Lecture 11
Database normalization

Subject Lecturer: Kevin K.F. YUEN, PhD.

Acknowledgement: Slides were offered from Prof. Ken Yiu.
Some parts might be revised and indicated.

Slides content modified from
Database Systems Concepts

Outline

- Motivation
- Concepts: functional dependency, closure, cover
- Properties of normalization
- Normal forms: BCNF vs. 3NF

First Normal Form

- An attribute is **atomic** if it cannot be divided into parts

  Examples of non-atomic attributes

  - Set of names, composite attributes
  - Course code (CS101) that can be divided into "CS" and "101"
    → programmer may extract "CS" to find the department
    → bad idea: encode data in program rather than in DBMS

- Drawback of non-atomic attributes
  - Complicated to store them
  - Redundant storage of data → may cause data inconsistency

- In this lecture, we assume all attributes are atomic, i.e., each relation schema is in **first normal form**

Example Application: University Information Management System

- What if we use one (large) table only?
- *Problem*: redundant information

ID 	name staff name	salary staff salary	dept_name 	building dept. location	budget dept. budget
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

What if we use many (small) tables?

staff information

ID	name	street	city	salary
⋮				
57766	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

employee

ID and name

ID	name
⋮	
57766	Kim
98776	Kim
⋮	

other staff attributes

name	street	city	salary
⋮			
Kim	Main	Perryridge	75000
Kim	North	Hampton	67000
⋮			

natural join

ID	name	street	city	salary
⋮				
57766	Kim	Main	Perryridge	75000
57766	Kim	North	Hampton	67000
98776	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

Problem: lossy decomposition!

Example of Lossless-Join Decomposition

- **Lossless-join decomposition**
  - No information lost
- E.g., decompose $R = (A, B, C)$ into

  $R_1 = (A, B)$ and $R_2 = (B, C)$

r

A	B	C
x	1	A
y	2	B

decompose

$\Pi_{A,B}(r)$

A	B
x	1
y	2

$\Pi_{B,C}(r)$

B	C
1	A
2	B

$\Pi_{A,B}(r) \bowtie \Pi_{B,C}(r)$

A	B	C
x	1	A
y	2	B

How to design a good schema?

- **Decide** whether a particular relation R is in "good" form
  - E.g., BCNF, 3NF
- If a relation R is not in "good" form, **decompose** it into a set of relations $\{R_1, R_2, ..., R_n\}$ such that
  - each relation is in good form
  - the decomposition is a lossless-join decomposition
- Our theory is based *functional dependencies*

Outline

- Motivation
- Concepts: functional dependency, closure, cover
- Properties of normalization
- Normal forms: BCNF vs. 3NF

Functional dependency: $\alpha \to \beta$ generalize the concept of key

- $\alpha \to \beta$ is like a **rule**: the values of attributes in α determines **uniquely** the values of attributes in β

- *Example*: consider this instance of relation $r(A,B)$
  - $A \to B$ doesn't hold on r
  - $B \to A$ holds on r

1	4
1	5
3	7

- *Definition*: Given a relation schema R,
the **functional dependency $\alpha \to \beta$** holds on R means that:
for any legal relation $r(R)$, whenever any two tuples t_1 and t_2 of r agree on attributes in α, they also agree on attributes in β. That is,

$$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$

Express the concept of key by $\alpha \rightarrow \beta$

- K is a superkey for relation schema R if and only if $K \rightarrow R$
- K is a candidate key for R if and only if
  - $K \rightarrow R$, and
  - there exists no $\alpha \subset K$ such that $\alpha \rightarrow R$
- Functional dependencies allow us to express constraints that cannot be expressed using superkeys.
  Consider the schema:

  inst_dept (<u>*ID,*</u> *name, salary*<u>*, dept_name,*</u> *building, budget*)

  We expect these functional dependencies to hold:

  $$dept\_name \rightarrow building$$

  and $$ID \rightarrow building$$

  but would not expect the following to hold:

  $$dept\_name \rightarrow salary$$

Use of Functional Dependencies

- We use functional dependencies to:
  - **test a relation** r to see if it is legal under a given set $\mathcal{F}$ of functional dependencies
    - If yes, then we say that: r **satisfies** $\mathcal{F}$
  - **specify constraints** on the set of legal relations
    - We say that $\mathcal{F}$ **holds on** R if all legal relations on R satisfy $\mathcal{F}$
- *Note*: we ignore a functional dependency if it cannot hold on all legal instances
  - E.g., an instance of *instructor* may satisfy, by chance,

$$name \rightarrow ID$$

  but we should ignore this functional dependency

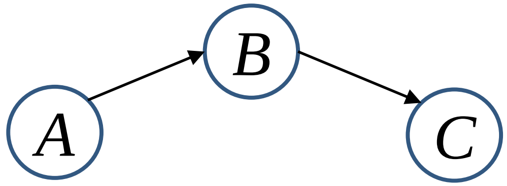
Functional Dependencies (Cont.)

- *A* functional dependency is **trivial** if it is satisfied by all instances of a relation
  - $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$
  - Example:
    - *ID*, *name* $\rightarrow$ *ID*
    - *name* $\rightarrow$ *name*

Closure of Functional Dependencies

- **Armstrong's axioms**
  - if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ **(reflexivity)**
  - if $\alpha \rightarrow \beta$, then $\gamma\,\alpha \rightarrow \gamma\,\beta$ **(augmentation)**
  - if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ **(transitivity)**

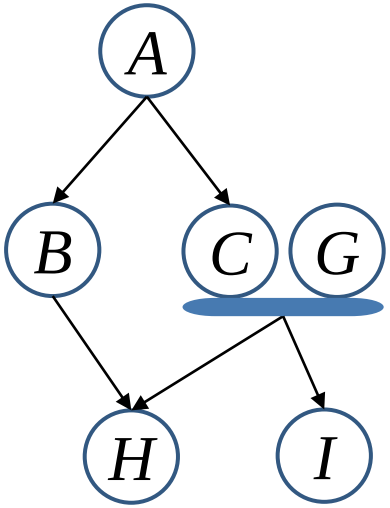
- *Example*:
  - given $A \rightarrow B$ and $B \rightarrow C$,
    we obtain: $A \rightarrow C$



- The **closure** of $\mathcal{F}$, denoted by $\mathcal{F}^+$
  - The set of **all** functional dependencies implied by a set $\mathcal{F}$ of functional dependencies
  - We compute $\mathcal{F}^+$ by repeatedly applying **Armstrong's axioms**

Example

- $R = (A, B, C, G, H, I)$
- $\mathcal{F} = \{\ A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$



$\mathcal{F}^+$ contains:

- $A \rightarrow H$
  - by transitivity from $A \rightarrow B$ *and* $B \rightarrow H$
- $AG \rightarrow I$
  - by augmenting $A \rightarrow C$ with G, to get $AG \rightarrow CG$ and then transitivity with $CG \rightarrow I$
- $CG \rightarrow HI$
  - by augmenting $CG \rightarrow I$ to infer $CG \rightarrow \mathrm{CG}I$, and augmenting of $CG \rightarrow H$ to infer $CGI \rightarrow HI$, and then transitivity
- $\ldots \rightarrow \ldots$
- until you cannot find a new functional dependency

We have a systematic way to compute $\mathcal{F}^+$ (see page 18)

Closure of Functional Dependencies (Cont.)

- Additional *short-cut* rules:
  - If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds **(union)**
  - If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds **(decomposition)**
  - If $\alpha \rightarrow \beta$ holds *a*nd $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds **(pseudotransitivity)**

The above rules can be derived from Armstrong's axioms

Attribute Set Closure

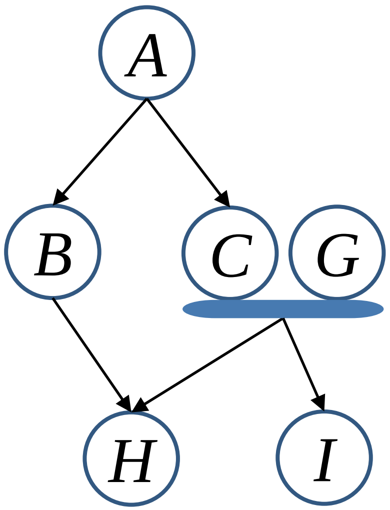
- Let α be a set of attributes
- The ***closure*** of α **under** $\mathcal{F}$ (denoted by α^+) is the set of attributes that are determined by α under $\mathcal{F}$

- Algorithm to compute α^+:

  result := α
  while (*result* has changed) **do**
      for each $\beta \rightarrow \gamma$ **in** $\mathcal{F}$ **do**
          if $\beta \subseteq$ *result* **then**
              result := *result* $\cup\ \gamma$

Example on α^+

- $R = (A, B, C, G, H, I)$
- $\mathcal{F} = \{A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$



- $(AG)^+$
  1. It contains AG
  2. It contains $ABCG$ $(A \rightarrow C$ and $A \rightarrow B)$
  3. It contains $ABCGH$ $(CG \rightarrow H)$
  4. It contains $ABCGHI$ $(CG \rightarrow I)$

  Therefore, $(AG)^+ = ABCGHI$

- Is AG a candidate key?
  - Is AG a super key?
    - Does $(AG)^+$ contain R?
  - Is any subset of AG a superkey?
    - Does $(A)^+$ contain R?
    - Does $(G)^+$ contain R?

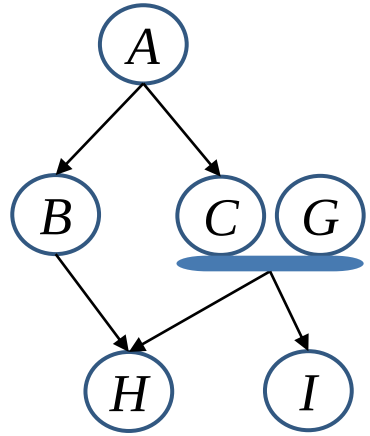
The above figure is for reference only. In general, it can be complicated visualize $\mathcal{F}$ by using a figure.

Uses of Attribute Closure

- How to test if α is a superkey?
  - Compute α^+, then check if α^+ contains all attributes of R
- How to test if a functional dependency $\alpha \rightarrow \beta$ holds?
  - (in other words, we want to check whether $\alpha \rightarrow \beta$ is in $\mathcal{F}^+$)
  - Compute α^+ by using attribute closure, and then check if α^+ contains β
- How to compute $\mathcal{F}^+$ (i.e., the closure of $\mathcal{F}$)?
  - for each $\gamma \subseteq R$
    find the closure γ^+
    for each $S \subseteq \gamma^+$
    output a functional dependency $\gamma \rightarrow S$

Example on finding all candidate keys

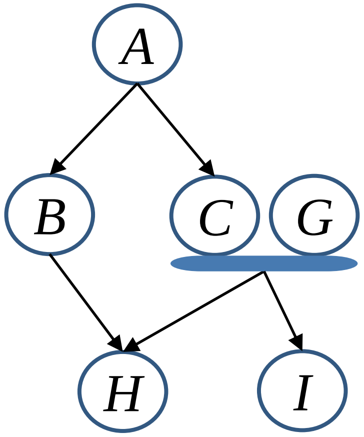
- $R = (A, B, C, G, H, I)$
- $\mathcal{F} = \{A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$



- Size-1 attribute sets
  - $A^+ = ABCH$
  - $B^+ = BH$
  - $C^+ = C$
  - $G^+ = G$
  - $H^+ = H$
  - $I^+ = I$
- Size-2 attribute sets
  - $(AB)^+ = ABCH$
  - $(AC)^+ = ABCH$
  - $(AG)^+ = ABCGHI$
    - Candidate key!
  - $(AH)^+ = ABCH$
  - $(AI)^+ = ABCHI$
  - $(BC)^+ = BCH$
  - ……
- Size-3 attribute sets except supersets of candidate keys
  - $(ABC)^+ = ABCH$
  - $\cancel{(ABG)^+}$
  - $(ABH)^+ = ABCH$
  - $(ABI)^+ = ABCHI$
  - $(ACH)^+ = ABCH$
  - ……

Example on finding $\mathcal{F}^+$ (the closure of $\mathcal{F}$)

- $R = (A, B, C, G, H, I)$
- $\mathcal{F} = \{A \to B$
  $A \to C$
  $CG \to H$
  $CG \to I$
  $B \to H\}$



- Size-1 attribute sets
  - $A \rightarrow ABCH$
    - $A \rightarrow ABC$
    - $A \rightarrow ABH$
    - $A \rightarrow ACH$
    - $A \rightarrow BCH$
    - $A \rightarrow AB$
    - $A \rightarrow AC$
    - ……
  - ……
- Size-2 attribute sets
  - $AB \rightarrow ABCH$
    - $AB \rightarrow \ldots$
    - ……
  - $AC \rightarrow ABCH$
    - ……
  - $AG \rightarrow ABCGHI$
    - ……
  - ……
- Size-3 attribute sets
  - $ABC \rightarrow ABCH$
    - $ABC \rightarrow \ldots$
    - ……
  - $ABG \rightarrow ABCGHI$
    - ……
  - $ABH \rightarrow ABCH$
    - ……
  - $ABI \rightarrow ABCHI$
    - ……
  - $ACH \rightarrow ABCH$
    - ……
  - ……

Canonical Cover

- A set of functional dependencies may have redundant dependencies that can be inferred from the others
  - *Example #1*: $A \to C$ is redundant in:

$$\{A \to B,\ \ B \to C, A \to C\}$$

  - *Example #2*: parts of a functional dependency redundant

E.g., RHS: $\{A \to B,\ \ B \to C,\ \ A \to CD\}$ can be simplified to

$$\{A \to B,\ \ B \to C,\ \ A \to D\}$$

E.g., LHS: $\{A \to B,\ \ B \to C,\ \ AC \to D\}$ can be simplified to

$$\{A \to B,\ \ B \to C,\ \ A \to D\}$$

- A canonical cover of $\mathcal{F}$ is a "**minimal**" set of functional dependencies equivalent to $\mathcal{F}$
  - no redundant (parts of) dependencies

Testing if an Attribute is Extraneous

- Consider a functional dependency $\alpha \rightarrow \beta$ in a given set $\mathcal{F}$ of functional dependencies
- To test if attribute A is extraneous in α
  - compute $(\alpha - A)^+$ using dependencies in $\mathcal{F}$
  - if $(\alpha - A)^+$ contains β, then A is extraneous in α

- *Example*: Given $\mathcal{F} = \{A \rightarrow C, AB \rightarrow C\}$
  - Is B is extraneous in $AB \rightarrow C$?

    Compute $(AB - B)^+ = A^+$

    $= AC$
  - Yes, because the above result contains C

Testing if an Attribute is Extraneous

- Consider a functional dependency $\alpha \rightarrow \beta$ in a given set $\mathcal{F}$ of functional dependencies
- To test if attribute $B \in \beta$ is extraneous in β
  - compute α^+ using only dependencies in
    $$\mathcal{F}' = (\mathcal{F} - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - B)\}$$
  - if α^+ contains B, then B is extraneous in β

- *Example*: Given $\mathcal{F} = \{A \rightarrow C, AB \rightarrow CD\}$
  - Is C extraneous in $AB \rightarrow CD$?

    Use only dependencies in $\mathcal{F}' = \{A \rightarrow C, AB \rightarrow D\}$

    Compute $(AB)^+ = ABCD$
  - Yes, because the above result contains C

Canonical Cover

- A **canonical cover** for $\mathcal{F}$ is a set of dependencies $\mathcal{F}_c$ such that
  - $\mathcal{F}$ implies all dependencies in $\mathcal{F}_c$, and
  - $\mathcal{F}_c$ implies all dependencies in $\mathcal{F}$, and
  - No functional dependency in $\mathcal{F}_c$ contains an extraneous attribute, and
  - Each left side of functional dependency in $\mathcal{F}_c$ is unique
- To compute a canonical cover for $\mathcal{F}$:

repeat

  Use the union rule to replace any dependencies in $\mathcal{F}$

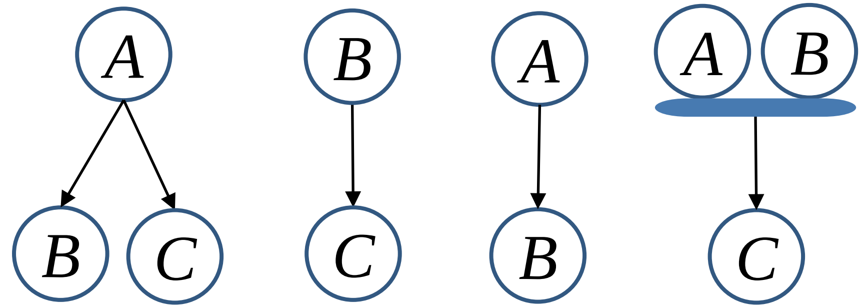
  $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ with $\alpha_1 \rightarrow \beta_1 \beta_2$

  Find a functional dependency $\alpha \rightarrow \beta$ with an extraneous attribute either in α or in β

  If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$

until $\mathcal{F}$ does not change

Example



- $R = (A, B, C)$
  $\mathcal{F} = \{A \to BC,\ B \to C$
  $A \to B,\ \ AB \to C\}$
- Combine $A \to BC$ and $A \to B$ into $A \to BC$
  - $\mathcal{F}$ becomes $\{A \to BC, B \to C, AB \to C\}$ now
- A is extraneous in $AB \to C$
  - Check if the result of deleting A from $AB \to C$ is implied by other dependencies
    - Yes: in fact, $B \to C$ is already present!
  - $\mathcal{F}$ becomes $\{A \to BC, B \to C\}$ now
- C is extraneous in $A \to BC$
  - Check if $A \to C$ is implied by $A \to B$ and other dependencies
    - Yes: using transitivity on $A \to B$ *and* $B \to$ C
      - Can use attribute closure of A in more complex cases
- The **canonical cover** is: $\{A \to B,\ B \to C\}$

Outline

- Motivation
- Concepts: functional dependency, closure, cover
- Properties of normalization
- Normal forms: BCNF vs. 3NF

Property 1: Lossless-join Decomposition

- For the case of $R = (R_1, R_2)$, we require that for all possible relations r on schema R:

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

- A decomposition of R into R_1 and R_2 is **lossless join** if at least one of the following dependencies is in F^+:

$$R_1 \cap R_2 \rightarrow R_1 \qquad or \qquad R_1 \cap R_2 \rightarrow R_2$$

Example

- $R = (A, B, C)$, $\qquad \mathcal{F} = \{A \rightarrow B, B \rightarrow C\}$
- Decompose into $R_1 = (A, B)$, $R_2 = (B, C)$
  - Lossless because: $R_1 \cap R_2 = \{B\}$ and $B \rightarrow BC$
- Decompose into $R_1 = (A, B)$, $R_2 = (A, C)$
  - Lossless because: $R_1 \cap R_2 = \{A\}$ and $A \rightarrow AB$

Property 2: Dependency Preservation

- Let $\mathcal{F}_i$ be the set of dependencies $\mathcal{F}^+$ that include only attributes in R_i
- A decomposition is **dependency preserving** if

$$(\mathcal{F}_1 \cup \mathcal{F}_2 \cup \ldots \cup \mathcal{F}_n)^+ = \mathcal{F}^+$$

  - If it is not, then checking updates for violation of functional dependencies may require computing joins, which is expensive

Example

- $R = (A, B, C), \qquad \mathcal{F} = \{A \rightarrow B, B \rightarrow C\}$
- Decompose into $R_1 = (A, B), \quad R_2 = (B, C)$
  - Dependency preserving
- Decompose into $R_1 = (A, B), \quad R_2 = (A, C)$
  - Not dependency preserving
    (cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

Goals of Normalization

- If a relation scheme R is not in "good" form, decompose it into $\{R_1, R_2, ..., R_n\}$ such that
  - each relation scheme is in "good" form, and
  - the decomposition is a lossless-join decomposition
  - [*preferably*] the decomposition should be dependency preserving

Outline

- Motivation
- Concepts: functional dependency, closure, cover
- Properties of normalization
- Normal forms: BCNF vs. 3NF

Boyce-Codd Normal Form (BCNF)

A relation schema R is in **BCNF** with respect to a set $\mathcal{F}$ of functional dependencies if for all functional dependencies in $\mathcal{F}^+$ of the form

$$\alpha \rightarrow \beta$$

where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

- $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
- α is a superkey for R

Example schema *not* in BCNF:

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

because *dept_name*→ *building, budget* holds on *instr_dept*,
but *dept_name* is not a superkey

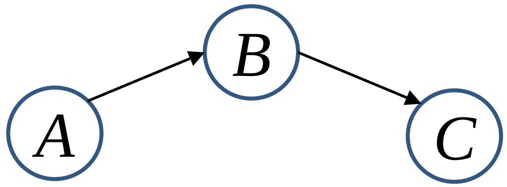
BCNF Decomposition Algorithm

result := $\{R\}$
compute $\mathcal{F}^+$
while (some schema R_i in *result* is not in BCNF) **do**
  let $\alpha \to \beta$ be a nontrivial functional dependency that
    holds on R_i such that $\alpha \to R_i$ is not in $\mathcal{F}^+$,
    and $\alpha \cap \beta = \varnothing$
  $result := (result - R_i) \cup (R_i - \beta) \cup (\alpha, \beta)$

Note: each R_i is in BCNF, and
decomposition is lossless-join

Example of BCNF Decomposition

- $R = (A, B, C)$
  $\mathcal{F} = \{A \rightarrow B$
  $B \rightarrow C\}$
  Key = $\{A\}$



- R is not in BCNF
  ($B \rightarrow C$ but B is not superkey)
- Decompose R into: $R_1 = (A, B)$, $R_2 = (B, C)$
  - R_1 and R_2 are in BCNF
  - Lossless-join decomposition

Example of BCNF Decomposition

- *class* (*course_id*, *title*, *dept_name*, *credits*, *sec_id*, *semester*, *year*, *building*, *room_number*, *capacity*, *time_slot_id*)
- Given functional dependencies:
  - *course_id*→ *title*, *dept_name*, *credits*
  - *building*, *room_number*→*capacity*
  - *course_id*, *sec_id*, *semester*, *year*→*building*, *room_number*, *time_slot_id*
- Find a candidate key: (*course_id*, *sec_id*, *semester*, *year*)
- BCNF Decomposition:
  - *course_id*→ *title*, *dept_name*, *credits* holds
    - but *course_id* is not a superkey
  - We replace *class* by:
    - *course*(*course_id*, *title*, *dept_name*, *credits*)
    - *class-1* (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *capacity*, *time_slot_id*)

BCNF Decomposition (Cont.)

- *course* is in BCNF
  - How do we know this?

- *building*, *room_number*→*capacity* holds on *class-1*
  - but {*building*, *room_number*} is not a superkey for *class-1*
  - We replace *class-1* by:
    - *classroom* (*building*, *room_number*, *capacity*)
    - *section* (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *time_slot_id*)

- *classroom* and *section* are in BCNF

Third Normal Form (3NF)

- A relation schema R is in **3NF** if for all:

$$\alpha \rightarrow \beta \text{ in } \mathcal{F}^{+}$$

  at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \in \alpha$)
  - α is a superkey for R
  - Each attribute A in $\beta - \alpha$ is contained in a candidate key for R

  (**NOTE**: each attribute may be in a different candidate key)

- A relation is in BCNF → it is in 3NF
  - Because BCNF requires one of the first two conditions
- The third condition is a relaxation of BCNF
  - Ensure dependency preservation

3NF Example

- Relation *dept_advisor*:
  - *dept_advisor* (s_ID, i_ID, $dept\_name$)
    $\mathcal{F} = \{s\_ID, dept\_name \rightarrow i\_ID,\ i\_ID \rightarrow dept\_name\}$
  - Two candidate keys: (s_ID, $dept\_name$), and (i_ID, s_ID)
  - R is in 3NF
    - $s\_ID, dept\_name \rightarrow i\_ID \quad s\_ID$
      - s_ID, $dept\_name$ is a superkey
    - $i\_ID \rightarrow dept\_name$
      - $dept\_name$ is contained in a candidate key

3NF Decomposition Algorithm

let $\mathcal{F}_c$ be a canonical cover for $\mathcal{F}$

$i := 0$

for each functional dependency $\alpha \rightarrow \beta$ in $\mathcal{F}_c$ **do**

if no schema R_j, $1 \leq j \leq i$ contains $\alpha\ \beta$ **then**

$i := i + 1$

$R_i := \alpha\ \beta$

if no schema R_j, $1 \leq j \leq i$ contains a candidate key for R **then**

$i := i + 1$

R_i := any candidate key for R

/* remove redundant relations */

while (some schema R_j is contained in another schema R_k)

$R_j := R_i$

$i := i - 1$

return $(R_1, R_2, ..., R_i)$

It ensures:

- each schema R_i is in 3NF
- decomposition is dependency preserving and lossless-join

3NF Decomposition: An Example

- Relation schema:

$cust\_banker\_branch = (\underline{customer\_id, employee\_id}, branch\_name, type)$

- Given functional dependencies:
  - $customer\_id, employee\_id \rightarrow branch\_name, type$
  - $employee\_id \rightarrow branch\_name$
  - $customer\_id, branch\_name \rightarrow employee\_id$
- We first compute a **canonical cover**
  - *branch_name* is extraneous in the r.h.s. of the 1^{st} dependency
  - No other attribute is extraneous, so we get $\mathcal{F}_C =$

$$customer\_id, employee\_id \rightarrow type$$
$$employee\_id \rightarrow branch\_name$$
$$customer\_id, branch\_name \rightarrow employee\_id$$

3NF Decompsition Example (Cont.)

- The **for** loop generates following 3NF schema:

  (*customer_id*, *employee_id*, *type*)

  (*<u>employee_id</u>*, *branch_name*)

  (*customer_id*, *branch_name*, *employee_id)*

  - Observe that (*customer_id*, *employee_id*, *type*) contains a candidate key of the original schema, so no further relation schema needs to be added
- At end of for loop, detect and delete schemas, such as (*<u>employee_id</u>*, *branch_name*), which are subsets of other schemas
  - result will not depend on the order in which FDs are considered
- The resultant simplified 3NF schema is:

  (*customer_id*, *employee_id*, *type*)

  (*customer_id*, *branch_name*, *employee_id)*

BCNF vs. 3NF

- We can always decompose a relation into a set of relations that are in 3NF such that:
  - the decomposition is lossless
  - the dependencies are preserved

- We can always decompose a relation into a set of relations that are in BCNF such that:
  - the decomposition is lossless
  - but it may not preserve dependencies

BCNF vs. 3NF

- First, we try to satisfy all requirements:
  - BCNF
  - Lossless join
  - Dependency preservation
- If we cannot achieve this, we accept one of:
  - Lack of dependency preservation, or
  - Redundancy due to the use of 3NF

Appendix: BCNF and Dependency Preservation

It is not always possible to get a BCNF decomposition that is dependency preserving

- $R = (J, K, L)$
  $\mathcal{F} = \{JK \rightarrow L$
  $\qquad L \rightarrow K\}$
  Two candidate keys = JK and JL
- R is not in BCNF
- Any decomposition of R will fail to preserve

$$JK \rightarrow L$$

This implies that testing for $JK \rightarrow L$ requires a join

Appendix: Redundancy in 3NF

- There is some redundancy in this schema
- Example of problems due to redundancy in 3NF
  - $R = (J, K, L)$
    $\mathcal{F} = \{JK \rightarrow L, L \rightarrow K\}$

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

- repetition of information (e.g., the relationship l_1, k_1)
  - *(i_ID, dept_name)*
- need to use null values (e.g., to represent the relationship l_2, k_2 where there is no corresponding value for J).
  - *(i_ID, dept_nameI)* if there is no separate relation mapping instructors to departments