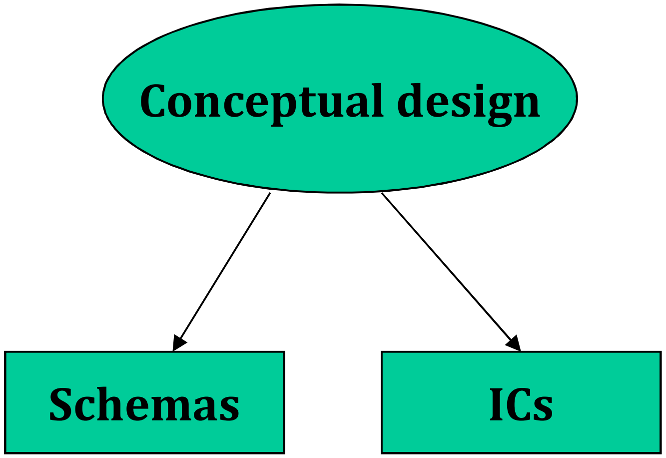
Important Concepts

- X^+ : Closure of an attribute set X
  - The set of all attributes that are determined by X
- F^+ : Closure of a dependency set F
  - The set of all dependencies that are implied from F
- $F_{min} = \{X_1 \text{->} A_1, X_2 \text{->} A_2, ..., X_n \text{->} A_n\}$ is a minimum cover of F
  - A_i is a single attribute
  - No Y in X_i can be removed such that $\{X_i - Y\}$ -> Y
  - No X_i->A_i can be removed such that $F_{min} - \{X_i \text{->} A_i\}$ can still infer X_i->A_i
- K: a key
  - A minimum set of attributes that determines all attributes

Schema Refinement and Normal Forms

- Given a design, how do we know it is good or not?
- What is the best design?
- Can a bad design be transformed into the best one? If cannot, how about 2^{nd} best? ...



Normal Form and Normalization

A relation is said to be in a particular normal form if it satisfies a certain set of constraints.

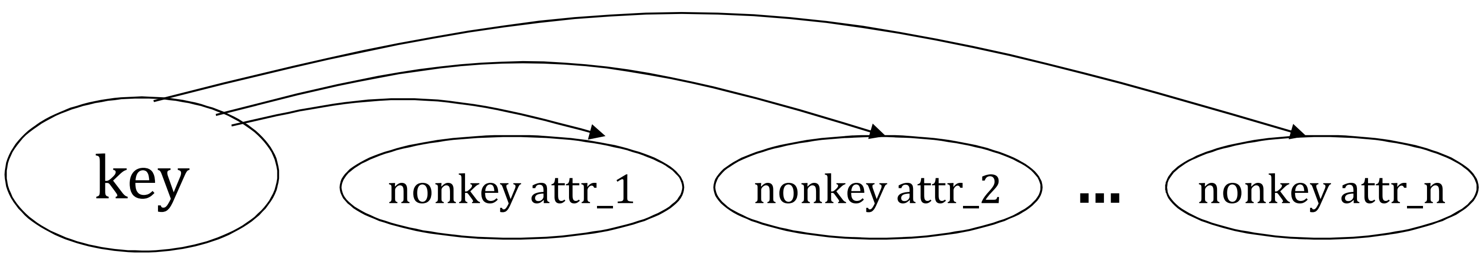
- If a relation is in a certain normal form, we know what problems it has and what problems it does not have
    - Each normal form eliminates or minimizes certain kinds of problems
- Given a relation, the process of making it to be in certain normal form is called normalization
    - Typically this is done by breaking up the relation into a set of smaller relations that possess desirable properties.

Boyce-Codd Normal Form (BCNF)

A relation R is in BCNF if whenever an FD $X \rightarrow A$ holds in R, one of the following statements is true.

- $X \rightarrow A$ is a trivial FD.
- X is a superkey.

$X \rightarrow A$ is a trivial FD if A is a subset of X



BCNF allows no data redundancy

- If there is any non-trivial FD $X \rightarrow A$, then X must be a super key, i.e., $X^+ \rightarrow R$ (all attributes in R)
- Every non-key attributes describes nothing but some aspect of the key

Primary Key

- **Key**: ***minimal*** **set of the fields of a relation that can uniquely identify a tuple**
  - **{SSN} is a key**
  - **If there's >1 key for a relation, one of the keys is chosen (by DBA) to be the primary key. The other keys are called candidate keys.**
- **Superkey: set of the fields of a relation that can uniquely identify a tuple**
  - **E.g., {SSN, Name, Age} is a superkey.**
  - **A key must be a super key, but not vice versa**

Checking for BCNF Violations

- Let F be a set of FDs
- Remove all trivial FDs in F
- For each X→A in F, check if X is a super key
  - X is a super key if X^+ contains all attributes in R

NOTE:
By default, we need to compute F^+, but actually we don't have to. If every X→A in F satisfies BCNF, then it is impossible for F^+ to have an FD Y→B such that Y→B violates BCNF

Example 1:

- Schema: Hourly_Emps (*ssn*, *name, lot, rating, hrly_wages, hrs_worked*)
- Constraints: *ssn* is the primary key and *Rating* → *hrly_wages*

Example 2:

- Schema: R(A, B, C, D)
- Constraints: A is the primary key, B is a candidate key, is R a BCNF?

An alternative definition of BCNF

Original Definition

A relation R is in BCNF if whenever an FD $X \rightarrow A$ holds in R, one of the following statements is true.

- $X \rightarrow A$ is a trivial FD.
- X is a superkey.

$\Updownarrow$

An Alternative definition

Let R be a relation and F a minimum cover. R is in BCNF if for every dependency $X \rightarrow A$ in F, X must be a key.

Checking for BCNF Violations

- Let F be a set of FDs
- Make F a minimum cover
- For each $X \rightarrow A$ in F, check if X is a key
  - X is a key if X^+ contains all attributes in R

- Let F be a set of FDs
- For each dependancy $X \rightarrow A$ in F
- Compute X^+,
- check if either $X^+=\{X\}$ or $X^+=R$

Checking for BCNF Violations

Example 1:

- Schema: Hourly_Emps (*ssn*, *name, lot, rating, hrly_wages, hrs_worked*)
- Constraints: *ssn* is the primary key and *Rating* $\rightarrow$ *hrly_wages*

Example 2:

- Schema: R(A, B, C, D)
- Constraints: A is the primary key, B is a candidate key, is R a BCNF?

BCNF is the most desirable form
--- from data redundancy perspective

A BCNF relation does not allow redundancy

- Every field of every tuple records a piece of information that cannot be inferred from the values in all other non-key fields
- If a relation is NOT in BCNF, redundancy exists

Hourly_Emps (*<u>ssn</u>, name, lot, rating, hrly_wages, hrs_worked*)

If a relation is not in BCNF, can we make it BCNF?
This process of conversion is called normalization

A Motivation Example

Constraints:

- *ssn* is the primary key
- *Rating* → *hrly_wages*

Through decomposition, we can make each table to be in BCNF.

Semantically, one table describes employee, while the other table describes wages

Hourly_Emps

SSN	Name	Lot	Rd d	W	H
123-22-3666	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_Emps2

S	N	L	R	H
123-22-3666	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

ssn is the primary key

Wages

R	W
8	10
5	7

Rating → *hrly_wages*

Normalization through Decomposition

- A decomposition of a relation schema R
  - The replacement of the schema R by two or more relation schemas, each contains a subset of R and together include all attributes of R.

- A decomposition must ensure two properties:
  - Lossless join
  - Dependency preservation

Lossless Join Decomposition

- Decomposition of R into X and Y is a *lossless-join* decomposition if every instance in R can be recovered through a nature join between X and Y

Hourly_Emps2

S	N	L	R	H
123-22-3666	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

⋈

Wages

R	W
8	10
5	7

Nature join between X and Y:

For each record x_i in X and each record y_j in Y, "union" them into a record $x_i y_i$ if they have the same value on their common attributes

SSN	Name	Lot	Rd d	W	H
123-22-3666	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

Lossless Join Decomposition: Property 1

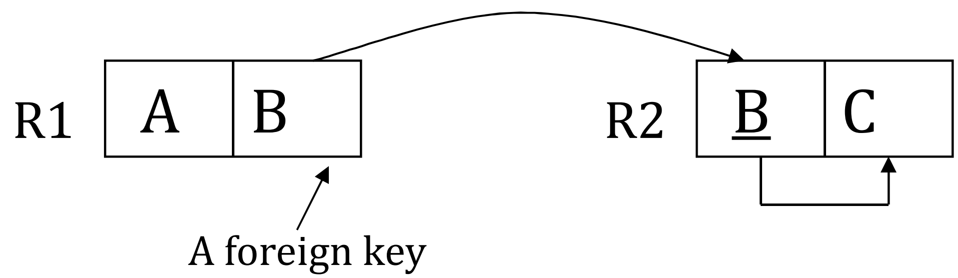
<u>Property 1</u>: A decomposition of R into R1 and R2 has the lossless join property with respect to a set of FDs F of R if and only if either

$R1 \cap R2 \rightarrow R1$ is in F^+ or

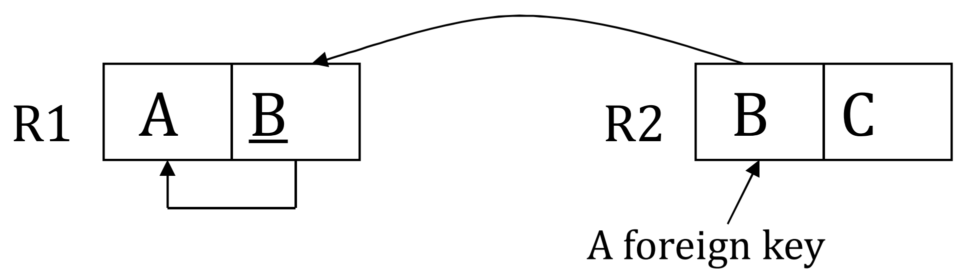
$R1 \cap R2 \rightarrow R2$ is in F^+.

The common attribute must be a super key for either R1 or R2.

Case 1:



Case 2:

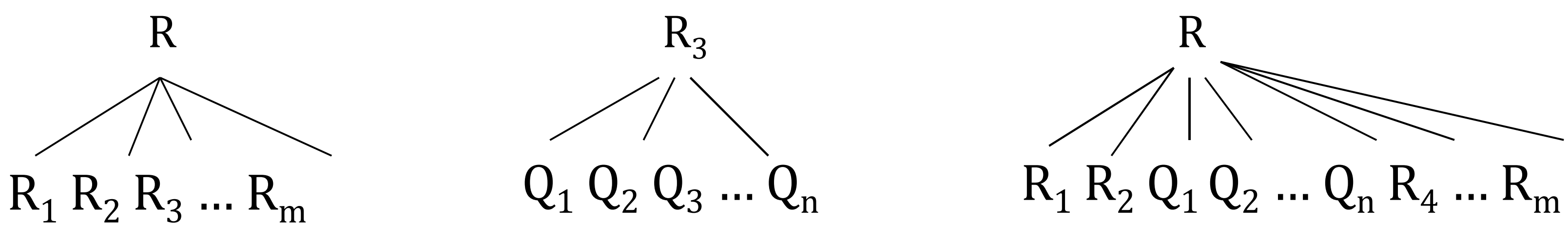


Lossless Join Decomposition: Property 2

If (1) a decomposition of R into $\{R_1, \ldots, R_m\}$ the lossless join property with respect to a set of FDs F on R, and

(2) a decomposition R_i into $\{Q_1, \ldots,$ and $Q_n\}$ has the lossless join property with respect to the projection of F on R_i.

then the decomposition of R into $\{R_1, R_2, \ldots, R_{i-1}, Q_1, \ldots, Q_n, R_{i+1}, \ldots, R_m\}$ of R has the lossless join property with respect to F.



Lossless Join Decomposition into BCNF relations

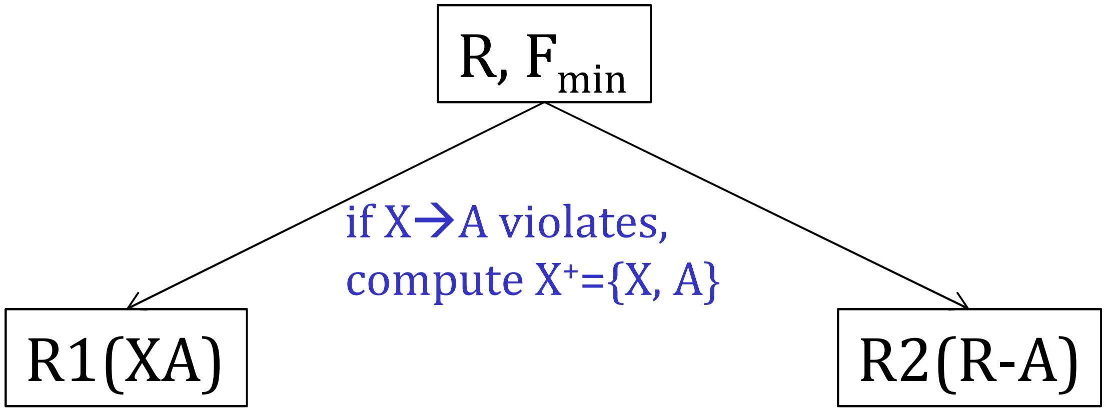
1. Set $D \leftarrow \{R\}$
2. While there is a relation schema Q in D that is not in BCNF do begin
   - Find a functional dependency $X \rightarrow Y$ in Q that violates BCNF;
   - Compute X^+
   - Replace Q in D by two schemas $(Q\text{-}X^+)$ and (X^+)

   end;

Q can be recovered by performing a nature join between $(Q\text{-}X^+)$ and (X^+). Why?

- X is their common attribute set and X^+ is a super key in one relation

Lossless Join Decomposition into BCNF relations



1. R can be recovered by joining R1 and R2, because
   - X is their common attribute set
   - X is a key in one relation
2. Check if R1 and R2 are in BCNF, decompose them if not
3. How to check Ri is in BCNF???
   - Need to know all dependencies on Ri
   - For each subset of Ri's attributes X, compute X^+
     - Either X^+=Ri (X being a super key) or
     - X^+=X (X being non-key attributes)

Example: BCNF Decomposition

Drinkers(name, addr, beersLiked, manf, favBeer)
FDs = name$\rightarrow$addr, name $\rightarrow$favBeer, beerLiked$\rightarrow$manf

1. Set D$\leftarrow$\{R\}
2. While there is a relation schema Q in D that is not in BCNF do begin
   - Find a functional dependency X$\rightarrow$Y in Q that violates BCNF;
   - Compute X^+
   - Replace Q in D by two schemas ($Q-X^+$) and (X^+)

   end;

Example: BCNF Decomposition

Drinkers(name, addr, beersLiked, manf, favBeer)
FDs = name→addr, name →favBeer, beerLiked→manf

Drinkers(name, addr, beersLiked, manf, favBeer)

- name→addr violates BCNF
- {name}+ = {name, addr, favBeer}

Drinkers1(name, addr, favBeer)

Drinkers2(name, beersLiked, manf)

- beersLiked->manf violates BCNF
- {beersLiked}+ = {name, beersLiked}

Drinkers3(beersLiked, manf)

Drinkers4(name, beersLiked)

Example: BCNF Decomposition

Drinkers(name, addr, beersLiked, manf, favBeer)
FDs = name→addr, name →favBeer, beerLiked→manf

- The resulting decomposition of Drinkers
    1. Drinkers1(name, addr, favBeer)
    2. Drinkers3(beersLiked, manf)
    3. Drinkers4(name, beersLiked)

- Notes
    - Drinkers1 tells us about drinkers,
    - Drinkers3 tells us about beers, and
    - Drinkers4 tells us the relationship between drinkers and the beers they like.

Quick Review

1. BCNF
    - For each non-trivial X→A, X is a superkey
    - Algorithms for checking BCNF
2. Lossless-join decomposition
    - Decomposition of R into X and Y is a lossless-join decomposition if every instance in R can be recovered through a nature join between X and Y
3. Algorithm for lossless-join decomposition into BCNF
    1. <u>While</u> there is a relation schema Q in D that is not in BCNF <u>do</u>
    2. <u>begin</u>
        - Choose a relation schema Q in D that is not in BCNF;
        - Find a functional dependency X→Y in Q that violates BCNF;
        - Compute X^+
        - Replace Q in D by two schemas ($Q-X^+$) and (X^+)

        <u>end</u>;

> What we have achieved so far: Given a relation R and a set of dependency FD, we can always perform lossless join decomposition on R such that each of the resulted smaller relations is in BCNF

How bout dependency?

Unfortuantely, it may be lost

Example

R(ABC), F={AB → C, C → A}

How bout dependency?

Unfortuantely, it may be lost

Example

$R(ABC)$, $F=\{AB \rightarrow C,\ C \rightarrow A\}$

- Is it in BCNF? $C \rightarrow A$ violates
- Decompose R into $R_1(CA)$ and $R_2(BC)$
  - Is lossless-join decomposition?
  - Is $AB \rightarrow C$ preserved?

If a dependency is lost, we would have to perform a join to check integrity, which is expensive.

For example, when a new record (a_i, b_i, c_i) is inserted, we need to join R_1 and R_2 in order to check if two records having the same values of a_i and b_i have same value on c_i.

We Want Dependency Preservation

- When a dependency is lost, we need to perform join operation in order to check the constraint, which is expensive!

- Unfortunately, in general, there may not be a dependency preserving decomposition into BCNF.

Dependancy Preservation

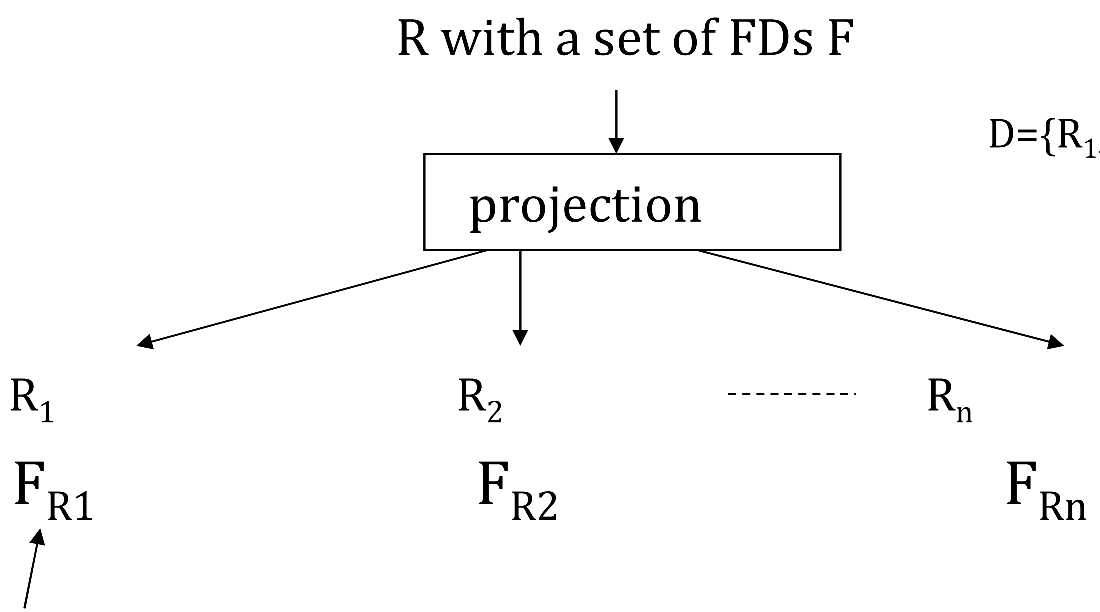
R with a set of FDs F

R_1 F_{R1} R_2 F_{R2}

Decomposition of R into R_1 and R_2 is dependency preserving if $(F_{R1} \cup F_{R2})^+ = F^+$

A Formal Definition: Dependancy Preservation

$D=\{R_1,\ldots,R_n\}$ is a decomposition of R.



The projection of F on R_i (F_{Ri}) is defined as:

$$F_{Ri} = \{X \rightarrow Y \mid X \rightarrow Y \in F^+, X \cup Y \subseteq R_i\}$$

$D=\{R_1,\ldots,R_n\}$ of R is dependency preserving with respect to F if

- $(\cup_{i=1}^{n} F_{Ri})^+ = F^+$, meaning that $(\cup_{i=1}^{n} F_{Ri})$ is equivalent to F.

THIRD NORMAL FORM (3NF)

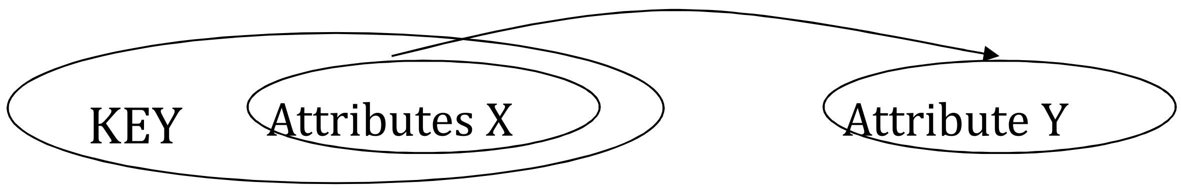
A relation R is in 3NF if whenever an FD X→A holds in R, one of the following statements is true:

- X→A is a trivial FD, or
- X is a superkey, or
- A is part of some key

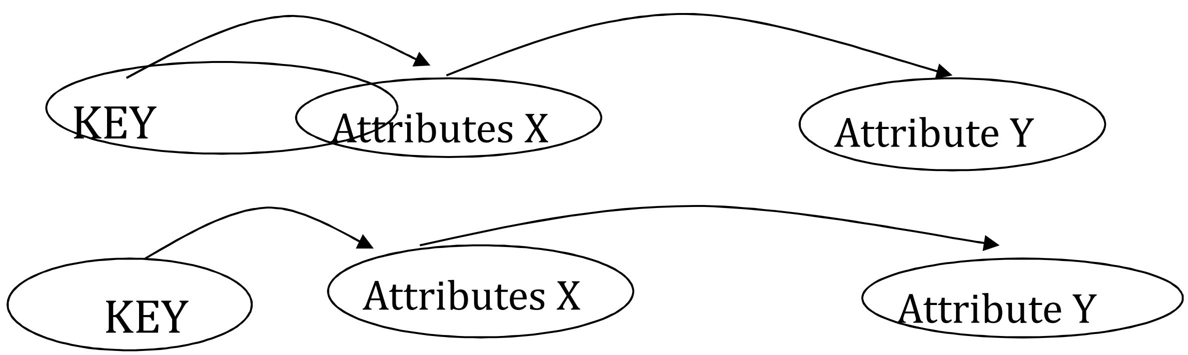
By making an exception for certain dependencies involving some key attributes, we can ensure that every relation schema can be decomposed into a collection of 3NF with two desirable properties: lossless-join and dependency-preserving.

3NF allows X→Y in two cases

- Case 1: X is a proper subset of some key (***partial dependency***)

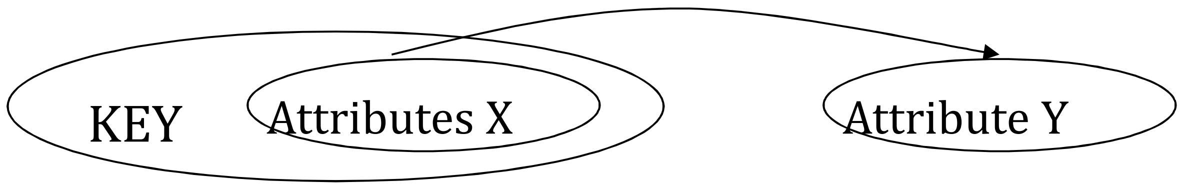


- Case 2: X is not a proper subset of any key (***transitive dependency***)

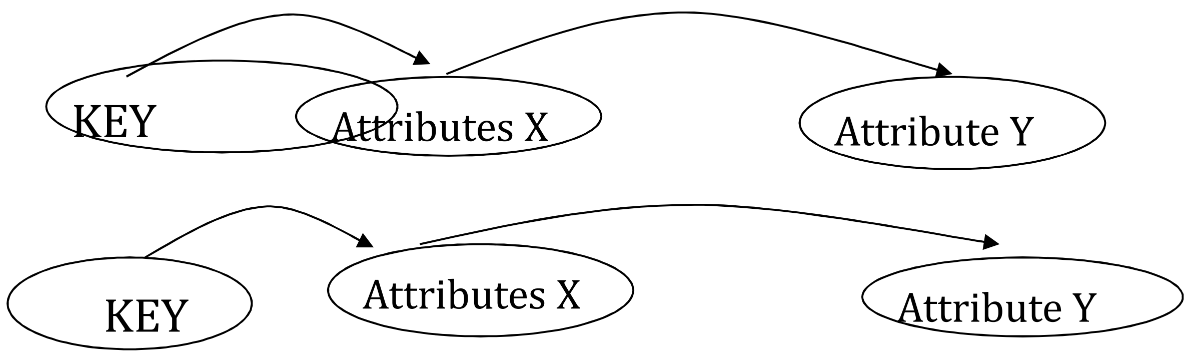


as long as Y is part of some key

What problems with them?



- Partial dependency causes data redundancy
  - Since X→Y and X is not a key, X could be redundant, so Y.



- Transitive dependency K→X→Y makes it impossible to record the values of (K, X, Y) unless all of them are known
  - Since K→X→Y, we cannot associate an X value with a K value unless we also associate an X value with Y value

Dependency-Preserving Decomposition

<u>Inputs:</u> A relation R with a set F_{min} of FDs that is a minimum cover
D(R1, R2, …, Rn) is a lossless-join decomposition of R

1. Find all dependencies in F that are not preserved
2. For each such dependency X→A, create a relation schema XA and add it to the decomposition of R
   - Every dependency in F is now preserved

<u>Proof:</u> XA is in 3rd NF

1. X must be a key for XA
   - Since X→A is in a minimal cover, Y→A does not hold for any Y that is a subset of X
2. For any other dependencies hold over XA, say Y→Z, in F_{min}, it must satisfy 3NF conditions
   - If Z is A, Y must be X
   - If Z is not A, Z must be part of X

Bottom-Up Approach (Synthesis): Lossless-Join and Dependency Preserving Decomposition into 3NF

- Given a relation R and a dependency set F
  - Find a minimum cover
  - For each dependency X-->A in F, make it a relation XA

R(ABCDE)

F={ABCD→E,E→D,A→B,AC→D}

Step 1: Find a minimum cover, G={AC→E,E→D,A→B}

Step 2: R1(<u>AC</u>E), R2(<u>E</u>D), R3(<u>A</u>B)

Step 3: Is this a lossless-join decomposition?

Top-Down Approach: Lossless-Join and Dependency Preserving Decomposition into 3NF

A. Lossless-Join Decomposition

1. Set D $\leftarrow$ {R}, and make F a minimum cover
2. <u>While</u> there is a relation schema Q in D that is not in BCNF
   - Find a functional dependency X $\rightarrow$ Y in Q that violates BCNF;
   - Replace Q in D by two schemas (Q-Y) and (XUY)

B. Dependency-Preserving Decomposition

1. Assume the decomposition is D(R1, R2, …, Rn) and the FD sets are accordingly F1, F2, …, and Fn (let their union be F')
2. For each dependency X $\rightarrow$ A in the original F (*needs to be a minimum cover*), check if it can be inferred from F'
   - If not, create a relation schema XA and add it to the decomposition of R

Exercise

R(ABCDE)
$F=\{ABCD \rightarrow E, E \rightarrow D, A \rightarrow B, AC \rightarrow D\}$

Step 1: Lossless join decomposition
Step 2: Dependency preserving decomposition

Conceptual Schema (ER diagram)

DBMS independent

DBMS specific

Data Model Mapping

Conceptual Schema (Relations)

Normalization

- BCNF/3NF?
- Decomposition
  - Lossless join
  - Dependency preservation

Determine Normal Forms

1) BCNF
   - For each X→A, is it a trivial dependency?
   - Is X a superkey?
2) 3NF
   - Suppose X→A violate BCNF
   - Is A part of some key?

Exercise 1

Indicate the strongest normal form of each of the following relations

- R1(ABCDE) F1={A→B, C→D, ACE→ABCDE}
- R2(ABCEF) F2={AB→C, B→F, F→E}

1) BCNF
   - For each X→A, is it a trivial dependency?
   - Is X a superkey?
2) 3NF
   - Suppose X→A violate BCNF
   - Is A part of some key?

Exercise 2

- Consider a relation R with five attributes: ABCDE. F={A→B, BC→E, ED→A}
  - Are {ECD}, {ACD}, {BCD} keys for R?
  - Is R in BCNF?
  - Is R in 3NF?

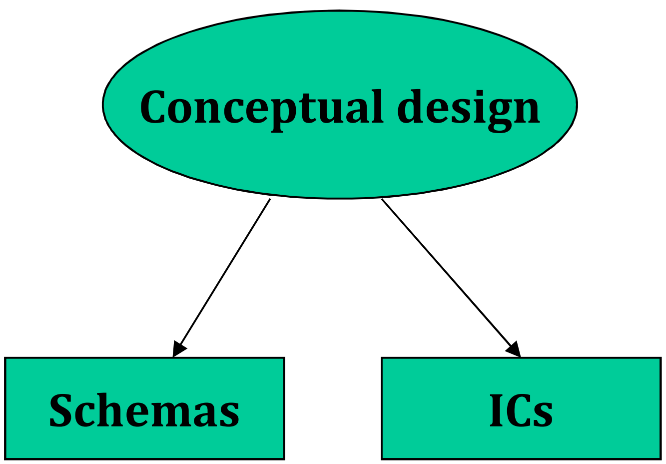
Exercise 3

- Prove that, if R is in 3NF and every key is simple (i.e, a single attribute), then R is in BCNF

Exercise 4

- Prove that, if R has only one key, it is in BCNF if and only if it is in 3NF.

Normalization Review



1. Functional Dependency
   a) Amstrong's axioms
   b) Attribute closure (A^+)
   c) Dependency closure (F^+)
   d) Minimum cover (F_{min})
2. Normal Forms
   a) BCNF
   b) 3NF
3. Decomposition
   a) Lossless join
   b) Dependency preserving