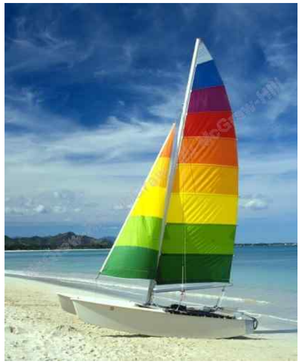


Chapter 8: Relational Database Design

Database System Concepts, 6th Ed.

©Silberschatz, Korth and Sudarshan
See www.db-book.com for conditions on re-use

Relational Database Design

- Find a "good" collection of relation schemas for our database
- Two major pitfalls to avoid in designing a database schema
  - Redundancy
    - repeating information → data inconsistency
  - Incompleteness
    - difficult or impossible to model certain aspects of the enterprise

Design Alternatives: Larger Schemas

- Suppose we combine *instructor* and *department* into *inst_dept*

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

- Redundancy
  - Wastes space
  - Complicates updating ➔ possibility of inconsistency
- Null values may be introduced ➔ difficulty in handling

Design Alternatives: Smaller Schemas

- Suppose we decompose *employee(ID, name, street, city, salary)* into *employee1* (*ID*, *name*) and *employee2* (*name*, *street, city, salary*)

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
⋮				

- Lossy decomposition
  - Introduces a loss of information

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
- Atomicity is actually a property of how the elements of the domain are used
- Non-atomic values complicate storage and encourage redundant (repeated) storage of data
- A relational schema R is in **first normal form (1NF)** if the domains of all attributes of R are atomic
- We assume all relations are in first normal form

Relational Theory

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*
- Provide the theoretical basis for good decomposition

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.

Keys and Functional Dependencies

- *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 → *building*

  and *ID* → *building*

  but would not expect the following to hold:

  dept_name → *salary*

Use of Functional Dependencies

- Test relations to see if they are legal under a given set of FDs
  - If a relation *r* is legal under a set *F* of FDs, we say that *r* **satisfies** *F.*
- Specify constraints on the set of legal relations
  - If all legal relations on *R* satisfy a set *F* of FDs, *we* say that *F* **holds on** *R.*

- Note: A specific instance of a relation schema may satisfy a FD even if the FD does not hold on all legal instances.
  - For example, a specific instance of *instructor* may, by chance, satisfy $name \rightarrow ID$.

Trivial Functional Dependency

- *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 FDs, there are certain other FDs that are logically implied by F.
  - For example: If $A \to B$ and $B \to C$, then we can infer that $A \to C$
- The set of **all** functional dependencies logically implied by F is the **closure** of F (denoted by $\mathbf{F^+}$).
- We can find F^+ by repeatedly applying **Armstrong's Axioms:**
  - if $\beta \subseteq \alpha$, then $\alpha \to \beta$ (**reflexivity**)
  - if $\alpha \to \beta$, then $\gamma\,\alpha \to \gamma\,\beta$ (**augmentation**)
  - if $\alpha \to \beta$, and $\beta \to \gamma$, then $\alpha \to \gamma$ (**transitivity**)
- These rules are
  - **sound** (generate only functional dependencies that actually hold)

    and
  - **complete** (generate all functional dependencies that hold)

Example

- $R = (A, B, C, G, H, I)$
  $F = \{ A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$
- some members of F^+
  - $A \rightarrow H$
    - 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 $CG \rightarrow CGI$,
      and augmenting of $CG \rightarrow H$ to infer $CGI \rightarrow HI$,
      and then transitivity

Additional Rules for Closure of FDs

- Additional 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 and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds (**pseudotransitivity**)

  The above rules can be inferred from Armstrong's axioms.

©Silberschatz, Korth and Sudarshan

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

©Silberschatz, Korth and Sudarshan

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. $result = ABCG$ $\quad (A \rightarrow C$ and $A \rightarrow B)$
  3. $result = ABCGH$ $\quad (CG \rightarrow H$ and $CG \subseteq AGBC)$
  4. $result = ABCGHI$ $\quad (CG \rightarrow I$ 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 $(G)^+ \supseteq R$

Uses of Attribute Closure

There are several uses of the attribute closure algorithm:

- Testing for superkey: "is α a superkey?"
  - Compute α^{+}, and check if α^{+} contains all attributes of R
- Testing functional dependencies: "does $\alpha \rightarrow \beta$ hold? (Is $\alpha \rightarrow \beta$ in F^{+}?)"
  - Just check if $\beta \subseteq \alpha^{+}$.
- Computing the closure of F
  - For each $\gamma \subseteq R$, we find the closure γ^{+}, and
    for each $S \subseteq \gamma^{+}$, we output a functional dependency $\gamma \rightarrow S$.

Canonical Cover

- Sets of functional dependencies may have redundant dependencies that can be inferred from the others
  - 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 $\{A \rightarrow B, \; B \rightarrow C, \; A \rightarrow D\}$
- A **canonical cover** for F is a set of dependencies F_c such that
  - $F^+ = F_c^+$, and
  - No functional dependency in F_c contains an extraneous attribute, and
  - Each left side of functional dependency in F_c is unique
- Intuitively, F_c is a "minimal" set of FDs equivalent to F, having no redundant dependencies or redundant parts of dependencies

Extraneous Attributes

- Let $\alpha \rightarrow \beta$ in F.
  - Attribute A $\in \alpha$ is **extraneous** if $F \Rightarrow (F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$.
  - Attribute A $\in \beta$ is **extraneous** if $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\} \Rightarrow F$.
- *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
    $\{A \rightarrow C, AB \rightarrow C\} \Rightarrow A \rightarrow C$ (i.e. the result of dropping B from $AB \rightarrow C$).
- Example: Given $F = \{A \rightarrow C, AB \rightarrow CD\}$
  - C is extraneous in $AB \rightarrow CD$ because
    $AB \rightarrow C$ can be inferred even after deleting C

Testing if an Attribute is Extraneous

Let $\alpha \rightarrow \beta$ in *F*.

- To test if attribute $A \in \alpha$ is extraneous
  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
  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

Computing a Canonical Cover

- Algorithm

$F_c = F$

repeat

  Replace any $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ in F_c with $\alpha_1 \rightarrow \beta_1 \beta_2$ (union rule)

  Find $\alpha \rightarrow \beta$ in F_c with an extraneous attribute either in α or in β

  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

- Example

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

- Combine $A \rightarrow BC$ and $A \rightarrow B$ into $A \rightarrow BC$
  - Now, $F_c = \{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$
- *A* is extraneous in $AB \rightarrow C$
  - $B \rightarrow C$ logically implies $AB \rightarrow C$
  - Now, $F_c = \{A \rightarrow BC, B \rightarrow C\}$
- *C* is extraneous in $A \rightarrow BC$
  - $A \rightarrow C$ is logically implied by $A \rightarrow B$ and $B \rightarrow C$.
- The canonical cover
  - $F_c = \{A \rightarrow B, B \rightarrow C\}$

Lossless-join Decomposition

- $\{R_1, \ldots, R_n\}$ is a **decomposition** of R if $R_1 \cup \ldots \cup R_n$
- $\{R_1, R_2\}$ is a **loseless-join decomposition** of R if

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

- Decomposition $\{R_1, R_2\}$ of R is lossless join if at least one of the following dependencies is in F^+:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$

(i.e., if one of the two sub-schemas hold the key of the other sub-schema)

Dependency Preservation

- Let F be set of FD on R, and $\{R_1, \ldots, R_n\}$ is a decomposition of R.
- The **restriction** of F to R_i, denoted by F_i, is the set of FDs in F^+ that include only attributes in R_i.
- A decomposition is **dependency preserving**, if

$$(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$$

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

Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C)$
  - Can be decomposed in two different ways
- $R_1 = (A, B), \quad R_2 = (B, C)$
  - Lossless-join decomposition:
    $$R_1 \cap R_2 = \{B\} \text{ and } B \rightarrow BC$$
  - Dependency preserving
- $R_1 = (A, B), \quad R_2 = (A, C)$
  - Lossless-join decomposition:
    $$R_1 \cap R_2 = \{A\} \text{ and } A \rightarrow AB$$
  - Not dependency preserving
    (cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

Goals of Normalization

- Let R be a relation scheme with a set F of functional dependencies
- Decide whether a relation scheme R is in "good" form
- In the case that a relation scheme R is not in "good" form, decompose it into a set of relation scheme $\{R_1, R_2, ..., R_n\}$ such that
  - Each relation scheme is in good form
  - The decomposition is a lossless-join decomposition
  - Preferably, the decomposition should be dependency preserving

Boyce-Codd Normal Form

A relation schema *R* is in **BCNF** with respect to a set *F* of FDs 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:

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

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

BCNF Example

- 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,
  - 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*)

- Of course, we are only interested in **lossless join decomposition!**

Testing for BCNF

- To check if **a non-trivial dependency** $\alpha \rightarrow \beta$ causes a violation of BCNF
  1. Compute α^+ (the attribute closure of α), and
  2. Verify that it includes all attributes of R, that is, it is a superkey of R

- To check if **a relation schema *R*** is in BCNF
  - **Simplified test**: check only the FDs in F (not in F^+) for violation of BCNF
  - If none of the dependencies in F causes a violation of BCNF, then none of the dependencies in F^+ will cause a violation of BCNF either.

- However, simplified test using only F is **incorrect** when testing **a relation *R*$_i$ in a decomposition of R**
  - Example: consider $R = (A, B, C, D, E)$, with $F = \{ A \rightarrow B, BC \rightarrow D \}$
    - Decompose R into $R_1 = (A, B)$ and $R_2 = (A, C, D, E)$
    - Neither of the dependencies in F contain only attributes from (A,C,D,E) so we might be mislead into thinking R_2 satisfies BCNF.
    - In fact, dependency $AC \rightarrow D$ in F^+ shows R_2 is not in BCNF.

BCNF Decomposition Algorithm

result := $\{R\}$;
done := false;
compute F^+;
while (not *done*) **do**
    if (there is a schema R_i in *result* that is not in BCNF)
        then begin
            let $\alpha \rightarrow \beta$ be a nontrivial functional dependency that
                holds on R_i such that $\alpha \rightarrow R_i$ is not in F^+,
                  and $\alpha \cap \beta = \emptyset$;
            result := $(result - R_i) \cup (R_i - \beta) \cup (\alpha, \beta)$;
        end
        else *done* := **true;**

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

Example of BCNF Decomposition

- *class* (*course_id*, *title*, *dept_name*, *credits*, *sec_id*, *semester*, *year*, *building*, *room_number*, *capacity*, *time_slot_id*)
- Functional dependencies:
  - *course_id* → *title*, *dept_name*, *credits*
  - *building*, *room_number* → *capacity*
  - *course_id*, *sec_id*, *semester*, *year* → *building*, *room_number*, *time_slot_id*
- 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

©Silberschatz, Korth and Sudarshan

BCNF and Dependency Preservation

- Constraints, including functional dependencies, are costly to check in practice unless they pertain to only one relation
  - It is sufficient to test only those dependencies on **each individual relation** of a decomposition
  - If so, the decomposition is *dependency preserving.*
- It is not always possible to get a BCNF decomposition that is dependency preserving
  - Example: $R = (J, K, L)$

    $F = \{JK \rightarrow L$

    $\quad L \rightarrow K\}$

    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

Third Normal Form: Motivation

- There are some situations where
  - BCNF is not dependency preserving, and
  - Efficient checking for FD violation on updates is important

- Solution: define a weaker normal form, called Third Normal Form (3NF)
  - Allows some redundancy (with resultant problems)
  - But FDs can be checked on individual relations without computing a join.
  - There is always a lossless-join, dependency-preserving decomposition into 3NF.

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).

3NF Example

- Example: $R = (J, K, L)$
  $F = \{JK \rightarrow L$
  $L \rightarrow K\}$
  Two candidate keys = JK and JL
  - R is in 3NF
    - $JK \rightarrow L$: JK is a superkey
    - $L \rightarrow K$: K is contained in a candidate key
  - But R is not in BCNF ($L \rightarrow K$: nontrivial, L is not a superkey)

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

- There is some redundancy in this schema
  - Repetition of information
    - e.g., the relationship l_1, k_1
  - Need to use null values
    - e.g., to represent the relationship l_2, k_2 where there is no corresponding value for J).

Testing for 3NF

- Need to check only FDs in F (not all FDs in F^+)
- For each dependency $\alpha \rightarrow \beta$,
  - Check if α is a superkey (using attribute closure)
- If α is not a superkey,
  - We have to verify if each attribute in β is contained in a candidate key of R
  - This test is rather more expensive, since it involves finding candidate keys
  - Testing for 3NF has been shown to be NP-hard
- Interestingly, decomposition into third normal form (described shortly) can be done in polynomial time

 ©Silberschatz, Korth and Sudarshan

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**
  $i := i + 1$;
  $R_i := \alpha\ \beta$
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 ***/**
    $R_j = R_i$;;
    $i = i - 1$;

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

©Silberschatz, Korth and Sudarshan

3NF Decomposition: An Example

- Relation schema:

  cust_banker_branch = (*customer_id, employee_id*, *branch_name, type*)

- The functional dependencies for this relation schema are:
  1. *customer_id, employee_id* → *branch_name, type*
  2. *employee_id* → *branch_name*
  3. *customer_id, branch_name* → *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 → *type*
    employee_id → *branch_name*
    customer_id, branch_name → *employee_id*

3NF Decompsition Example (Cont.)

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

  (*customer_id, employee_id, type*)

  (*employee_id*, *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 (*employee_id, branch_name*), which are subsets of other schemas

- The resultant simplified 3NF schema is:

  (*customer_id, employee_id, type*)

  (*customer_id, branch_name, employee_id)*

Comparison of BCNF and 3NF

- It is always possible to decompose a relation into a set of relations that are in 3NF such that:
  - The decomposition is lossless
  - The dependencies are preserved
- It is always possible to decompose a relation into a set of relations that are in BCNF such that:
  - The decomposition is lossless
  - It may not be possible to preserve dependencies

 ©Silberschatz, Korth and Sudarshan

Overall Database Design Process

- We have assumed schema *R* is given
  - *R* could have been generated when converting E-R diagram to a set of tables.
  - *R* could have been a single relation containing *all* attributes that are of interest (called **universal relation**).
    - Normalization breaks *R* into smaller relations
  - *R* could have been the result of some ad hoc design of relations, which we then test/convert to normal form.

Denormalization for Performance

- May want to use non-normalized schema for performance
  - For example, displaying *prereqs* along with *course_id,* and *title* requires join of *course* with *prereq*
- Alternative 1: Use denormalized relation containing attributes of *course* as well as *prereq* with all above attributes
  - faster lookup
  - extra space and extra execution time for updates
  - extra coding work for programmer and possibility of error in extra code
- Alternative 2: use a materialized view defined as

  $$course \bowtie prereq$$

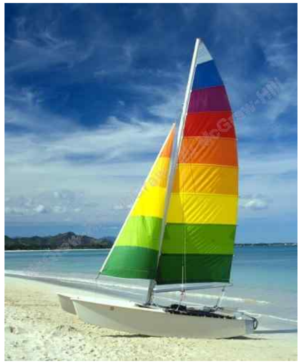
  - Benefits and drawbacks same as above, except no extra coding work for programmer and avoids possible errors

Other Design Issues

- Some aspects of database design are not caught by normalization
- Examples of bad database design, to be avoided:

  Instead of *earnings* (*company_id, year, amount*), use

  - *earnings_2004, earnings_2005, earnings_2006*, etc., all on the schema (*company_id, earnings*).
    - Above are in BCNF, but make querying across years difficult and needs new table each year
  - *company_year* (*company_id, earnings_2004, earnings_2005, earnings_2006*)
    - Also in BCNF, but also makes querying across years difficult and requires new attribute each year
    - Is an example of a **crosstab**, where values for one attribute become column names – used in spreadsheets, and in data analysis tools



End of Chapter 8

Database System Concepts, 6th Ed.

©Silberschatz, Korth and Sudarshan
See www.db-book.com for conditions on re-use