Chapter 8
Relational Database Design

Relational Database Design

- Pitfalls in Relational Database Design
- Functional Dependencies
- Normal Forms
- Decomposition
- Overall Database Design Process

Pitfalls in Design

- A bad design may lead to
  - Repetition of Information(Redundancy).
  - Inability to represent certain information (Incompleteness).
- Design Goals:
  - Avoid redundant data
  - Ensure that relationships among attributes are represented
  - Facilitate the checking of updates for violation of database integrity constraints.

What About larger Schemas?

- Suppose we combine *instructor* and *department* into *inst_dept*
- Redundancy:
  - Result is possible repetition of information
    - Wastes space
    - Complicates updating, introducing possibility of inconsistency of value
- Information Representation Problem:

  Cannot represent directly the information concerning a branch unless there exists at least one loan at the branch

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**.

Smaller Schemas?

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
⋮				

Functional Dependencies（函数依赖）

- 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,

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

A	B
1	4
1	5
3	7

$A \rightarrow B$?

$B \rightarrow A$?

key

- 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 $\alpha \subset K$, $\alpha \rightarrow R$

Example

- 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:

$$dept\_name \rightarrow salary$$

Use of Functional Dependencies

- We use functional dependencies to:
  - test relations to see if they are legal under a given set of functional dependencies.
    - If a relation *r* is legal under a set *F* of functional dependencies, we say that *r* satisfies *F*.
  - specify constraints on the set of legal relations
    - We say that *F* holds on *R* if all legal relations on *R* satisfy the set of functional dependencies *F*.

Trivial dependency（平凡依赖） *DataBase System Concepts*

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

- *A* functional dependency is transitive if:

$\alpha \to \beta$, （$\beta \nsubseteq \alpha$）, $\beta \nrightarrow \alpha$, $\beta \to \gamma$,

Then γ is transitive dependency on α .

CTO（CNO，TNO，ADDRESS，OFFICE）:

CNO→TNO，TNO→ ADDRESS

transitive dependency:

CNO→ADDRESS

Partial dependency（部分依赖）

- *A* functional dependency is partial if:

$$\alpha \to \beta, \gamma \subset \alpha, \gamma \to \beta$$

β is partially dependent on α.

CTO（CNO，TNO，ADDRESS，OFFICE）:

CNO→TNO，TNO→ ADDRESS

partial dependency:

(CNO, TNO)→ADDRESS

Logically imply（逻辑蕴含）

R, the Set of Functional dependencies F holds on R, a functional dependency *f* is logically implied by F:

If every relation instance r(R) satisfies *f*.

Closure of a Set of Functional Dependencies（函数依赖集闭包）

- For R(U,F), The set of all functional dependencies logically implied by *F* is the *closure* of *F*.
  - E.g. If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
- We denote the *closure* of *F* by **F^+**.

Armstrong's Axioms

- **reflexivity** (自反律）: if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$
- **Augmentation**（增广律）**:** if $\alpha \rightarrow \beta$, then $\gamma\,\alpha \rightarrow \gamma\,\beta$
- **transitivity** （传递律）: if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$

- These rules are
  - sound (generate only functional dependencies that actually hold)
  - complete (generate all functional dependencies that hold).

Proof

- **Reflexivity(** if $\beta \subseteq \alpha$, then $\alpha \to \beta$):

$\beta \subseteq \alpha \subseteq U$, for any two tuples t_1 and t_2 of r ,

If $t1[\alpha]=t2[\alpha]$, and $\beta \subseteq \alpha$,

Then $t1[\beta]=t2[\beta]$,$\alpha \to \beta$

- **Augmentation** (if $\alpha \to \beta$, then $\gamma\alpha \to \gamma\beta$**):**

$\alpha \to \beta$,$\gamma \subseteq U$, for any two tuples t and s of r ,

If $t[\alpha\gamma]=s[\alpha\gamma]$,

Then $t[\alpha]=s[\alpha]$, $t[\gamma]=s[\gamma]$,

and $\alpha \to \beta$, $t[\beta]=s[\beta]$,

So $t[\beta\gamma]=s[\beta\gamma]$, $\alpha\gamma \to \beta\gamma$

Procedure for Computing F^+

$F^+ = F$
repeat
for each functional dependency f in F^+
apply reflexivity and augmentation rules on f
add the resulting functional dependencies to F^+
for each pair of functional dependencies f_1 and f_2 in F^+
if f_1 and f_2 can be combined using transitivity
then add the resulting functional dependency to F^+
until F^+ does not change any further

- $R = (A, B, C, G, H, I)$

  $F = \{ A \rightarrow B$

  $A \rightarrow C$

  $CG \rightarrow H$

  $CG \rightarrow I$

  $B \rightarrow H\}$
- 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 $CG \rightarrow \mathrm{CG}I$, and augmenting of $CG \rightarrow H$ to infer $CGI \rightarrow HI$, and then transitivity

additional rules

union :

If $\alpha \rightarrow \beta$ holds *a*nd $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds

($\alpha \rightarrow \beta$, $\alpha \rightarrow \alpha\beta$; $\alpha \rightarrow \gamma$, $\alpha\beta \rightarrow \beta\gamma$; $\alpha \rightarrow \beta\gamma$)

Decomposition:

If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds

($\alpha \rightarrow \beta\gamma$, $\beta\gamma \rightarrow \beta$, $\alpha \rightarrow \beta$)

pseudotransitivity

If $\alpha \rightarrow \beta$ holds *a*nd $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds

($\alpha \rightarrow \beta$, $\alpha\gamma \rightarrow \gamma\beta$; $\gamma\beta \rightarrow \delta$, $\gamma\alpha \rightarrow \delta$)

Closure of Attribute Sets

- Given a set of attributes α, define the *closure* of α under *F* α^+（属性集闭包）：

  as the set of attributes that are functionally determined by α under *F:*

  $\alpha \rightarrow \beta$ is in F^+ $\beta \subseteq \alpha^+$

Algorithm to compute a^+

Input: α , F

Output: α^+

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

- $R = (A, B, C, G, H, I)$,

$F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$

求： $(AG)^+$

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 $CG \subseteq AGBCH)$

Uses of Attribute Closure

- *Testing for superkey*:
  - To test if α is a superkey, we compute $\alpha^{+,}$ and check if α^+ contains all attributes of *R*.
- *Testing functional dependencies*
  - To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is in F^+), just check if $\beta \subseteq \alpha^+$.
  - That is, we compute α^+ by using attribute closure, and then check if it contains β.
- *Computing closure of F*
  - For each $\gamma \subseteq R$, we find the closure γ^+, and for each $S \subseteq \gamma^+$, we output a functional dependency $\gamma \rightarrow S$.

Equivalent FD Sets

- Let F、G be two sets of FDs,

  if $F^+ = G^+$

  We see that F and G are equivalent（等价）.

$F^+ = G^+ \Leftrightarrow F \subseteq G^+, \ G \subseteq F^+$

$(F^+ \subseteq G^+, \ G^+ \subseteq F^+)$

Canonical Cover /Minimal Cover（最小覆盖、正则覆盖）

- Sets of functional dependencies may have redundant dependencies that can be inferred from the others
- Intuitively, a canonical cover of F is a “minimal” set of functional dependencies equivalent to F, with no redundant dependencies or having redundant parts of dependencies

- $A \rightarrow C$ is redundant in: $\{A \rightarrow B,\ \ B \rightarrow C,\ \ A \rightarrow C\}$
  - on RHS: $\{A \rightarrow B,\ \ B \rightarrow C,\ \ A \rightarrow CD\}$ can be simplified to
    $\{A \rightarrow B,\ \ B \rightarrow C,\ \ A \rightarrow D\}$
  - on LHS: $\{A \rightarrow B,\ \ B \rightarrow C,\ \ AC \rightarrow D\}$ can be simplified to
    $\{A \rightarrow B,\ \ B \rightarrow C,\ \ A \rightarrow D\}$

Extraneous Attributes（无关属性）

- An attribute of a functional dependency is said to be extraneous if we can remove it without changing the closure of the set of functional dependencies.
- Consider a set F of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in F.

无关(冗余)

  - 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.

- Attribute A is extraneous in α if $A \in \alpha$ and F logically implies $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$.

隐含 　　左边去掉含A的函数依赖

IF F and $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$ are equivalent : $(F = G) \Leftrightarrow F \subseteq G^+$ 且 $G \subseteq F^+$

$F \subseteq ((F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\})^+$, 与F相比，只少了 $\alpha \rightarrow \beta$

要推出有 $\alpha \rightarrow \beta$

$(\alpha - A) \rightarrow \beta,\ \alpha \rightarrow (\alpha - A),\ \alpha \rightarrow \beta$

$(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\} \subseteq F^+$ $\Rightarrow$ 与左边相比，只少了 $(\alpha - A) \rightarrow \beta$

$(\alpha - A) \rightarrow \beta \in F^+$

$(\alpha - A)^+ (F)$ contains β 　证明

属性闭包 　　即 $(\alpha - A) \rightarrow \beta$

- 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.

右边去掉A后的F

IF F and $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ are equivalent :

$(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\} \subseteq F^+$ → 少了 $\alpha \rightarrow (\beta - A)$

$\alpha \rightarrow \beta$, $\beta \rightarrow (\beta - A)$, $\alpha \rightarrow (\beta - A)$ 证

$F \subseteq ((F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\})^+$, 少了 $\alpha \rightarrow \beta$ 即 $\alpha \rightarrow A$

$\alpha \rightarrow A \in ((F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\})^+$

α^+ $((F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\})$ contains A

- Given $F = \{A \rightarrow C, AB \rightarrow C\}$

  B is extraneous in $AB \rightarrow C$ because $A \rightarrow C$ logically implies $AB \rightarrow C$.

  $A^+(F)=\{A,C\}$ contains C

- Given $F = \{A \rightarrow C, AB \rightarrow CD\}$

  C is extraneous in $AB \rightarrow CD$ since $A \rightarrow C$ can be inferred even after deleting C

  $AB^+\{A \rightarrow C, AB \rightarrow D\}=\{A,B,C,D\}$ *Contains C*

Testing Extraneous Attributes

- 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 $(\alpha - A)^+ (F)$ contains β; if it does, A is extraneous
  - 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 $\alpha^+ (F')$ contains A; if it does, A is extraneous

Canonical Cover/Minimal Cover

- A *canonical cover* for F is a set of dependencies F_c such that

  - F logically implies all dependencies in F_c
  - F_c logically implies all dependencies in F
    - No functional dependency in F_c contains an extraneous attribute
    - Each left side of functional dependency in F_c is unique.

computing a canonical cover

$F_c = F$;

repeat

左边相同的合成为一个依赖

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

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

until F does not change

Example

- $R = (A, B, C)$

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

- Combine $A \rightarrow BC$ and $A \rightarrow B$ into $A \rightarrow BC$

  Set is now $\{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$

- A is extraneous in $AB \rightarrow C$ because $B \rightarrow C$ logically implies $AB \rightarrow C$. $B^+\{F\}=\{B,C\}$ *Contains C*

  Set is now $\{A \rightarrow BC, B \rightarrow C\}$

- C is extraneous in $A \rightarrow BC$ since $A \rightarrow BC$ is logically implied by $A \rightarrow B$ and $B \rightarrow C$.

  $A^+\{A \rightarrow B, B \rightarrow C\}=\{A,B,C\}$ *contains C*

- The canonical cover is:

$$A \rightarrow B$$
$$B \rightarrow C$$

Normal Forms

- Based on functional dependencies and data dependencies:

    - First Normal Form 1NF
    - Second Normal Form 2NF
    - Third Normal Form 3NF
    - Boyce-Codd Normal Form BCNF

First Normal Form 1NF

- Domain is atomic if its elements are considered to be indivisible units.
- A relational schema R is in first normal form if the domains of all attributes of R are atomic

•Examples of non-atomic domains:

Set of names, composite attributes

Second Normal Form 2NF

- A relation schema *R* is in second normal form (2NF) if each attribute A in R meets one of the following criteria:
  - It appears in a candidate key;
  - It is not partially dependent on a candidate key.

（若R是1NF，且每个非键属性完全依赖于候选键，则称R为2NF（消除非键属性对候选键的部分依赖）。

S(<u>SNO</u> , SN , SD , DEAN , <u>CNO</u> , G) :

SNO $\rightarrow$SN, SNO $\rightarrow$SD

SC(<u>SNO , CNO</u> , G)

S_SD(<u>SNO</u> , SN , SD , DEAN)

Third Normal Form 3NF

- A relation schema R is in third normal form (3NF) if for all: $\alpha \rightarrow \beta$ 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)

（若R是2NF，且非键属性不传递依赖于R的候选键，则称R是第三范式。）

S_SD(<u>SNO</u> , SN , SD , DEAN) :

$SNO \rightarrow SD, SD \rightarrow DEAN$

STUDENT(<u>SNO</u> , SN , SD)

DEPT(<u>SD</u> , DEAN)

Boyce-Codd Normal Form BCNF

- 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

（如果关系模式R是1NF，且每个属性都不部分依赖于候选键也不传递依赖于候选键，那么称R是BC范式。）

If a relation is in BCNF it is in 3NF

SPC(SNO , PNO , CNO) :

PNO→CNO

（SNO，CNO）→PNO

SP（SNO，PNO），PC（PNO，CNO）

- 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

- Example of problems due to redundancy in 3NF

$$R = (J, K, L)$$

$$F = \{JK \rightarrow L, L \rightarrow K\}$$

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

A schema that is in 3NF but not in BCNF has the problems of

- 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).

多属性依赖集候选关键字求法：

输入：关系模式R及其函数依赖集F

输出：R的所有候选关键字。

方法：

（1）将R的所有属性分为四类：

L类：仅出现在F的函数依赖左部的属性；

R类：仅出现在F的函数依赖右部的属性；

N类：在F的函数依赖左右两边均未出现的属性；

LR类：在F的函数依赖左右两边均出现的属性；

并令X代表L、N类，Y代表LR类；

（2）求X^+，若包含了R的所有属性，则X即为R的唯一候选关键字，转（5），否则转（3）；

（3）在Y中取一属性A，求（$XA)^+$,若它包含了R的所有属性，则转（4），否则，调换一属性反复进行这一过程，直到试完所有Y中的属性；

（4）如果已找出所有的候选关键字，则转（5），否则在Y中依此取两个、三个，…，求他们的属性闭包，直到其闭包包含R的所有的属性。

（5）停止，输出结果。

（1） R（U， F）， U={A , B , C , D} , F ={B→D, AB →C};

Candidate Key {A， B}， 1NF.

（2） R（U， F）， U={A , B , C , D， E} , F ={AB→CE, E →AB， C →D};

Candidate Key {A， B} and {E}， 2NF.

（3） R（U， F）， U={A , B , C , D} , F ={B→D, D →B， AB →C};

Candidate Key {A， B} and {A， D}， 3NF.

（4）R（U，F），U={A，B , C} , F ={A→B, B →A，A →C};

Candidate Key {A}和{B}，BCNF.

（5）R（U，F），U={A，B , C} , F ={A→B, B →A，C →A};

Candidate Key {C}，2NF.

（6）R（U，F），U={A，B , C , D} , F ={A→C, D →B};

Candidate Key {A，D}，1NF.

Decomposition

- 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 $\{R_1, R_2, \ldots, R_n\}$

- All attributes of an original schema (R) must appear in the decomposition ($R_1, R_2, \ldots, R_n$):

$$R = R_1 \cup R_2 \ldots \cup R_n$$

Lossless-join decomposition

Lossless-join decomposition:

decompose R into a set of relations $\{R_1, R_2, ..., R_n\}$, For all possible relations r on schema R

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

Lossy-join decompositions result in information loss.

Example:

Decomposition of student(sno, dept, head)

- R1=(sno), R2=(dept), R3=(head)
- R1=(sno,dept), R2=(sno,head)

Lossless-join decomposition(2)

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

Eg. $R = (A, B, C)$

$F = \{A \rightarrow B, B \rightarrow C)$

$R_1 = (A, B), \quad R_2 = (B, C)$

- Lossless-join decomposition:

$$R_1 \cap R_2 = \{B\} \text{ and } B \rightarrow BC$$

Dependency preservation

- decompose a relation schema R with a set of functional dependencies F into R_1, R_2,.., R_n. Let F_i be the set of dependencies F^+ that include only attributes in R_i,

dependency preserving:

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

$F_i = \{X \rightarrow Y \mid X \rightarrow Y \in F^+ \cap XY \subseteq R_i\}$

Eg. $R = (A, B, C)$, $F = \{A \rightarrow B, B \rightarrow C)$

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

  Dependency preserving

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

  Not dependency preserving

Testing for Dependency Preservation

- check if a dependency $\alpha \rightarrow \beta$ is preserved in a decomposition of R into $R_1, R_2, \ldots, R_n$:

  result = α
  while (changes to *result*) do
      for each R_i in the decomposition
          $t = (result \cap R_i)^+(F) \cap R_i$
          $result = result \cup t$

  If *result* contains all attributes in β, then the functional dependency $\alpha \rightarrow \beta$ is preserved.

- Student(*sno*, *dept*, *head*),
  $F = \{sno \rightarrow dept, dept \rightarrow head\}$, $F^+ = F \cup \{sno \rightarrow head\} \cup \{\ldots\}$

- Decomposition 1
  $R_1(sno, dept)$, $F_1=\{sno \rightarrow dept\}$
  $R_2(sno, head)$, $F_2=\{sno \rightarrow head\}$
  - *lossless*, because
    - $R_1 \cap R_1=\{sno\}$, and is the key of R_1 and R_2
  - non-dependency preservation, because
    - $(F_1 \cup F_2)^+ \neq F^+$, $dept \rightarrow head$ is lost,

- for $dept \rightarrow head$ in **F**,

(1) with respect to R_1,

$\boldsymbol{result}=(dept \cap R_1)^+ \cap R_1 = \{dept\}^+ \cap \{sno, dept\}$

$= \{dept, head\} \cap \{sno, dept\}$

$= \{dept\}$;

(2) with respect to R_2,

$\boldsymbol{result}=(dept \cap R_2)^+ \cap R_2$

$= \Phi^+ \cap \{sno, head\} = \Phi$

$dept \rightarrow head$ is not preserved

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

return $(R_1, R_2, \ldots, R_i)$

$U=\{SNO，SD，MN，CNO，G\}$

$F=\{SNO\rightarrow SD，SNO\rightarrow MN，SD\rightarrow MN，(SNO,CNO)\rightarrow G\}$

1. $G=\{SNO\rightarrow SD，SD\rightarrow MN，(SNO,CNO)\rightarrow G\}$

2.

$U1 = \{SNO, SD\}, F1=\{SNO\rightarrow SD\}$

$U2 = \{SD, MN\}, F2=\{SD\rightarrow MN\}$

$U3 = \{SNO, CNO, G\}, F3 = \{(SNO,CNO)\rightarrow G\}$

3NF Decomposition Algorithm(2)

Above algorithm ensures:

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

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;**

BCNF Decomposition Example(1)

- U={SNO， SD， MN， CNO， G}

  F={$SNO \rightarrow SD$， $SNO \rightarrow MN$， $SD \rightarrow MN$， $(SNO,CNO) \rightarrow G$}

- （1）U1={SNO， SD} , F1={$SNO \rightarrow SD$}

  U2={SNO， MN， CNO， G}， F2={$SNO \rightarrow MN$， $(SNO,CNO) \rightarrow G$}

- （2）U1 = {SNO, SD}, F1={$SNO \rightarrow SD$}

  U2 = {SNO, MN}, F2={$SNO \rightarrow MN$}

  U3 = {SNO, CNO, G}, F3 = {$(SNO,CNO) \rightarrow G$}

BCNF Decomposition Example(2)

- R = (*branch-name, branch-city, assets, customer-name, loan-number, amount*)
  F = {*branch-name* $\rightarrow$ *assets branch-city*
  loan-number $\rightarrow$ *amount branch-name*}
  Key = {*loan-number, customer-name*}

- Decomposition
  - R_1 = (*branch-name, branch-city, assets*)
  - R_2 = (*branch-name, customer-name, loan-number, amount*)
  - R_3 = (*branch-name, loan-number, amount*)
  - R_4 = (*customer-name, loan-number*)

- Final decomposition

$$R_1, R_3, R_4$$

BCNF Decomposition Example(3)

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 - (β - α)) = (*ID, name, salary, dept_name*)

- Considering the schema ***R***(C, T, H, R, S, G),
  and ***F***={CS →G, C →T, TH →R, HR →C, HS →R}
  , give a decomposition of ***R*** into BCNF

- Step1. With respect to ***R*** and ***F***, the unique candidate key is HS, and ***R*** is not in BCNF
- Step2. Initially, ***result*** := {***R***}={*CTHRSG*}
  - ***R*** is not in BCNF, because there is CS →G in ***F***, and CS is not the candidate key of ***R***
  - ***R***(*CTHRSG*) is decomposed into $\boldsymbol{R_1}$(*CSG*) and $\boldsymbol{R_2}$(C*THRS)*
  - $\boldsymbol{R_1}$(*CSG*) is in BCNF, F_1= {CS →G}

- Step3. With respect to $\boldsymbol{R_2}(CTHRS)$
  - the restriction of $\boldsymbol{F}$ to $\boldsymbol{R_2}(CTHRS)$ is

    $F_2=\{C \rightarrow T,\ TH \rightarrow R,\ HR \rightarrow C,\ HS \rightarrow R\}$
  - the candidate key is HS
  - $\boldsymbol{R_2}(CTHRS)$ is not in BCNF, because there is $C \rightarrow T$ in $\boldsymbol{F}_2$, and C is not the candidate key of $\boldsymbol{R_2}$
  - $\boldsymbol{R_2}(CTHRS)$ is decomposed into $\boldsymbol{R_{21}}(CT)$ and $\boldsymbol{R_{22}}(CHRS)$
  - $\boldsymbol{R_{21}}(CT)$ is in BCNF , $F_{21}= \{C \rightarrow T\}$

- Step4. With respect to $\boldsymbol{R_{22}}(CHRS)$
  - the restriction of $\boldsymbol{F}$ to $\boldsymbol{R_{22}}(CHRS)$ is

    $\boldsymbol{F}_{22}$={HR →C, HS →R, CH → R} /* CH →TH, TH →R
  - the candidate key is HS
  - $\boldsymbol{R_{22}}(CHRS)$ is not in BCNF, because there is HR →C in $\boldsymbol{F}_{22}$, and HR is not candidate key of $\boldsymbol{R_{22}}$
  - $\boldsymbol{R_{22}}(CHRS)$ is decomposed into $\boldsymbol{R_{221}}(HRC)$ and $\boldsymbol{R_{222}}(HRS)$
  - $\boldsymbol{R_{221}}(HRC)$ is in BCNF
- Step5. With respect to $\boldsymbol{R_{222}}(HRS)$,
  - the restriction of $\boldsymbol{F}$ to $\boldsymbol{R_{222}}(HRS)$ is

    $\boldsymbol{F}_{222}$={HS →R}, and the candidate key is HS
  - $\boldsymbol{R_{222}}(HRS)$ is in BCNF

- Finally, the BCNF decomposition of $\boldsymbol{R}$(C, T, H, R, S, G) is
  - $\boldsymbol{R_1}(CSG)$, $\boldsymbol{R_{21}}(CT)$, $\boldsymbol{R_{221}}(HRC)$, $\boldsymbol{R_{222}}(HRS)$

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.
- Simplified test: To check if a relation schema R is in BCNF, it suffices to check only the dependencies in the given set F for violation of BCNF, rather than checking all dependencies in F^+.
  - 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, using only F is incorrect when testing a relation in a decomposition of R
  - 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.

- To check if a relation R_i in a decomposition of R is in BCNF,
  - use the original set of dependencies F that hold on R, but with the following test:
    - for every set of attributes $\alpha \subseteq R_i$, check that α^+ (the attribute closure of α) either includes no attribute of R_i - α, or includes all attributes of R_i.
  - If the condition is violated by some $\alpha \rightarrow \beta$ in F, the dependency

    $$\alpha \rightarrow (\alpha^+ - \alpha) \cap R_i$$

    can be shown to hold on R_i, and R_i violates BCNF.

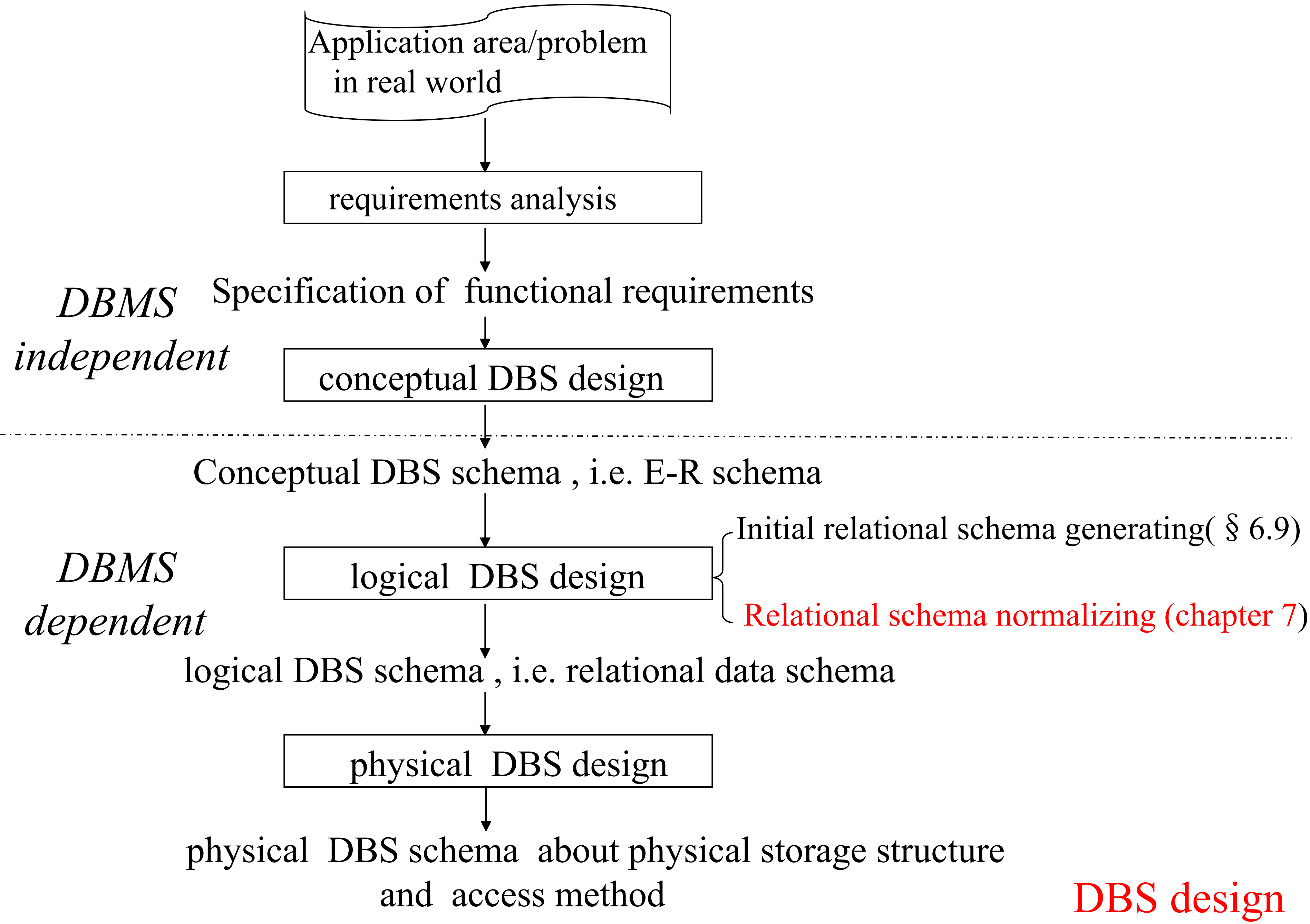
Design Goals

- Goal for a relational database design is:
  - BCNF.
  - Lossless join.
  - Dependency preservation.
- If we cannot achieve this, we accept one of
  - Lack of dependency preservation
  - Redundancy due to use of 3NF
- SQL does not provide a direct way of specifying functional dependencies other than superkeys.

Overall Database Design Process

- *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.

DBS design


Application area/problem in real world
requirements analysis
Specification of functional requirements
DBMS independent
conceptual DBS design
Conceptual DBS schema , i.e. E-R schema
Initial relational schema generating(§ 6.9)
logical DBS design
Relational schema normalizing (chapter 7)
DBMS dependent
logical DBS schema , i.e. relational data schema
physical DBS design
physical DBS schema about physical storage structure and access method

ER Model and Normalization

- When an E-R diagram is carefully designed, identifying all entities correctly, the tables generated from the E-R diagram should not need further normalization.
- However, in a real (imperfect) design there can be FDs from non-key attributes of an entity to other attributes of the entity
- E.g. *employee* entity with attributes *department-number* and *department-address*, and an FD *department-number* → *department-address*
  - Good design would have made department an entity
- FDs from non-key attributes of a relationship set possible, but rare --- most relationships are binary

Denormalization for Performance

- May want to use non-normalized schema for performance
- E.g. displaying *customer-name* along with *account-number* and *balance* requires join of *account* with *depositor*
- Alternative 1: Use denormalized relation containing attributes of *account* as well as *depositor* 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 account $\bowtie$ depositor
  - 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-2000, earnings-2001, earnings-2002*, 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-2000, earnings-2001, earnings-2002*)
    - 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

Modeling Temporal Data

- *Temporal data* have an association time interval during which the data are *valid.*
- A *snapshot* is the value of the data at a particular point in time
- Several proposals to extend ER model by adding valid time to
  - attributes, e.g. address of a customer at different points in time
  - entities, e.g. time duration when an account exists
  - relationships, e.g. time during which a customer owned an account
- But no accepted standard

- Adding a temporal component results in functional dependencies like

  customer_id → *customer_street, customer_city*

  not to hold, because the address varies over time

- A *temporal functional dependency* X → Y holds on schema *R* if the functional dependency X → Y holds on all snapshots for all legal instances r (*R*)

- In practice, database designers may add start and end time attributes to relations
  - E.g. *course*(*course_id, course_title*) → *course*(*course_id, course_title, start, end*)
    - Constraint: no two tuples can have overlapping valid times
      - Hard to enforce efficiently
- Foreign key references may be to current version of data, or to data at a point in time
  - E.g. student transcript should refer to course information at the time the course was taken

Physical Schema Design

- Indexing
- Storage
  - Files
    - Data, Log, Control
  - Table Spaces
  - Segments
    - Data,index,Temporary,Rollback
  - Extent
  - Blocks (PCTFREE,PCTUSED)

Summary

- Requirements Analysis;
  - DFD, DD ,DBIPO
- Conceptual Schema Design;
  - E_R, View Integration

- **Logical schema design;**
  - **From E_R to Tables;**
  - **Normalization**
    - **Functional Dependency, Attribute Closure, Armstrong , Minimal Closure**
    - **1NF, 2NF, 3NF, BCNF**
  - **Decomposition**
    - **Lossless Decomposition,**
    - **Dependency Preservation**