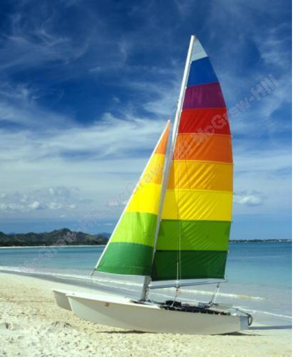

Normalization

Database System Concepts, 6th Ed.

Student_Course_Result Table

Student_Details			Course_Details				Result_Details		
101	Davis	11/4/1986	M4	Applied Mathematics	Basic Mathematics	7	11/11/2004	82	A
102	Daniel	11/6/1987	M4	Applied Mathematics	Basic Mathematics	7	11/11/2004	62	C
101	Davis	11/4/1986	H6	American History		4	11/22/2004	79	B
103	Sandra	10/2/1988	C3	Bio Chemistry	Basic Chemistry	11	11/16/2004	65	B
104	Evelyn	2/22/1986	B3	Botany		8	11/26/2004	77	B
102	Daniel	11/6/1987	P3	Nuclear Physics	Basic Physics	13	11/12/2004	68	B
105	Susan	8/31/1985	P3	Nuclear Physics	Basic Physics	13	11/12/2004	89	A
103	Sandra	10/2/1988	B4	Zoology		5	11/27/2004	54	D
105	Susan	8/31/1985	H6	American History		4	11/22/2004	87	A
104	Evelyn	2/22/1986	M4	Applied Mathematics	Basic Mathematics	7	11/11/2004	65	B

The need for Normalization

Insert Anomaly

We cannot insert prospective course which does not have any registered student or we can not insert student details who is yet to register for any course.

Update Anomaly

If we want to update the course M4's name we need to do this operation three times. Similarly we may have to update student 103' s name twice if it changes.

Delete Anomaly

If we want to delete a course M4, in addition to M4 course details, other critical details of student also will be deleted. This kind of deletion is harmful to business. Moreover, M4 appears thrice in above table and needs to be deleted thrice.

Duplicate Data

Course M4's data is stored thrice and student 102' s data stored twice. This redundancy will increase as the number of course offering and students increases.

Larger Schemas?

- Suppose we combine *instructor* and *department* into *inst_dept*
  - *(No connection to relationship set inst_dept)*
- Result is possible repetition of information

ID	*name*	*salary*	*dept_name*	*building*	*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

Larger Schemas?

ID-> ID, name, salary, Dept_name, building, budget

Dept_name->Dept_name, building, budget

Dept_name, name -> Name

ID	*name*	*salary*	*dept_name*	*building*	*budget*

R = ID, name, salary, Dept_name, building, budget

Dept_name->building, budget

R1 = ID, name, salary, Dept_name

R2 = Dept_name,building, budget

What About Smaller Schemas?

- Suppose we had started with *inst_dept.* How would we know to split up (**decompose**) it into *instructor* and *department*?
- Write a rule "if there were a schema (*dept_name, building, budget*), then *dept_name* would be a candidate key"
- Denote as a **functional dependency**:

  $$dept\_name \rightarrow building,\ budget$$

- In *inst_dept*, because *dept_name* is not a candidate key, the building and budget of a department may have to be repeated.
  - This indicates the need to decompose *inst_dept*

After Decomposition

Instructor

Instructor_Department

ID	*name*	*salary*	*dept_name*	*building*	*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

Department

building	*budget*
Watson	70000
Painter	120000
Painter	50000
Taylor	100000
Taylor	85000
Watson	90000
Packard	80000

What About Smaller Schemas?

- Suppose we had started with *inst_dept.* How would we know to split up (**decompose**) it into *instructor* and *department*?
- Write a rule "if there were a schema (*dept_name, building, budget*), then *dept_name* would be a candidate key"
- Denote as a **functional dependency**:

  $$dept\_name \rightarrow building, budget$$

- In *inst_dept*, because *dept_name* is not a candidate key, the building and budget of a department may have to be repeated.
  - This indicates the need to decompose *inst_dept*
- **Not all decompositions are good**. Suppose we decompose *employee(ID, name, street, city, salary)* into

  employee1 (*ID*, *name*)

  employee2 (*name*, *street, city, salary*)
- The next slide shows how we lose information -- we cannot reconstruct the original *employee* relation -- and so, this is a **lossy decomposition**.

A Lossy Decomposition

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

employee

ID	*name*
⋮	
57766	Kim
98776	Kim
⋮	

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
⋮				

First Normal Form

- Domain is **atomic** if its elements are considered to be indivisible units
  - Examples of non-atomic domains:
    - Set of names, composite attributes
    - Identification numbers like CS101 that can be broken up into parts
- A relational schema R is in **first normal form** if the domains of all attributes of R are atomic
- Non-atomic values complicate storage and encourage redundant (repeated) storage of data
  - Example: Set of accounts stored with each customer, and set of owners stored with each account
  - We assume all relations are in first normal form (and revisit this in Chapter 22: Object Based Databases)

First Normal Form (Cont'd)

- Atomicity is actually a property of how the elements of the domain are used.
  - Example: Strings would normally be considered indivisible
  - Suppose that students are given roll numbers which are strings of the form *CS0012* or *EE1127*
  - If the first two characters are extracted to find the department, the domain of roll numbers is not atomic.
  - Doing so is a bad idea: leads to encoding of information in application program rather than in the database.

Goal — Devise a Theory for the Following

- Decide whether a particular relation R is in “good” form.
- In the case that 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 on:
  - functional dependencies
  - multivalued dependencies

Functional Dependencies

- Constraints on the set of legal relations.
- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes.
- **A functional dependency is a generalization of**
  - **the notion of a *key.***

Functional Dependencies (Cont.)

- Let *R* be a relation schema

$$\alpha \subseteq R \;\; and \;\; \beta \subseteq R$$

- The **functional dependency**

$$\alpha \rightarrow \beta$$

  holds on *R* if and only if for any legal relations *r*(R), whenever any two tuples t_1 and t_2 of *r* agree on the attributes α, they also agree on the attributes β. That is,

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

Functional Dependencies (Cont.)

- Let R be a relation schema

$$\alpha \subseteq R \ \ and \ \ \beta \subseteq R$$

- The **functional dependency**

$$\alpha \rightarrow \beta$$

holds on R if and only if for any legal relations r(R), whenever any two tuples t_1 and t_2 of r agree on the attributes α, they also agree on the attributes β. That is,

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

- **Example:** Consider r(A,B) with the following instance of r.

1	4
1	5
3	7

- On this instance, $A \rightarrow B$ does **NOT** hold, but $B \rightarrow A$ does hold.

Functional Dependencies (Cont.)

- ***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
  - for no $\alpha \subset K$, $\alpha \rightarrow R$

Functional Dependencies (Cont.)

- *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
  - for no $\alpha \subset K$, $\alpha \rightarrow R$
- Functional dependencies allow us to express constraints that cannot be expressed using superkeys. Consider the schema:

  inst_dept (*ID, name, salary, dept_name, 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:

  $$dept\_name \rightarrow salary$$

Functional Dependencies (Cont.)

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

Closure of a Set of Functional Dependencies

- Given a set *F* of functional dependencies, there are certain other functional dependencies that are logically implied by *F*.
  - For example: If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
- The set of **all** functional dependencies logically implied by *F* is the **closure** of *F*.
- We denote the *closure* of *F* by $\mathbf{F^+}$.
- F^+ is a superset of *F*.

Boyce-Codd Normal Form

A relation schema R is in BCNF with respect to a set F of functional dependencies if for all functional dependencies in 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

Boyce-Codd Normal Form

A relation schema *R* is in BCNF with respect to a set *F* of functional dependencies if for all functional dependencies in 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 (*<u>ID</u>, name, salary, dept_name, building, budget*)

because *dept_name* $\rightarrow$ *building, budget*
holds on *instr_dept,* but *dept_name* is not a superkey

Decomposing a Schema into BCNF

- Suppose we have a schema *R* and a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF.

  We decompose *R* into:
  - $(\alpha \cup \beta)$
  - $(R - (\beta - \alpha))$

Decomposing a Schema into BCNF

- Suppose we have a schema *R* and a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF.

  We decompose *R* into:
  - $(\alpha \cup \beta)$
  - $(R - (\beta - \alpha))$

- In our example,

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

  - α = *dept_name*
  - β = *building, budget*

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

Problem 1

1. Consider R(XYZ) and Functional dependencies = {XY → Z, and Z → Y}

Is R in BCNF? If not normalize R into BCNF

To check if R is in BCNF, check if all the FD falls into one of the category

- FD is trivial
- Attributes on the Left hand side is a superkey

Consider FD XY → Z

Its not trivial so check if XY is a superkey

(XY)+

Result = XYZ

Since XY -> XYZ

XY is a superkey

Consider FD Z → Y

It is not trivial so check if Z is superkey

Problem 1 (contd.)

Consider FD $Z \rightarrow Y$

It is not trivial so check if Z is superkey

$(Z)+$ Result = ZY

Z is not a superkey

Hence because of FD $Z \rightarrow Y$, R is not in BCNF

We decompose *R* into

$R1 = (\alpha \cup \beta)$

$R2 = (R - (\beta - \alpha))$

R1 = YZ

R2 = (XYZ – (Z- Y)) =(XYZ-Z) =XY

Only non trivial FD that is valid is $Z \rightarrow Y$ for R1

There is no non trivial FD for R2

Problem

- Consider R(A,B,C,D,E,F,G,H)

  with FDs: {{AB→ C,D,E,F}, {F→G,H}, {B→ C,D}}

  Find key for R? Is R in BCNF? If not normalize R into BCNF

. F→G,H violates BCNF

with FDs: {{AB→ C,D,E,F}, {F→G,H}, B→ C,D, }}

R1 = ABCDEF

R2 = FGH

R1 is not in BCNF because of B→ C,D

R11=ABEF

R12=BCD

R2 = FGH

Closure of Attribute Sets

- Given a set of attributes α, define the ***closure*** of α **under** *F* (denoted by α^+) as the set of attributes
  - that are functionally determined by α under *F*

- Algorithm to compute α^+, the closure of α under *F*

> *result* := α;
> **while** (changes to *result*) **do**
>   **for each** $\beta \rightarrow \gamma$ **in** *F* **do**
>   **begin**
>   **if** $\beta \subseteq$ *result* **then** *result* := *result* $\cup\ \gamma$
>   **end**

Example of Attribute Set Closure

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$
- $(AG)^+$
  1. $result = AG$
  2. $result = ABCG$ $(A \rightarrow C$ and $A \rightarrow B)$
  3. $result = ABCGH$ $(CG \rightarrow H$ and $CG \subseteq AGBC)$
  4. $result = ABCGHI$ $(CG \rightarrow I$ and $CG \subseteq AGBCH)$
- **Is *AG* a candidate key?**
  1. Is AG a super key?
     1. Does $AG \rightarrow R$? == Is $(AG)^+ \supseteq R$
  2. Is any subset of AG a superkey?
     1. Does $A \rightarrow R$? == Is $(A)^+ \supseteq R$
     2. Does $G \rightarrow R$? == Is $(G)^+ \supseteq R$

Canonical Cover

- A **canonical cover** for F is a set of dependencies F_c such that
  - F logically implies all dependencies in F_c, and
  - F_c logically implies all dependencies in F, and
  - No functional dependency in F_c contains an extraneous attribute, and
  - Each left side of functional dependency in F_c is unique.
- **To compute a canonical cover for *F*:**
  repeat
  Use the union rule to replace any dependencies in 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
  /* Note: test for extraneous attributes done using F_c, not F*/
  If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$
  until F does not change
- **Note:** Union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied

Canonical Cover

- Sets of functional dependencies may have redundant dependencies that can be inferred from the others
  - For example: $A \rightarrow C$ is redundant in: $\{A \rightarrow B,\ \ B \rightarrow C,\ A \rightarrow C\}$
  - **Parts of a functional dependency may be redundant**
    - E.g.: on RHS: $\{A \rightarrow B,\ \ B \rightarrow C,\ \ A \rightarrow CD\}$ can be simplified to
      $$\{A \rightarrow B,\ \ B \rightarrow C,\ \ A \rightarrow D\}$$
    - E.g.: on LHS: $\{A \rightarrow B,\ \ B \rightarrow C,\ \ AC \rightarrow D\}$ can be simplified to
      $$\{A \rightarrow B,\ \ B \rightarrow C,\ \ A \rightarrow D\}$$

- Intuitively, a canonical cover of F is a **"minimal" set of functional dependencies** equivalent to F,
  - having no redundant dependencies or **redundant parts of dependencies**

Extraneous Attributes

- Consider a set F of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in F.
  - Attribute A is **extraneous** in α if $A \in \alpha$ and F logically implies $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$.
  - Attribute A is **extraneous** in β if $A \in \beta$ and the set of functional dependencies $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ logically implies F.
- ***Note:*** implication in the opposite direction is trivial in each of the cases above, **since a "stronger" functional dependency always implies a weaker one**
- **Example: Given $F = \{A \rightarrow C, AB \rightarrow C\}$**
  - B is extraneous in $AB \rightarrow C$ because $\{A \rightarrow C, AB \rightarrow C\}$ logically implies $A \rightarrow C$ (I.e. the result of dropping B from $AB \rightarrow C$).
- **Example: Given $F = \{A \rightarrow C, AB \rightarrow CD\}$**
  - C is extraneous in $AB \rightarrow CD$ since $AB \rightarrow C$ can be inferred even after deleting C

Testing if an Attribute is Extraneous

- Consider a set F of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in F.
- To test if attribute $A \in \alpha$ is extraneous in α
  1. compute $(\{\alpha\} - A)^+$ using the dependencies in F
  2. check that $(\{\alpha\} - A)^+$ contains β; if it does, A is extraneous in α
- To test if attribute $A \in \beta$ is extraneous in β
  1. compute α^+ using only the dependencies in
     $$F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\},$$
  2. check that α^+ contains A; if it does, A is extraneous in β

Example

Consider suppose F contains AB → CD, A → E, D → C.

- **Check if B is extraneous attribute in AB → CD.**

  A+ = AE

  B is not extraneous

- **Check if C is extraneous attribute in AB → CD.**

  F’ = {AB → D, A → E, D → C}

  (AB)+ = ABDEC

  C is extraneous

Computing a Canonical Cover

- $R = (A, B, C)$
  $F = \{A \rightarrow BC$
  $B \rightarrow C$
  $A \rightarrow B$
  $AB \rightarrow C\}$

Computing a Canonical Cover

Solution:

- Apply Union rule

  $F_c = \{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$
- Check if B is extraneous in $A \rightarrow BC$

  $F' = \{A \rightarrow C, B \rightarrow C, AB \rightarrow C\}$

  A+ =AC since B is not present in A+, B is not extraneous
- Check if C is extraneous in $A \rightarrow BC$

  $F_c' = \{A \rightarrow B, B \rightarrow C, AB \rightarrow C\}$

  A+ = ABC

  C is extraneous

  $F_c = \{A \rightarrow B, B \rightarrow C, AB \rightarrow C\}$
- Check if C is extraneous in $B \rightarrow C$

  $F_c' = \{A \rightarrow B, AB \rightarrow C\}$

  B+ =B

  C is not extraneous

  $F_c = \{A \rightarrow B, B \rightarrow C, AB \rightarrow C\}$
- Check if A is extraneous in $AB \rightarrow C$

  B+ = BC

  A is extraneous

  $F_c = \{A \rightarrow B, B \rightarrow C, B \rightarrow C\}$
- Apply union rule

  $F_c = \{A \rightarrow B, B \rightarrow C\}$

Find the canonical cover

R={A,B,C}

F={A->B, AB->C}

Find the candidate key

R={A,B,C}

F={A->B, AB->C}

Candidate Key

Necessary attributes:
An attribute A is said to be a necessary attribute if

(a) A occurs only in the L.H.S. (left hand side) of the fd's in F; or

(b) A is an attribute in relation R, but A does not occur in either L.H.S. or R.H.S. of any fd in F.

In other words, necessary attributes NEVER occur in the R.H.S. of any fd in F.

Useless attributes:
An attribute A is a useless attribute if A occurs ONLY in the R.H.S. of fd's in F.

Middle-ground attributes:
An attribute A in relation R is a middle-ground attribute if A is neither necessary nor useless.

Example.
Consider the relation R(ABCDEG) with set of fd's $F = \{AB \rightarrow C, C \rightarrow D, AD \rightarrow E\}$

Necessary	Useless	Middle-ground
A, B, G	E	C, D

Candidate Key

The algorithm for computing all candidate keys of R.

Input: A relation R={ A_1, A_2, ..., A_n}, and F, a set of functional dependencies.
Output: K={ $K_1, \ldots, K_t$}, the set of all candidate keys of R.

Step1.
Set F′ to a minimal cover of F (This is needed because otherwise we may not detect all useless attributes).

Step2.
Partition all attributes in R into necessary, useless and middle-ground attribute sets according to F′. Let X={ $C_1, \ldots, C_l$} be the necessary attribute set, Y = {B_1, ..., B_k} be the useless attribute set, and M = {A_1, ..., A_n} – (X ∪ Y) be the middle-ground attribute set.
If X={}, then go to step4.

Step3.
Compute X^+. If X^+ =R, then set K= {X}, terminate.

Candidate Key

Step4.

Let L = $\langle Z_1, Z_2, \ldots, Z_m \rangle$ be the list of all non-empty subsets of M (the middle-ground attributes) such that L is arranged in ascending order of the size of Z_i.

Add all attributes in X (necessary attributes) to each Z_i in L.

Set K = {}.

i $\leftarrow$ 0.

WHILE L $\neq$ empty do

BEGIN

i $\leftarrow$ i + 1.

Remove the first element Z from L.

Compute Z^+.

If Z^+ = R,

then

begin

set K $\leftarrow$ K $\cup$ {Z};

for any $Z_j \in$ L, if $Z \subset Z_j$

then L $\leftarrow$ L $-$ {Z_j}.

end

END

Example

Example. (Computing all candidate keys of R.)
Let R = R(ABCDEG) and F = {AB → CD, A → B, B → C, C → E, BD → A}. The process to compute all candidate keys of R is as follows:

(1) The minimal cover of F is {A → B, A → D, B → C, C → E, BD → A}.

(2) Since attribute G never appears in any fd's in the set of functional dependencies, G must be included in a candidate key of R. The attribute E appears only in the right hand side of fd's and hence E is not in any key of R. No attribute of R appears only in the left hand side of the set of fd's. Therefore X = G at the end of step 2.

(3) Compute G^+ = G, so G is not a candidate key.

(4) The following table shows the L, K, Z and Z^+ at the very beginning of each iteration in the **WHILE** statement.

Candidate Key

- Let R = R(ABCDEG) and F = {AB → CD, A → B, B → C, C → E, BD → A}.
- Candidate Keys are {AG, BDG}

Example

i	Z	Z^+	L	K
0	–	–	⟨AG, BG, CG, DG, ABG, ACG, ADG, BCG, BDG, CDG, ABCG, ABDG, ACDG, BCDG, ABCDG⟩	{}
1	AG	ABCDEG = R	⟨BG, CG, DG, BCG, BDG, CDG, BCDG⟩	{AG}
2	BG	BCEG ≠ R	⟨CG, DG, BCG, BDG, CDG, BCDG⟩	{AG}
3	CG	CEG ≠ R	⟨DG, BCG, BDG, CDG, BCDG⟩	{AG}
4	DG	DG ≠ R	⟨BCG, BDG, CDG, BCDG⟩	{AG}
5	BCG	BCEG ≠ R	⟨BDG, CDG, BCDG⟩	{AG}
6	BDG	ABCDEG = R	⟨CDG⟩	{AG, BDG}
7	CDG	CEDG ≠ R	⟨ ⟩	{AG, BDG}

Third Normal Form

- A relation schema R is in **third normal form (3NF)** if for all:

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

  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)

- If a relation is in BCNF it is in 3NF (since in BCNF one of the first two conditions above must hold).
- Third condition is a minimal relaxation of BCNF
  - to ensure dependency preservation (will see why later).

3NF Decomposition Algorithm

Let F_c be a canonical cover for F;
$i := 0$;
for each functional dependency $\alpha \rightarrow \beta$ in F_c **do**
 if none of the schemas R_j, $1 \leq j \leq i$ contains $\alpha\ \beta$
  then begin
   $i := i + 1$;
   $R_i := \alpha\ \beta$
  end
if none of the schemas R_j, $1 \leq j \leq i$ contains a candidate key for R
 then begin
   $i := i + 1$;
   R_i := any candidate key for R;
  end
/* Optionally, remove redundant relations */

repeat
if any schema R_j is contained in another schema R_k
  then /* delete R_j ***/**
   $R_j = R$;;
   $i=i-1$;
return $(R_1, R_2, ..., R_i)$

3NF Decomposition Algorithm (Cont.)

- Above algorithm ensures:
  - each relation schema R_i is in 3NF
  - decomposition is dependency preserving and lossless-join
  - Proof of correctness is at end of this presentation (click here)

3NF Decomposition: An Example

- Relation schema:

  cust_banker_branch = (<u>*customer_id, employee_id*</u>, *branch_name, type*)

- The functional dependencies for this relation schema are:

  1. *customer_id, employee_id* $\rightarrow$ *branch_name, type*
  2. *employee_id* $\rightarrow$ *branch_name*
  3. *customer_id, branch_name* $\rightarrow$ *employee_id*

- We first compute a canonical cover
  - *branch_name* is extraneous in the r.h.s. of the 1st dependency
  - No other attribute is extraneous, so we get 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 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)*