CS 44800: Introduction To Relational Database Systems

Prof. Chris Clifton

7 September 2021

Functional Dependencies

Functional Dependencies

$X \rightarrow A$ = assertion about a relation R that whenever two tuples agree on all the attributes of X, then they must also agree on attribute A

- Examples:
  – PurdueID → ClassYear
  – Credits → ClassYear
  – Name Address Year Credits Major → PurdueID
    - *May be true, but not something that we want to say must hold!*

FDs: Armstrong's Axioms

- Reflexivity:
  - If $\{B_1, B_2, \ldots, B_m\} \subseteq \{A_1, A_2, \ldots, A_n\} \Rightarrow A_1A_2\cdots A_n \to B_1B_2\cdots B_m$
  - Also called "trivial FDs"
- Augmentation:
  - $A_1A_2\cdots A_n \to B_1B_2\cdots B_m \Rightarrow$ $A_1A_2\cdots A_nC_1C_2\cdots C_k \to B_1B_2\cdots B_mC_1C_2\cdots C_k$
- Transitivity:
  - $A_1A_2\cdots A_n \to B_1B_2\cdots B_m$ and $B_1B_2\cdots B_m \to C_1C_2\cdots C_k \Rightarrow A_1A_2\cdots A_n \to C_1C_2\cdots C_k$

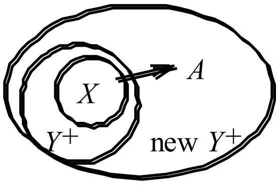
Closure of a Set of Functional Dependencies

- Given a set *F* set of functional dependencies, Armstrong's axioms show there are certain other functional dependencies that are logically implied by *F*.
  - If $A \to B$ and $B \to C$, then we can infer that $A \to C$
  - etc.
- The set of **all** functional dependencies logically implied by *F* is the **closure** of *F*.
- We denote the *closure* of *F* by $\boldsymbol{F^+}$.

Algorithm

Define Y^+ = *closure* of Y = set of attributes functionally determined by Y:

- Basis: $Y^+ := Y$.
- Induction: If $X \subseteq Y^+$, and $X \rightarrow A$ is a given FD, then add A to Y^+.

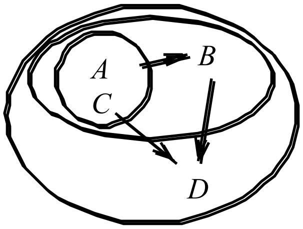


- End when Y^+ cannot be changed.

Example

$A \rightarrow B$, $BC \rightarrow D$.

- $A^+ = AB$.
- $C^+ = C$.
- $(AC)^+ = ABCD$.



Given Versus Implied FD's

Typically, we state a few FD's that are known to hold for a relation *R*.

- Other FD's may follow logically from the given FD's; these are *implied FD's*.
- We are free to choose any *basis* for the FD's of *R* – a set of FD's that imply all the FD's that hold for *R*.

Finding All Implied FD's

Motivation: Suppose we have a relation *ABCD* with some FD's *F*. If we decide to decompose *ABCD* into *ABC* and *AD*, what are the FD's for *ABC*, *AD*?

- Example: $F = AB \rightarrow C$, $C \rightarrow D$, $D \rightarrow A$. It looks like just $AB \rightarrow C$ holds in *ABC*, but in fact $C \rightarrow A$ follows from *F* and applies to relation *ABC*.
- Problem is exponential in worst case.

PURDUE UNIVERSITY

Department of Computer Science

Example

F = *AB* → *C*, *C* → *D*, *D* → *A*. What FD's follow?

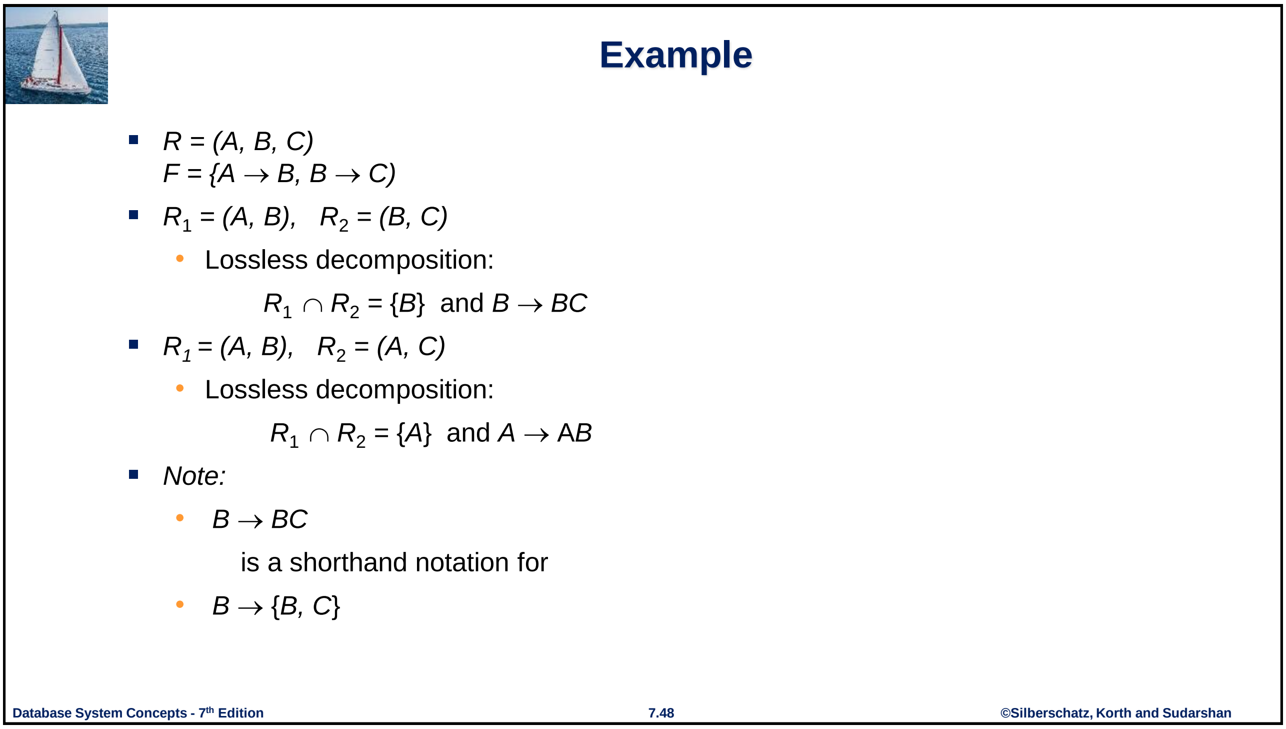
- $A^+ = A$; $B^+ = B$ (nothing).
- $C^+ = ACD$ (add *C* → *A*).
- $D^+ = AD$ (nothing new).
- $(AB)^+ = ABCD$ (add *AB* → *D*; skip all supersets of *AB*).
- $(BC)^+ = ABCD$ (nothing new; skip all supersets of *BC*).
- $(BD)^+ = ABCD$ (add *BD* → *C*; skip all supersets of *BD*).
- $(AC)^+ = ACD$; $(AD)^+ = AD$; $(CD)^+ = ACD$ (nothing new).
- $(ACD)^+ = ACD$ (nothing new).
- All other sets contain *AB*, *BC*, or *BD*, so skip.
- Thus, the only interesting FD's that follow from *F* are: *C* → *A*, *AB* → *D*, *BD* → *C*.

Lossless Decomposition

- We can use functional dependencies to show when certain decomposition are lossless.
- For the case of $R = (R_1, R_2)$, we require that for all possible relations *r* on schema *R*

$$r = \prod_{R1}(r) \bowtie \prod_{R2}(r)$$

- A decomposition of *R* into R_1 and R_2 is lossless decomposition if at least one of the following dependencies is in F^+:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$
- The above functional dependencies are a sufficient condition for lossless join decomposition; the dependencies are a necessary condition only if all constraints are functional dependencies



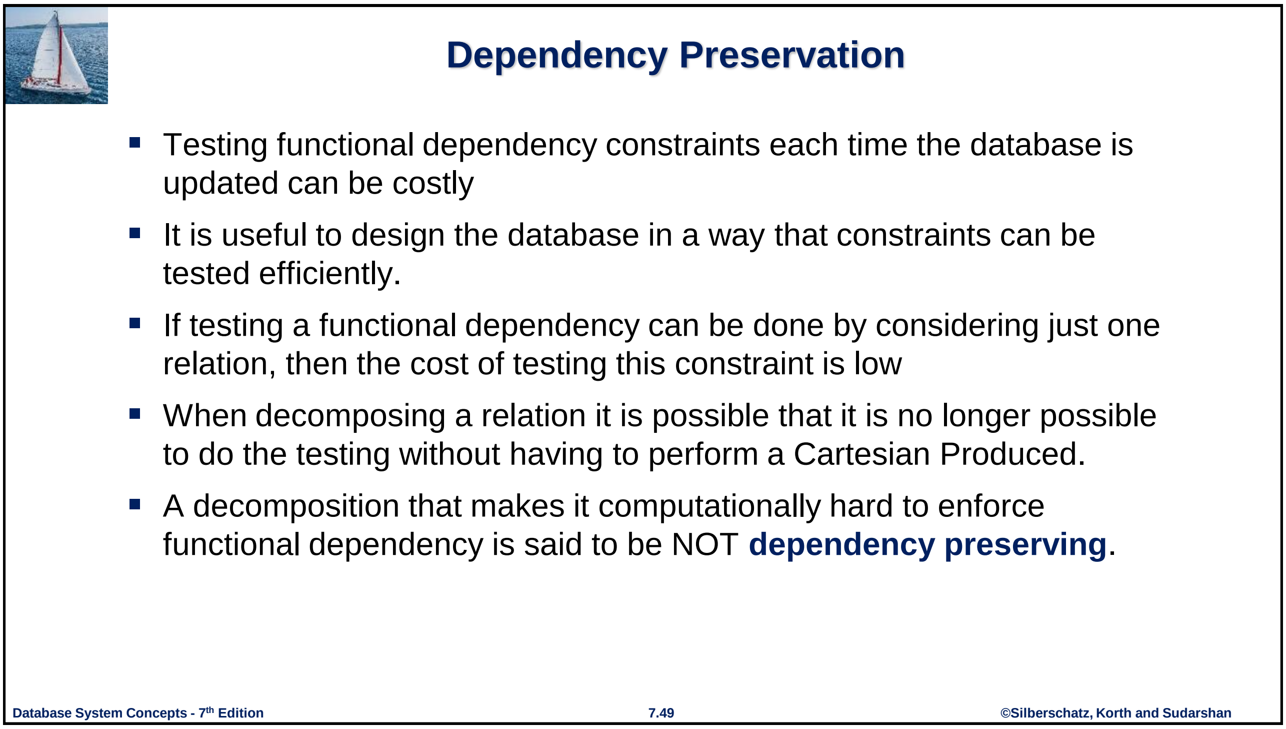
Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C)$
- $R_1 = (A, B), \quad R_2 = (B, C)$
  - Lossless decomposition:

    $R_1 \cap R_2 = \{B\}$ and $B \rightarrow BC$
- $R_1 = (A, B), \quad R_2 = (A, C)$
  - Lossless decomposition:

    $R_1 \cap R_2 = \{A\}$ and $A \rightarrow AB$
- *Note:*
  - $B \rightarrow BC$

    is a shorthand notation for
  - $B \rightarrow \{B, C\}$



Dependency Preservation

- Testing functional dependency constraints each time the database is updated can be costly
- It is useful to design the database in a way that constraints can be tested efficiently.
- If testing a functional dependency can be done by considering just one relation, then the cost of testing this constraint is low
- When decomposing a relation it is possible that it is no longer possible to do the testing without having to perform a Cartesian Produced.
- A decomposition that makes it computationally hard to enforce functional dependency is said to be NOT **dependency preserving**.

Dependency Preservation Example

- Consider a schema:

  dept_advisor(s_ID, i_ID, department_name)
- With function dependencies:

  i_ID → *dept_name*

  s_ID, dept_name → *i_ID*
- In the above design we are forced to repeat the department name once for each time an instructor participates in a *dept_advisor* relationship.
- To fix this, we need to decompose *dept_advisor*
- Any decomposition will not include all the attributes in

  s_ID, dept_name → *i_ID*
- Thus, the composition NOT be dependency preserving

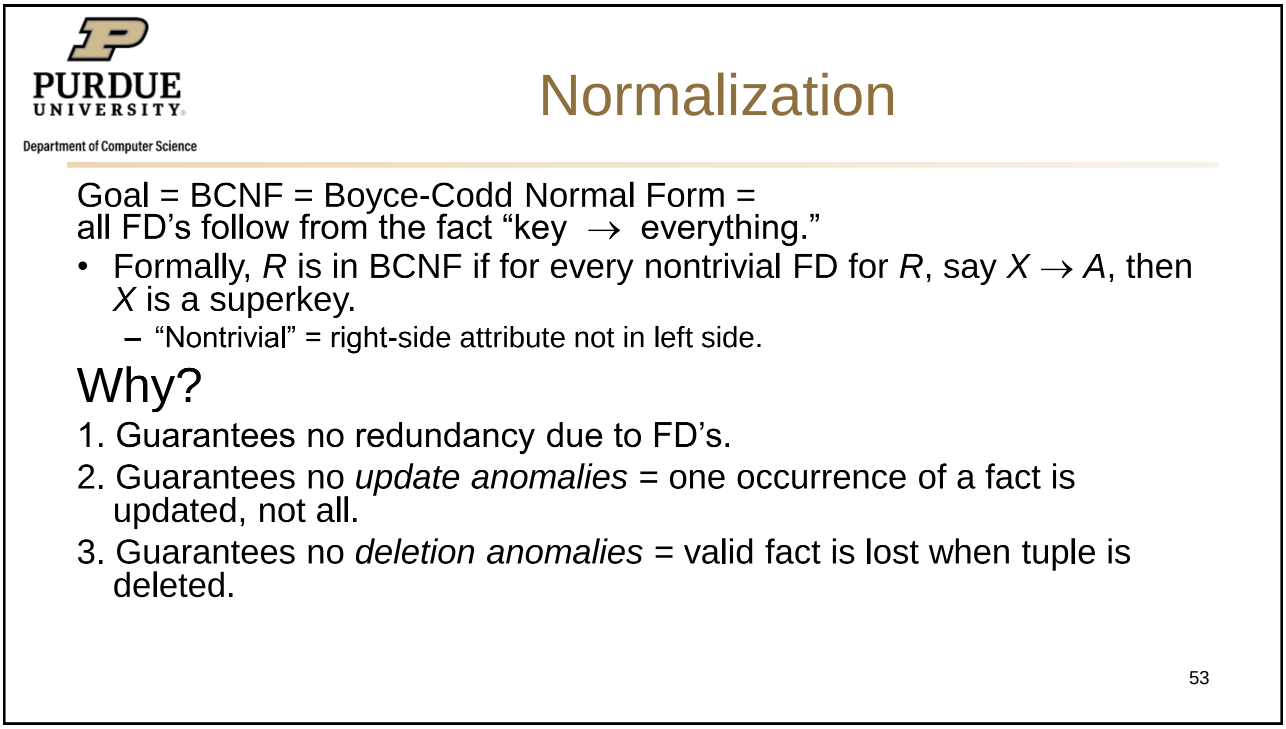
PURDUE UNIVERSITY
Department of Computer Science

Review – Functional Dependencies

In *ABC* with FD's *A* → *B*, *B* → *C*, project onto *AC*.

1. A^+ = *ABC*; yields *A* → *B*, *A* → *C*.
2. B^+ = *BC*; yields *B* → *C*.
3. AB^+ = *ABC*; yields *AB* → *C*; drop in favor of *A* → *C*.
4. AC^+ = *ABC* yields *AC* → *B*; drop in favor of *A* → *B*.
5. C^+ = *C* and BC^+ = *BC*; adds nothing.

- Resulting FD's: *A* → *B*, *A* → *C*, *B* → *C*.
- Projection onto *AC*: *A* → *C*.



PURDUE UNIVERSITY

Department of Computer Science

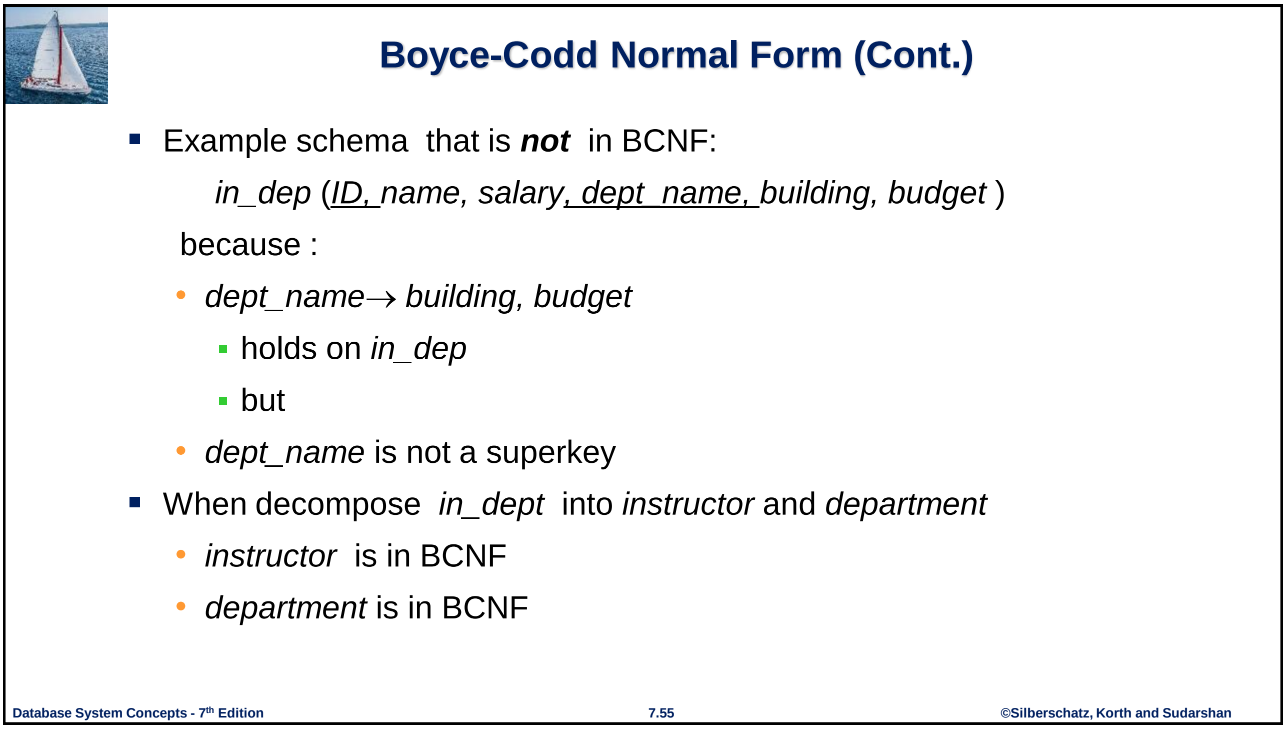
Normalization

Goal = BCNF = Boyce-Codd Normal Form =
all FD’s follow from the fact “key → everything.”

- Formally, *R* is in BCNF if for every nontrivial FD for *R*, say *X* → *A*, then *X* is a superkey.
  - “Nontrivial” = right-side attribute not in left side.

Why?

1. Guarantees no redundancy due to FD’s.
2. Guarantees no *update anomalies* = one occurrence of a fact is updated, not all.
3. Guarantees no *deletion anomalies* = valid fact is lost when tuple is deleted.

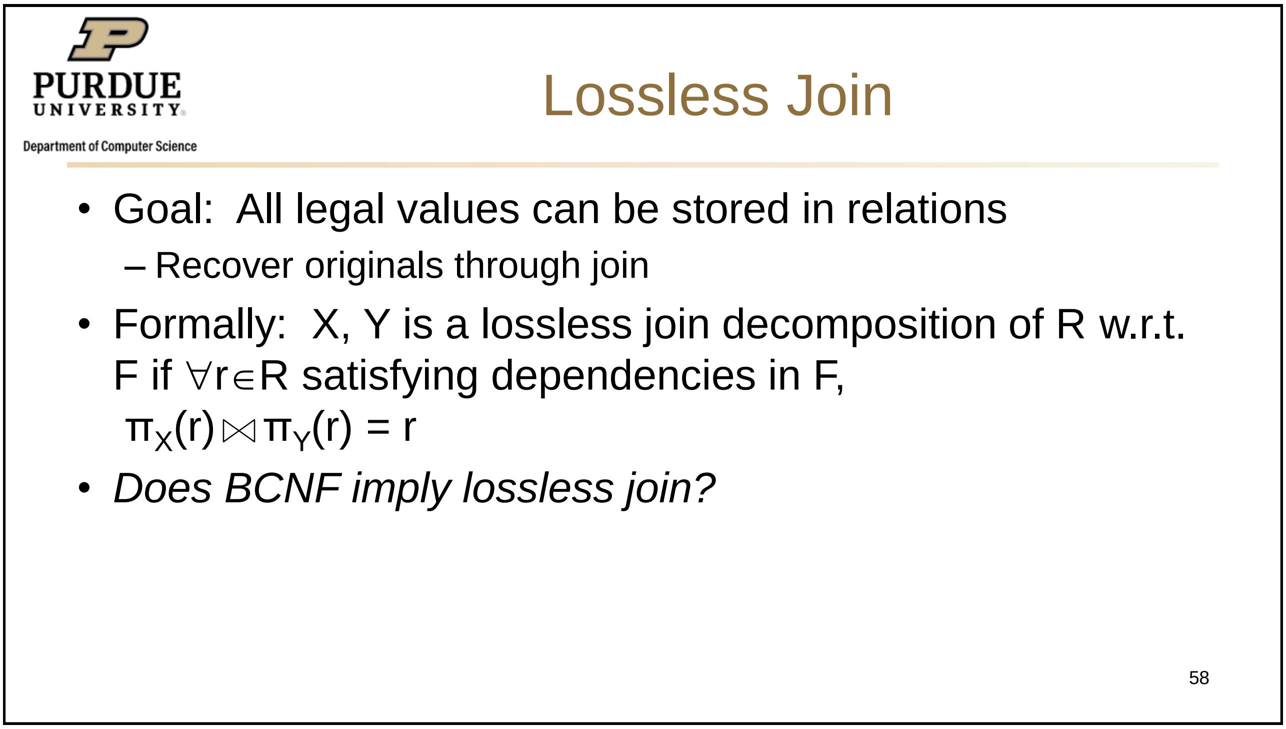


Boyce-Codd Normal Form (Cont.)

- Example schema that is ***not*** in BCNF:

  in_dep (*ID, name, salary, dept_name, building, budget*)

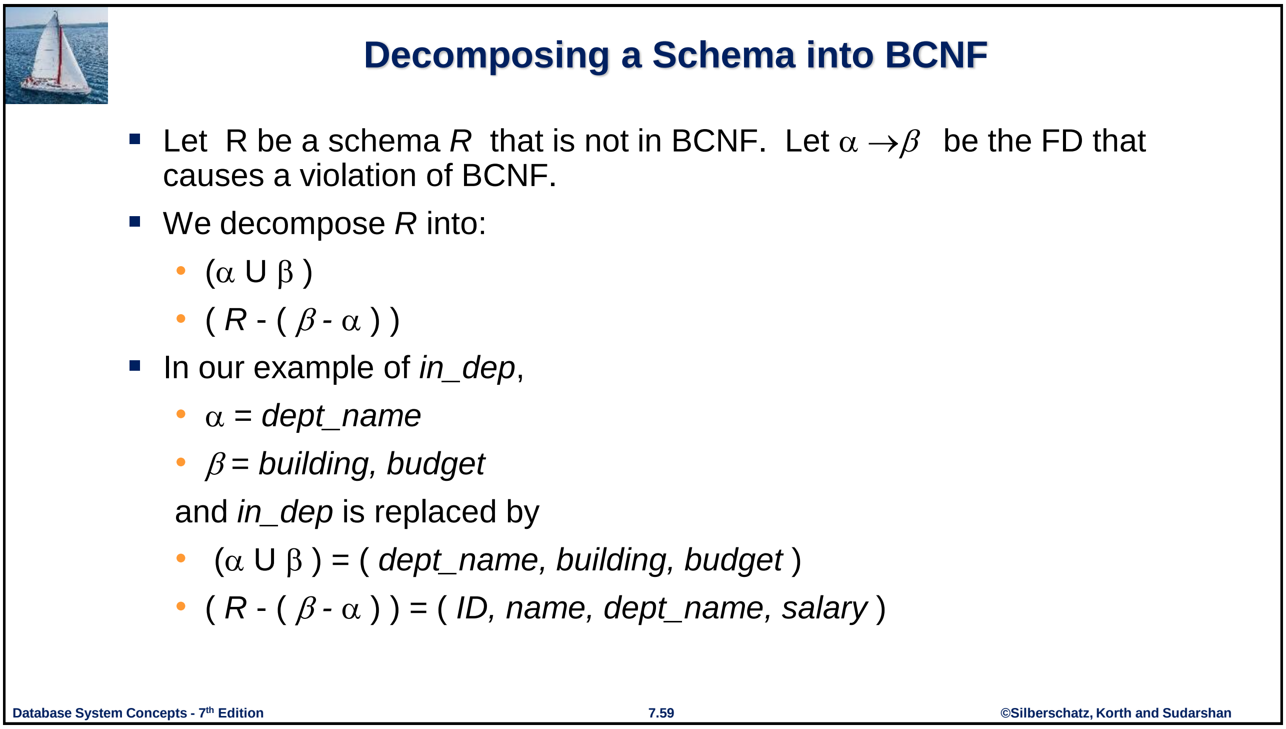
  because :
  - *dept_name*→ *building, budget*
    - holds on *in_dep*
    - but
  - *dept_name* is not a superkey
- When decompose *in_dept* into *instructor* and *department*
  - *instructor* is in BCNF
  - *department* is in BCNF



PURDUE UNIVERSITY
Department of Computer Science

Lossless Join

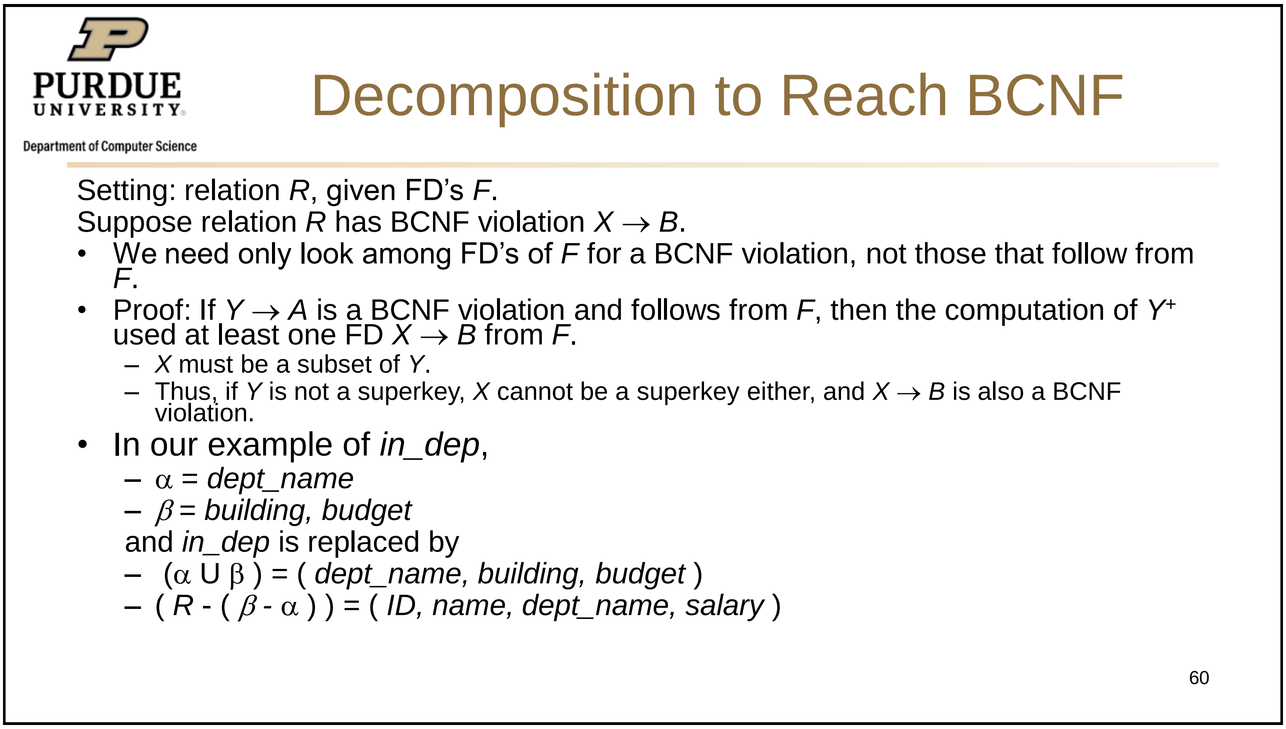
- Goal: All legal values can be stored in relations
  - Recover originals through join
- Formally: X, Y is a lossless join decomposition of R w.r.t. F if $\forall r \in R$ satisfying dependencies in F, $\pi_X(r) \bowtie \pi_Y(r) = r$
- *Does BCNF imply lossless join?*



Decomposing a Schema into BCNF

- Let R be a schema *R* that is not in BCNF. Let $\alpha \rightarrow \beta$ be the FD that causes a violation of BCNF.
- We decompose *R* into:
  - $(\alpha \cup \beta)$
  - $(R - (\beta - \alpha))$
- In our example of *in_dep*,
  - α = *dept_name*
  - β = *building, budget*

  and *in_dep* is replaced by
  - $(\alpha \cup \beta)$ = (*dept_name, building, budget*)
  - $(R - (\beta - \alpha))$ = (*ID, name, dept_name, salary*)

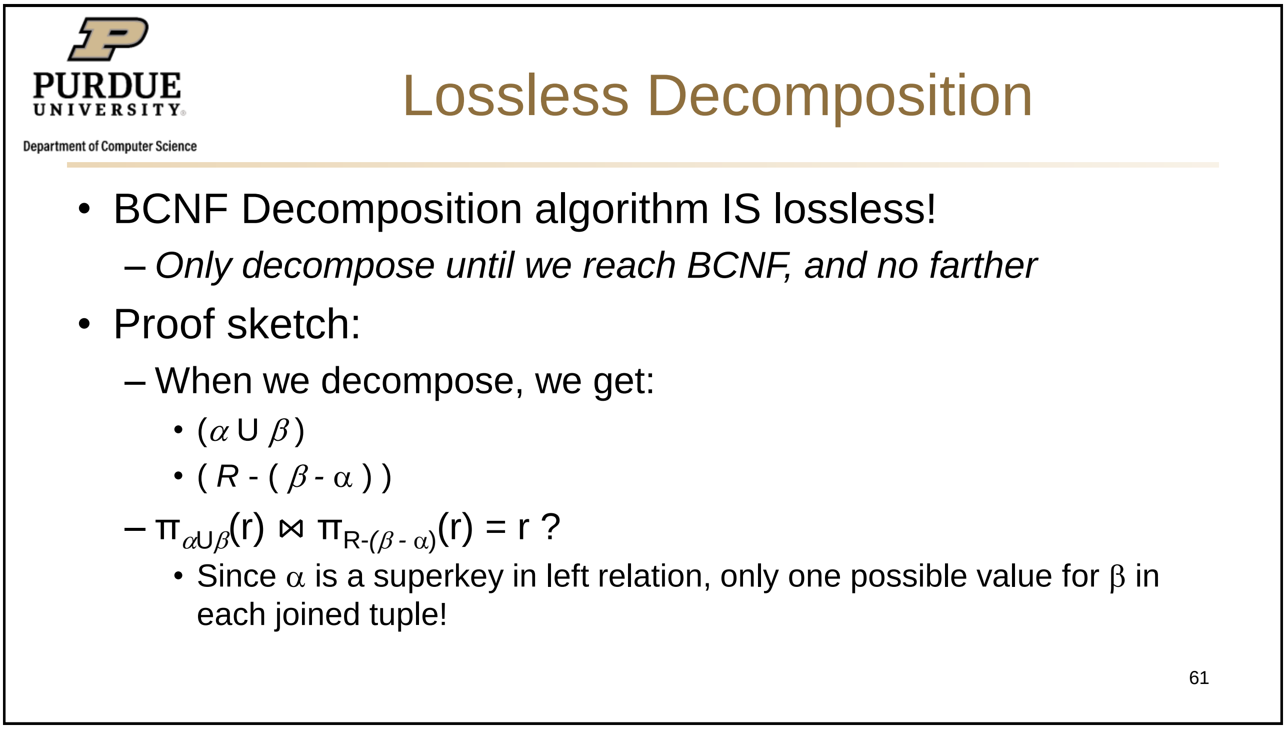


Decomposition to Reach BCNF

Setting: relation *R*, given FD's *F*.
Suppose relation *R* has BCNF violation $X \rightarrow B$.

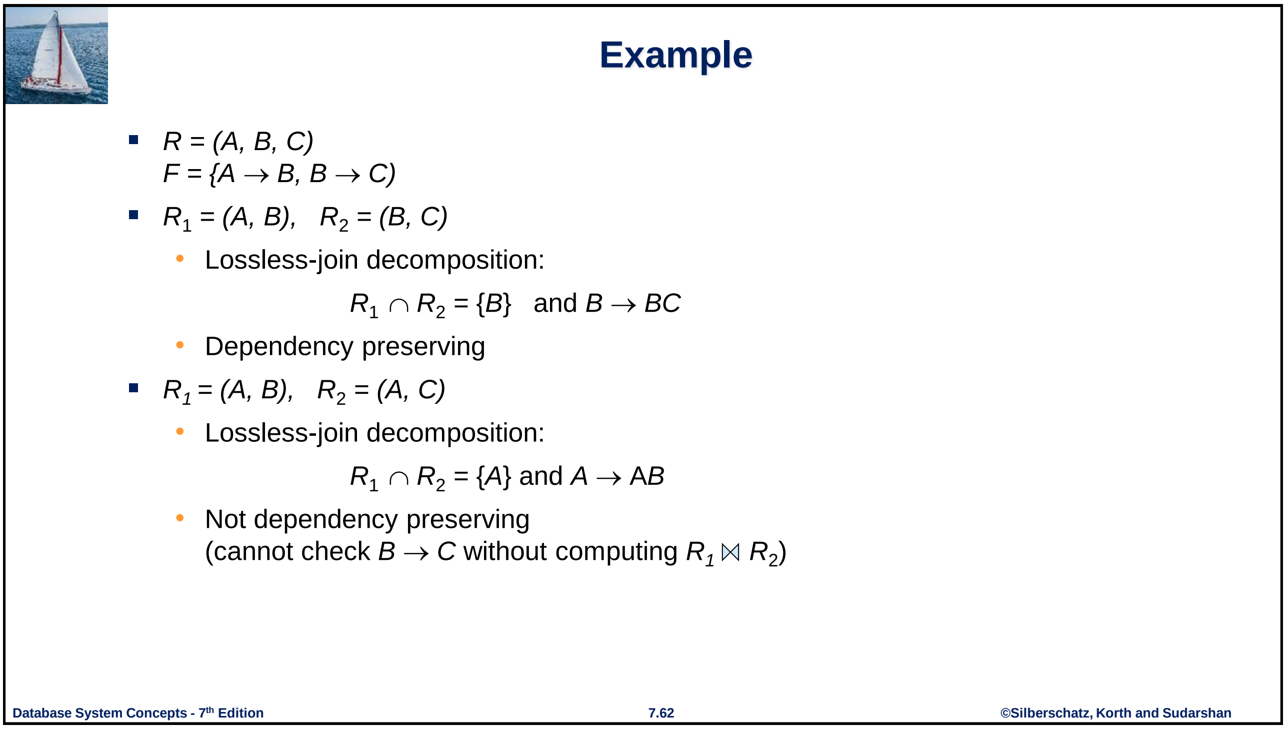
- We need only look among FD's of *F* for a BCNF violation, not those that follow from *F*.
- Proof: If $Y \rightarrow A$ is a BCNF violation and follows from *F*, then the computation of Y^+ used at least one FD $X \rightarrow B$ from *F*.
  - *X* must be a subset of *Y*.
  - Thus, if *Y* is not a superkey, *X* cannot be a superkey either, and $X \rightarrow B$ is also a BCNF violation.
- In our example of *in_dep*,
  - α = *dept_name*
  - β = *building, budget*

  and *in_dep* is replaced by
  - (α U β) = (*dept_name, building, budget*)
  - (*R* - (β - α)) = (*ID, name, dept_name, salary*)



Lossless Decomposition

- BCNF Decomposition algorithm IS lossless!
  - *Only decompose until we reach BCNF, and no farther*
- Proof sketch:
  - When we decompose, we get:
    - (α U β)
    - (*R* - (β - α))
  - $\pi_{\alpha U \beta}(r) \bowtie \pi_{R-(\beta - \alpha)}(r) = r$?
    - Since α is a superkey in left relation, only one possible value for β in each joined tuple!



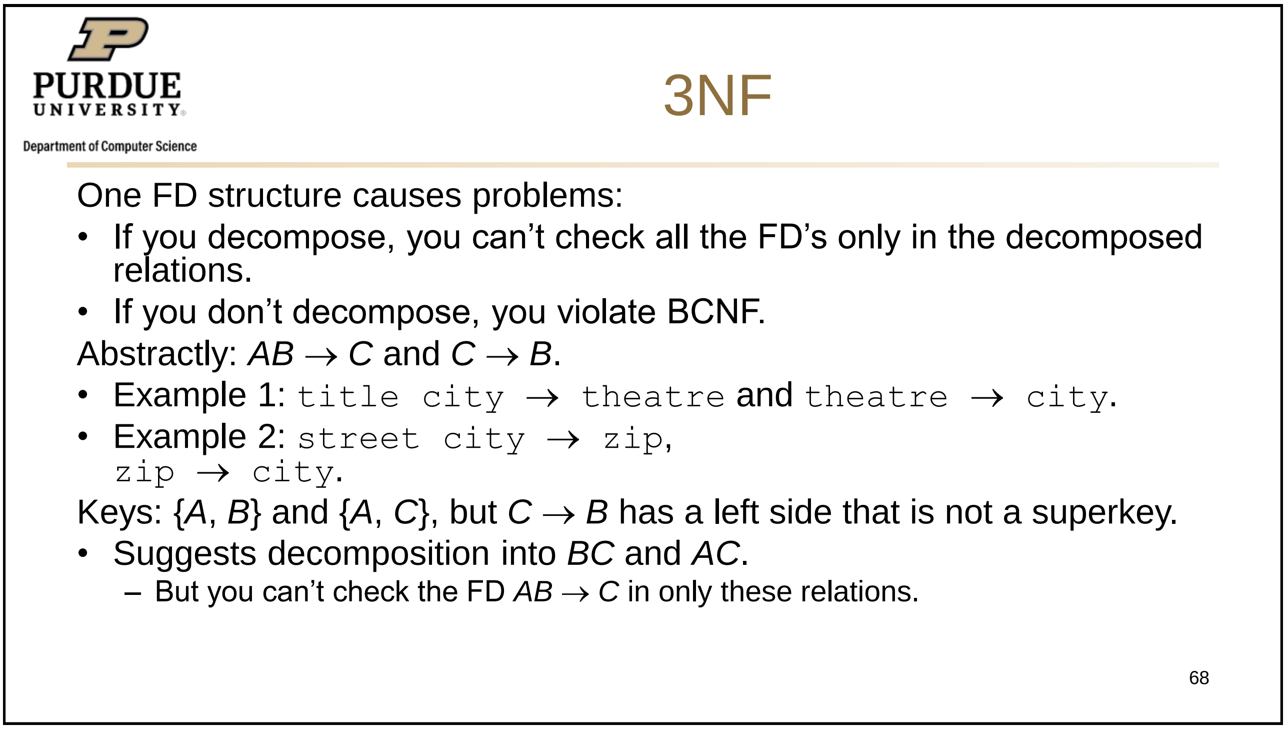
Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C)$
- $R_1 = (A, B)$, $R_2 = (B, C)$
  - Lossless-join decomposition:

    $R_1 \cap R_2 = \{B\}$ and $B \rightarrow BC$
  - Dependency preserving
- $R_1 = (A, B)$, $R_2 = (A, C)$
  - Lossless-join decomposition:

    $R_1 \cap R_2 = \{A\}$ and $A \rightarrow AB$
  - Not dependency preserving
    (cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

Database System Concepts - 7th Edition 7.62 ©Silberschatz, Korth and Sudarshan



PURDUE UNIVERSITY

Department of Computer Science

3NF

One FD structure causes problems:

- If you decompose, you can't check all the FD's only in the decomposed relations.
- If you don't decompose, you violate BCNF.

Abstractly: $AB \rightarrow C$ and $C \rightarrow B$.

- Example 1: `title city` → `theatre` and `theatre` → `city`.
- Example 2: `street city` → `zip`,
  `zip` → `city`.

Keys: $\{A, B\}$ and $\{A, C\}$, but $C \rightarrow B$ has a left side that is not a superkey.

- Suggests decomposition into BC and AC.
  - But you can't check the FD $AB \rightarrow C$ in only these relations.

68

PURDUE UNIVERSITY
Department of Computer Science

“Elegant” Workaround

Define the problem away.

- A relation R is in 3NF iff (if and only if) for every nontrivial FD $X \rightarrow A$, either:
  1. X is a superkey, or
  2. A is *prime* = member of at least one key.
- Thus, the canonical problem goes away: you don’t have to decompose because all attributes are prime.

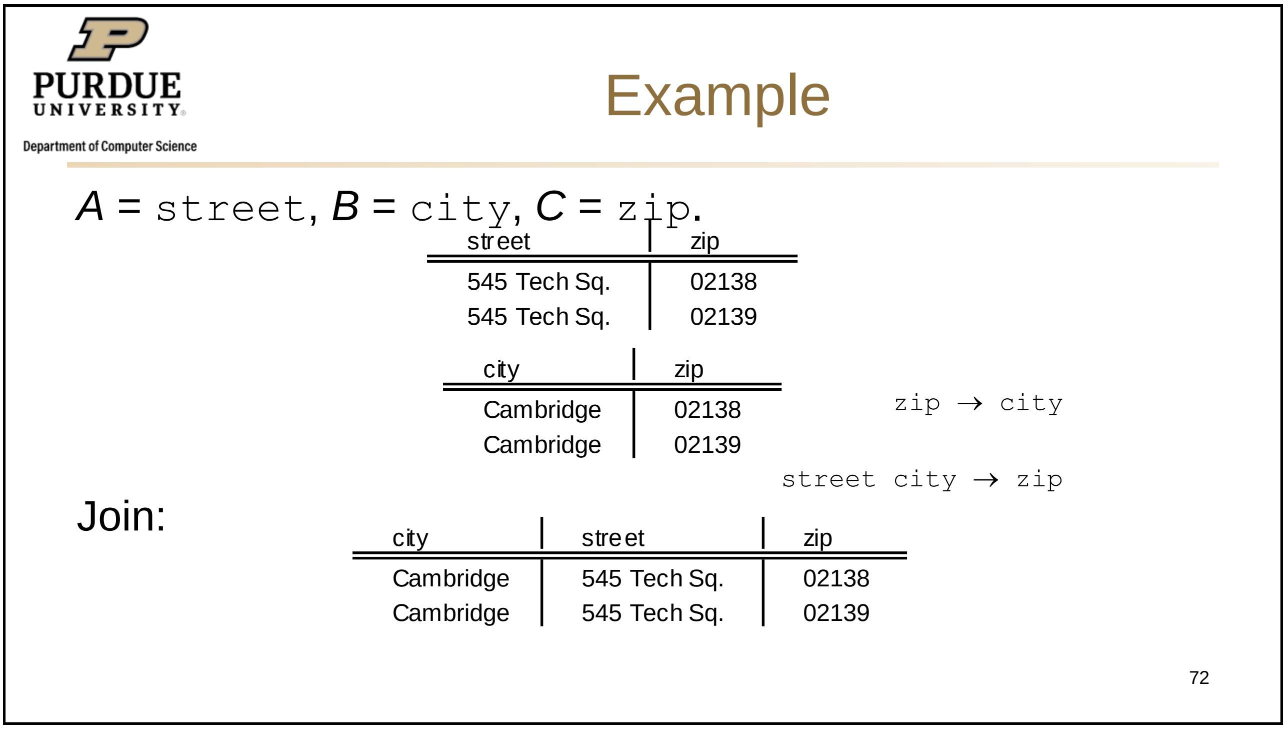
3NF Example

- Consider a schema:

  dept_advisor(s_ID, i_ID, dept_name)
- With function dependencies:

  i_ID → *dept_name*

  s_ID, dept_name → *i_ID*
- Two candidate keys = {*s_ID, dept_name*}, {*s_ID, i_ID* }
- We have seen before that *dept_advisor* is not in BCNF
- *R,* however, is in 3NF
  - *s_ID, dept_name* is a superkey
  - *i_ID* → *dept_name* *and* *i_ID* is NOT a superkey, but:
    - { *dept_name*} – {*i_ID* } = {*dept_name* } and
    - *dept_name* is contained in a candidate key



PURDUE UNIVERSITY

Department of Computer Science

Example

A = street, *B* = city, *C* = zip.

street	zip
545 Tech Sq.	02138
545 Tech Sq.	02139

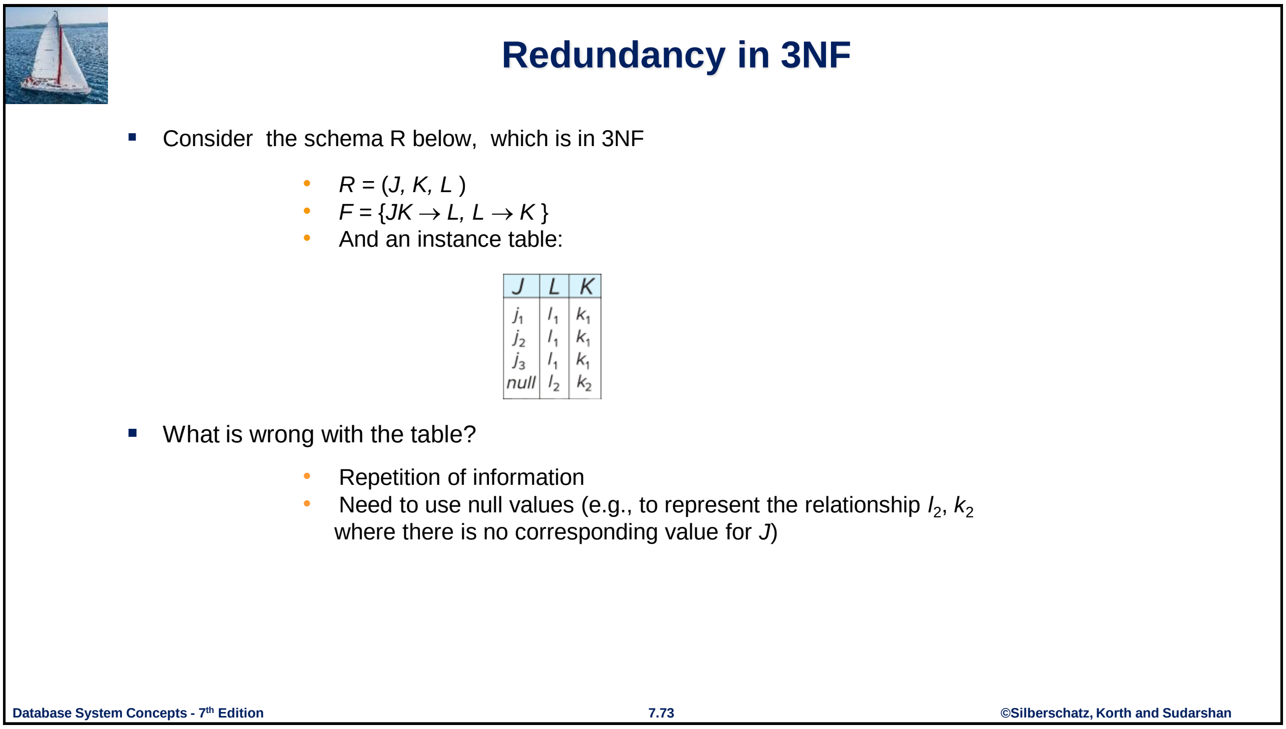
city	zip
Cambridge	02138
Cambridge	02139

zip → city

street city → zip

Join:

city	street	zip
Cambridge	545 Tech Sq.	02138
Cambridge	545 Tech Sq.	02139

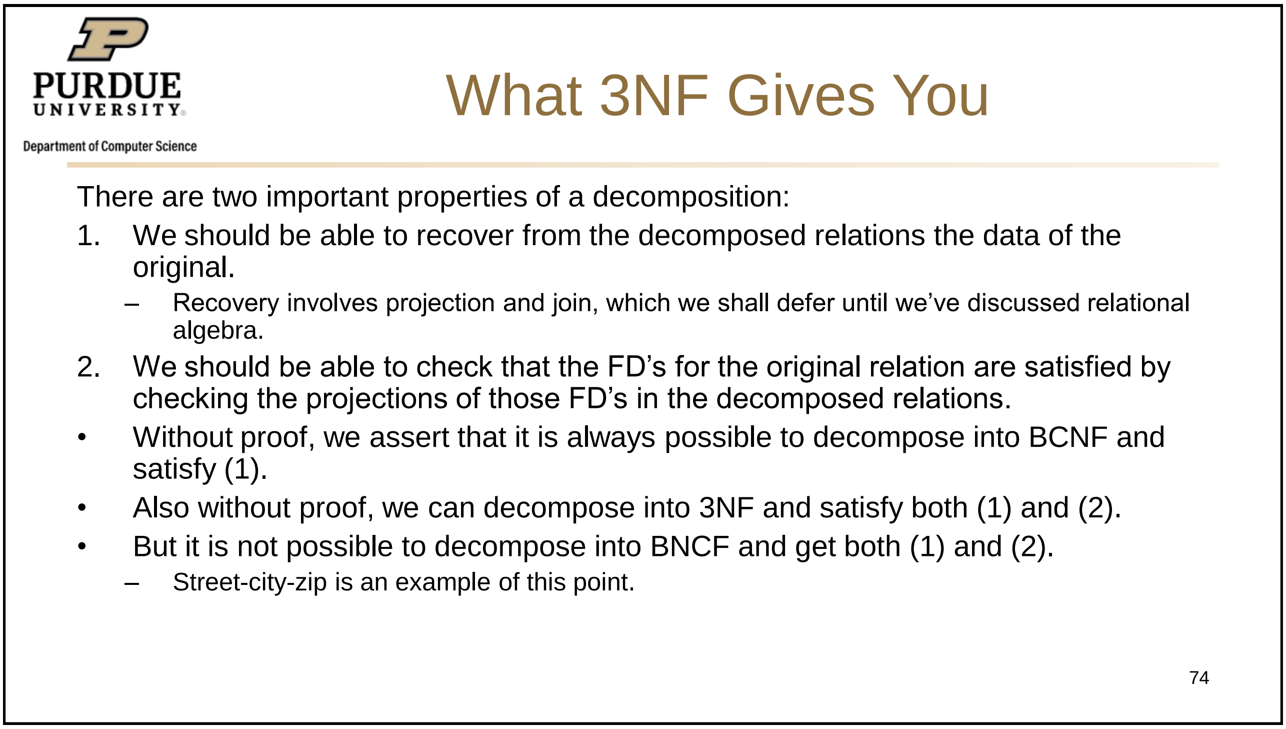


Redundancy in 3NF

- Consider the schema R below, which is in 3NF
  - $R = (J, K, L)$
  - $F = \{JK \rightarrow L, L \rightarrow K\}$
  - And an instance table:

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
null	l_2	k_2

- What is wrong with the table?
  - Repetition of information
  - Need to use null values (e.g., to represent the relationship l_2, k_2 where there is no corresponding value for J)



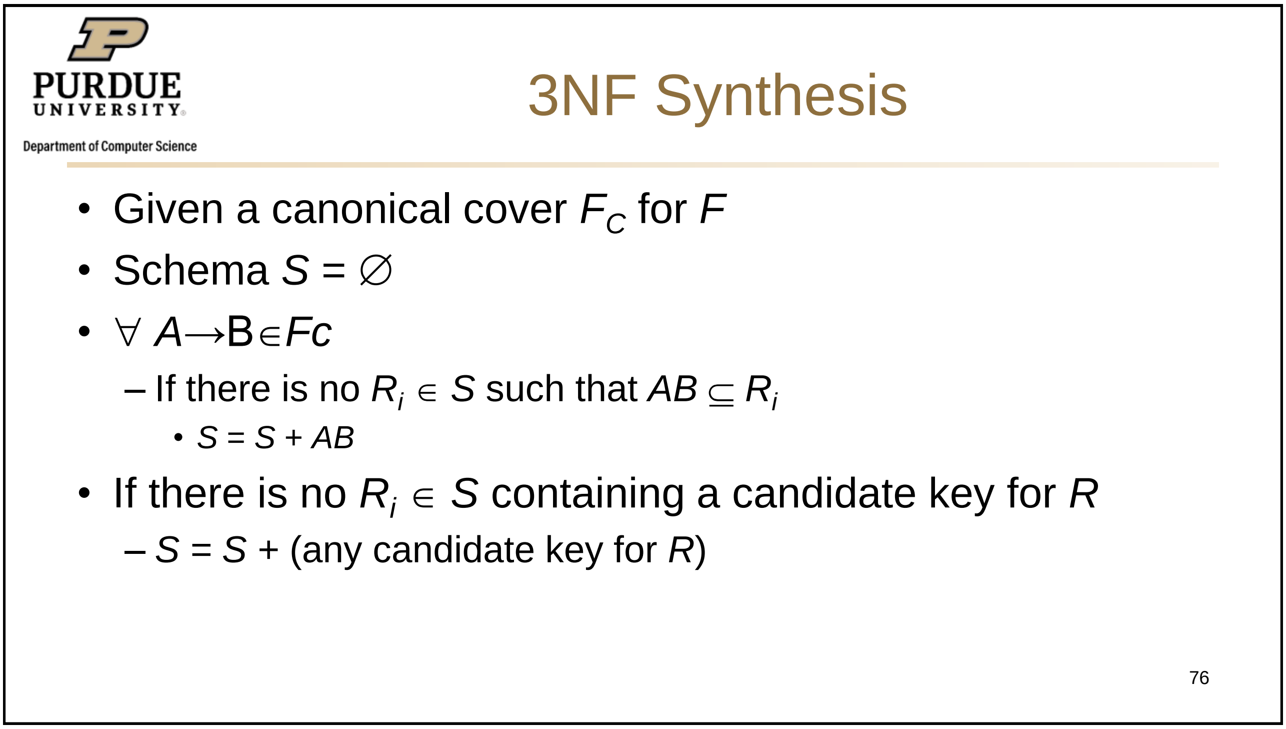
PURDUE UNIVERSITY
Department of Computer Science

What 3NF Gives You

There are two important properties of a decomposition:

1. We should be able to recover from the decomposed relations the data of the original.
   - Recovery involves projection and join, which we shall defer until we've discussed relational algebra.
2. We should be able to check that the FD's for the original relation are satisfied by checking the projections of those FD's in the decomposed relations.

- Without proof, we assert that it is always possible to decompose into BCNF and satisfy (1).
- Also without proof, we can decompose into 3NF and satisfy both (1) and (2).
- But it is not possible to decompose into BNCF and get both (1) and (2).
  - Street-city-zip is an example of this point.



PURDUE UNIVERSITY
Department of Computer Science

3NF Synthesis

- Given a canonical cover F_C for F
- Schema $S = \varnothing$
- $\forall\ A \rightarrow \mathrm{B} \in Fc$
  - If there is no $R_i \in S$ such that $AB \subseteq R_i$
    - $S = S + AB$
- If there is no $R_i \in S$ containing a candidate key for R
  - $S = S$ + (any candidate key for R)