functional dependencies S for R.
  
  R is in BCNF with respect to FD set S if for every FD $X \rightarrow A$ that holds over R (i.e., that is in the closure of S, denoted $S^+$), either:
  - $X \rightarrow A$ is trivial, or
  - $X$ contains a key (i.e., $X$ is a superkey)

Algorithm for Decomposing a Relation into BCNF Relations

- **Input:** relation schema R and set S of FDs for R
- **Output:** a decomposition of R into BCNF relations

1. Identify all keys for R based on set S of FDs for R
   // use attribute closure algorithm; NP-complete problem
2. Repeat until no more BCNF violations:
   - Pick any $R'(Y)$ with a (nontrivial) FD $X \rightarrow A$ that violates BCNF, where A is an individual attribute
   - Decompose $R'$ into $R_1(X \cup \{A\})$ and $R_2(Y - \{A\})$
   - Compute FDs that hold over $R_1$ and $R_2$
     // use Armstrong’s axioms or attribute closure algorithm (preferred)
     for every subset of attributes of $R_1$ and $R_2$ over the $R'$ FDs
   - Identify all keys for $R_1$ and $R_2$ based on their FDs

Final decomposed relations may be different depending on which violating FD is chosen in each iteration of Step 2, but all decompositions will be in BCNF.
Does BCNF Decomposition Algorithm Guarantee a Good Decomposition?

- Removes anomalies? Yes
- Enables reconstruction of original relation? Yes, through natural join
  - Can we get too many tuples in natural join result? No
  - Can we get too few tuples in natural join result? No

We will see why, but we need a few definitions first…

Valid Relation Decomposition

Let $R(A_1, A_2, \ldots, A_n)$ be a relation. Then:
- $R_1 = \pi_U(R)$ and
- $R_2 = \pi_V(R)$

form a decomposition of $R$ if $U \cup V = \{A_1, A_2, \ldots, A_n\}$

That is, $R_1$ and $R_2$ together have all attributes of $R$, nothing more, nothing less
Valid Relation Decomposition

Example: MovieBad(title, year, length, starName)
- Movie(title, year, length) and
- Stars(title, year, starName)
form a valid decomposition of MovieBad

How about:
- Movie(title, year, length) and
- Stars2(title, starName)?

Also a valid decomposition, but this is a bad one
Why? We cannot accurately “reconstruct” MovieBad from Movie and Stars2 (natural join might contain extra, “spurious” tuples not in MovieBad…)

Lossless Join Decompositions

A decomposition of a relation R into:
- \( R_1 = \pi_U(R) \) and
- \( R_2 = \pi_V(R) \)
is lossless join with respect to a given set of FDs S if, for every instance of R that satisfies S, \( R = R_1 \bowtie R_2 \)

Note: It is always the case that \( R \subseteq R_1 \bowtie R_2 \) for a valid, not necessarily lossless join decomposition of R into \( R_1 \) and \( R_2 \)

When is a decomposition guaranteed to be lossless join?
Lossless Join Decompositions: A Theorem

Theorem: The decomposition of a relation $R$ into:
- $R_1 = \pi_U(R)$ and
- $R_2 = \pi_V(R)$
is lossless join with respect to a given set of FDs $S$ if and only if $S^+$, the closure of $S$, contains either:
- $U \cap V \rightarrow U$ or
- $U \cap V \rightarrow V$

The BCNF decomposition algorithm always produces lossless join decompositions (proof?)

BCNF Decomposition Algorithm Always Produces Lossless Join Decompositions

Sketch of proof: Focus on one step of the algorithm and consider $R'(Y)$ and an “offending” (with respect to BCNF) FD $X \rightarrow A$, such that $A$ is a single attribute

We decompose $R'(Y)$ into:
- $R_1 = \pi_{X \cup \{A\}}(R)$ and
- $R_2 = \pi_{Y - \{A\}}(R)$

$(X \cup \{A\}) \cap (Y - \{A\}) = X$

And $X \rightarrow X \cup \{A\}$ holds, because $X \rightarrow A$ holds

Therefore, from the theorem, the decomposition of $R'$ into $R_1$ and $R_2$ is lossless join
Decompositions and Efficiency

• So far, we know that we want lossless join decompositions
• But how about efficiency? To check/enforce FDs over a decomposed relation, we sometimes need to take the natural join of the smaller parts, which has a cost
• Hence we want to figure out which decompositions avoid such natural joins to check/enforce FDs
• First, we need a definition

Projection of FD Set Onto an Attribute Set

Let relation R be decomposed into:
• $R_1 = \pi_U(R)$ and
• $R_2 = \pi_V(R)$
and let $S$ be a set of FDs over $R$, with closure $S^+$

Then, the **projection of $S$ onto $U$, denoted $S_U$**, is the set of FDs $X \rightarrow A$ in $S^+$ (not $S$!) such that all attributes in $X$ and $A$ are in $U$. ($S_V$ is defined analogously.)
Dependency Preserving Decompositions

A decomposition of a relation R into:
- $R_1 = \pi_U(R)$ and
- $R_2 = \pi_V(R)$

is dependency preserving with respect to a given set of FDs S if $(S_U \cup S_V)^+ = S^+$

In other words, if we enforce “locally” over $R_1$ only the FDs in $S_U$, and over $R_2$ only the FDs in $S_V$, then we don’t need to take any joins or do anything else to guarantee that all FDs in $S^+$ hold as well.

BCNF and Dependency Preservation

- We can always decompose a relation into BCNF relations with a lossless join decomposition, using the algorithm that we have seen.
- However, sometimes there is no dependency preserving decomposition into BCNF.
- This motivates the introduction of a new normal form, 3NF, that allows for a bit more redundancy than BCNF does.
- We can always decompose a relation into 3NF relations with a lossless join and dependency preserving decomposition.
### Third Normal Form, or 3NF

**3NF:** Consider a relation $R$ and a set of functional dependencies $S$ for $R$. $R$ is in **3NF** with respect to FD set $S$ if for every FD $X \rightarrow A$ that holds over $R$ (i.e., that is in the closure of $S$, denoted $S^+$), where $X$ is a set of attributes and $A$ is a single attribute:

- $X \rightarrow A$ is trivial, or
- $X$ contains a key (i.e., $X$ is a superkey), or
- $A$ is a member of some key (not superkey)

Because of the (new) third condition, 3NF is “more relaxed” than BCNF: every relation in BCNF is also in 3NF, but not the other way around.

### Checking for 3NF Compliance?

- Finding all keys of a relation from its FDs is an NP-complete problem
- Hence checking if a relation is in 3NF is an NP-complete problem as well
  - For each $X \rightarrow A$ that is nontrivial and such that $X$ is not a superkey, we need to know whether $A$ is part of a key or not…
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 goal.

• If a relation is not in BCNF, we can always decompose it into a collection of BCNF relations (algorithm covered in class), such that the decomposition:
  • is guaranteed to be lossless join, but
  • might not be dependency preserving

• 3NF tolerates a bit more redundancy than BCNF

• If a relation is not in 3NF, we can always decompose it into a collection of 3NF relations such that the decomposition is guaranteed to be both lossless join and dependency preserving (algorithm not covered in class)