数据库系统原理

Database System Principle

邵蓥侠

Email：shaoyx@bupt.edu.cn

北京邮电大学计算机学院

计算机应用技术中心

PART 2

RELATIONAL DATABASES

Chapter 7

Relational-Database Design

schema normalization

- Features of Good Relational Design
- Atomic Domains and First Normal Form
- Decomposition Using Functional Dependencies
- Functional Dependency Theory
- Algorithms for Functional Dependencies
- (Decomposition Using Multivalued Dependencies) 略
- (More Normal Form) 略
- Database-Design Process
- (Modeling Temporal Data) 略

§ 7.1 Features of Good Relational Design

-Why Normalization Needed ?

- As shown in Fig. 8.0.1, logical DBS design consists of
  - initial relational schema generating (§ 7.6)
  - relational schema normalizing

- A bad DB design, i.e. schema not being normalized well, may result in
  - repetition of information
  - inability to represent certain information

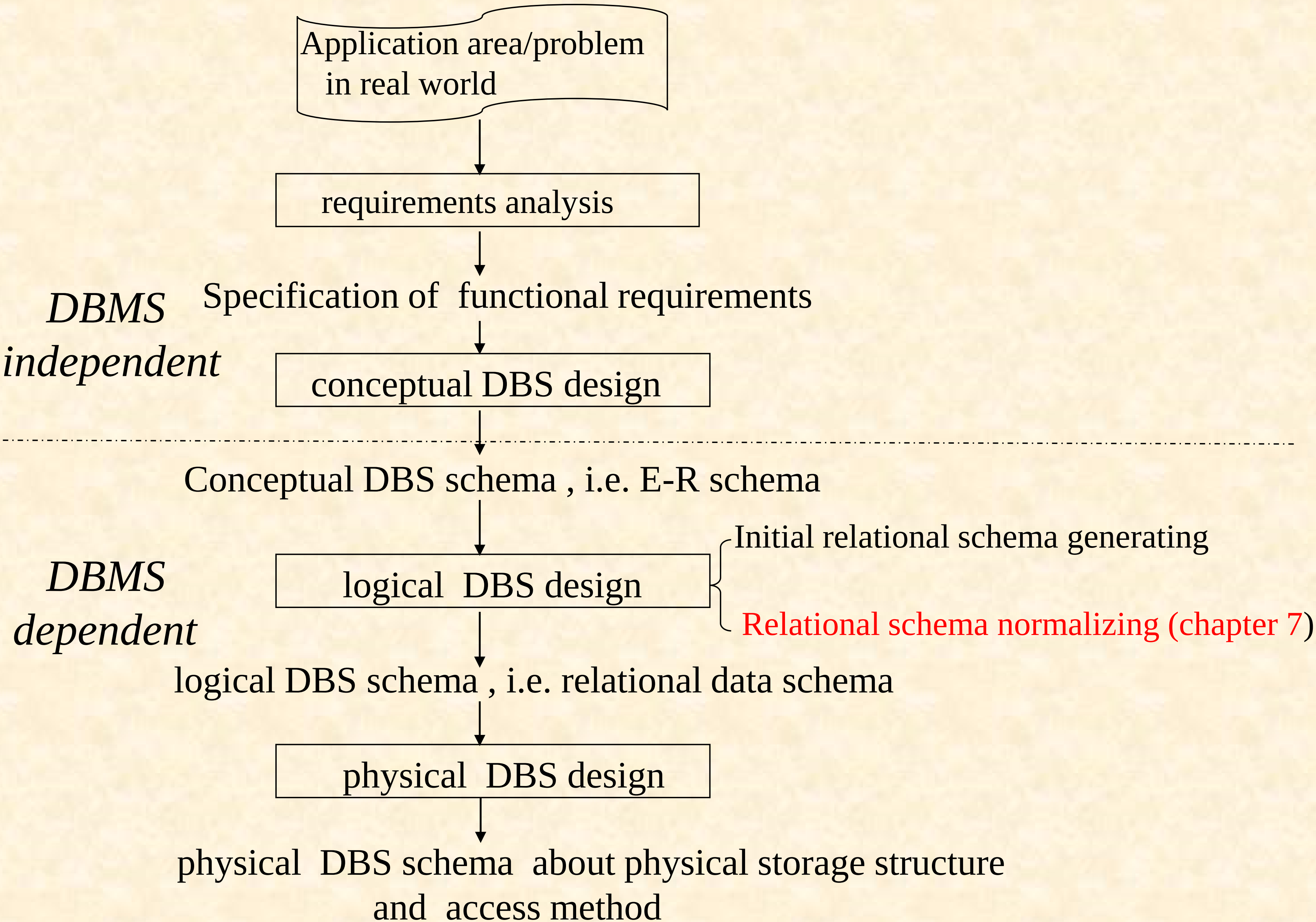


Fig.8.0.1 DBS design

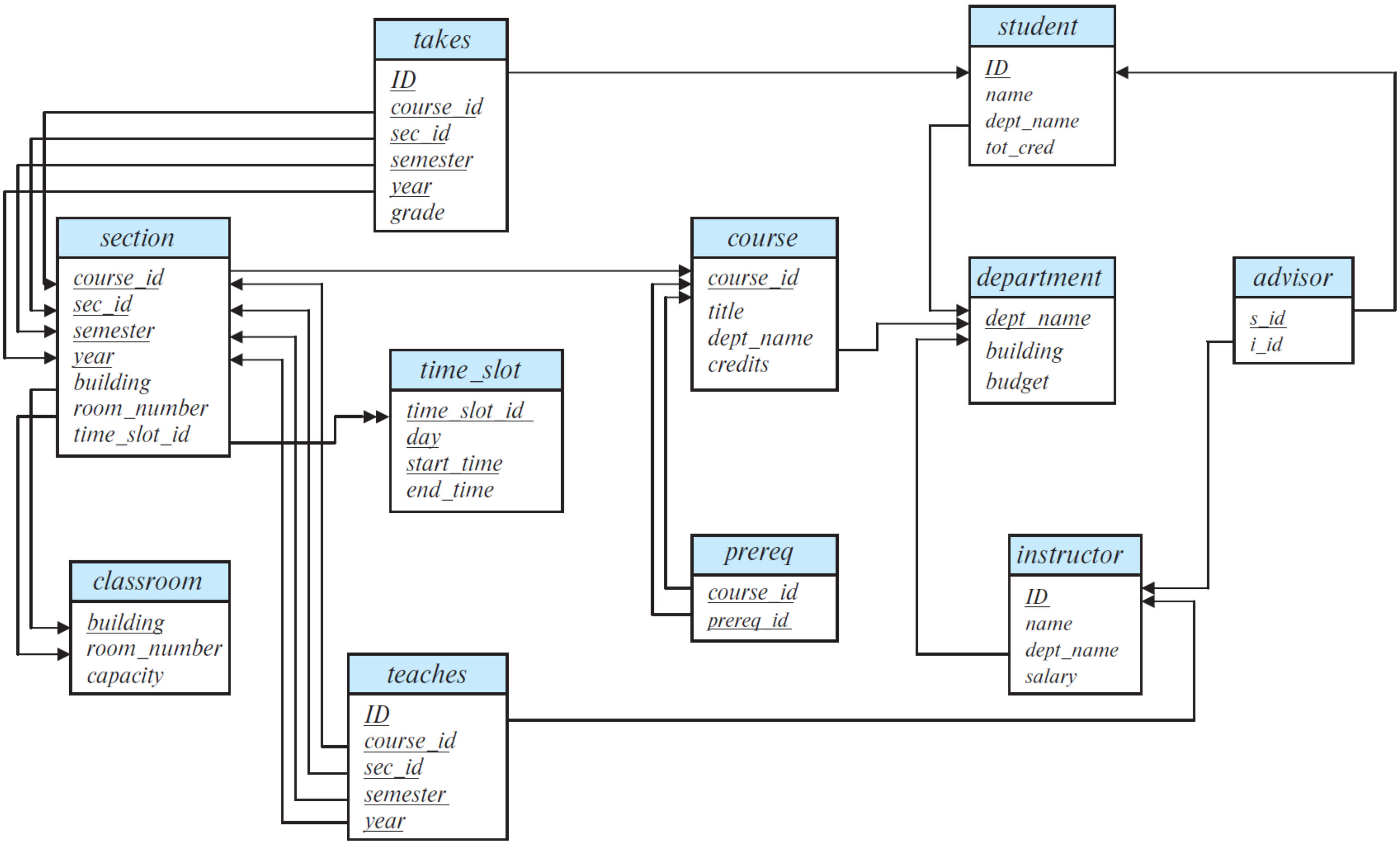


Fig. 8.0.2 Schema Diagram

7.1.1 Combine Schemas

- Suppose we combine *instructor*(*ID*, *name*, *salary*, *dept_name*) and *department*(*dept_name*, *building*, *budget*) into *inst_dept*
  - *(No connection to relationship set inst_dept)*
- Result is possible repetition of information on *department*
  - *many-to-one* mapping cardinality from *instructor* to *department*
- Pitfalls due to information repetition, when
  - *inserting*
  - *deleting*
  - *updating*

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

t_1	201081,	Zhang,	60000,	Comp.Sci,	Taylor,	100000
t_2	*null,*	*null,*	*null*,	Soft.Eng.,	*Taylor*,	100000

Fig. 7.2

Pitfalls

- ***Inserting*** problem and ***information redundancy***
  - adding a new ***instructor***
    - tuple t_1 = (201081, Zhang, 60000, Comp.Sci, Taylor, 10000) is inserted into *inst_dept*
  - data for the attributes *dept_name, building* and *budget* are repeated for the *instructors* in the 4rd, the 7th, the 9th, and the last row that the department "*Comp.Sci*" makes, space is wasted

Pitfalls

- ***Deleting*** problem
  - e.g. if the department ***Comp.Sci*** is canceled, all ***instructor*** in it should then be removed
  - in Fig 7.1, every tuples containing the ***Comp.Sci*** department should be deleted
    - the 4rd, the 7th, and the 9th row

Pitfalls

- ***Updating*** problem and ***information redundancy***
  - information redundancy complicates updating, introducing possibility of inconsistency of the ***budget*** values
  - e.g. change ***budget*** of the ***Comp.Sci*** department from 100000 to 120000,
  - in Fig7.2, every tuples belonging to ***Comp.Sci*** should be updated

Pitfalls

- Information Representation Problem
  - to represent directly information about a *new-opened* department in which there no exists ***instructor***, a tuple containing *null* values, such as

    t_2 = (null, null, null, Soft.Eng.,*Taylor*,100000)

    , is inserted
  - *null* values in DB complicate data handling in DBS

- An improved design
  - decompose ***inst_dept****-schema* into ***inst*** and ***department***

A Combined Schema Without Repetition

- Consider combining relations
  - *sec_class(sec_id, building, room_number)* and
  - *section(course_id, sec_id, semester, year)*

  into one relation
  - *section(course_id, sec_id, semester, year,*
    building, room_number)
- No repetition in this case, why?
  - ***one to one mapping*** *from section to sec_class cardinality*

7.1.2 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*

Smaller Schemas

- 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
⋮				

Fig.7.3

Example: Lossless-Join Decomposition

- **Lossless join decomposition**
- Decomposition of $R = (A, B, C)$

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

A	B	C
α	1	A
β	2	B

r

A	B
α	1
β	2

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

B	C
1	A
2	B

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

$\Pi_A (r) \bowtie \Pi_B (r)$

A	B	C
α	1	A
β	2	B

Normalization Principles

- 在逻辑DBS设计过程中，将E—R图进行转换，得到面向特定应用领域的初始关系模式集

- 这些初始关系模式集中可能存在多种（作为完整性约束的）关系模式属性间的数据依赖 (Data Dependencies) 关系
  - 函数依赖 (functional dependencies, FD, §7.4)
  - 多值依赖 (Mutivalued Dependencies, MVD, §7.6)
  - 连接依赖 (Join Dependencies, JD) 略

Principles of Relation Normalization (cont.)

- 如果直接根据初始关系模式构造DBS，由于初始关系模式中数据依赖关系的存在，可能会违反DB的完整性约束，导致DBS使用过程中出现如下问题，影响DBS的正确性、性能、效率
  - 数据冗余问题、插入问题、更新问题、删除问题 (pitfalls, § 7.1)

Principles of Relation Normalization (cont.)

- 因此，对初始关系模式集，需要根据关系规范化理论，在保证关系模式的
  - 函数无损连接性（lossless join)， **和/或**
  - 函数依赖保持性 (dependency preservation)

  约束前提下，对关系模式集进行规范化处理——**等价变换/模式分解**
- 关系模式规范化主要步骤为
  - 根据函数依赖的Armstrong’s 公理系统 (§7.4.1)和多值依赖的公理系统，从初始关系模式集中已知的函数依赖和多值依赖出发，推导出初始关系模式集中所有的函数依赖 (§8.4) 和多值依赖

Principles of Relation Normalization (cont.)

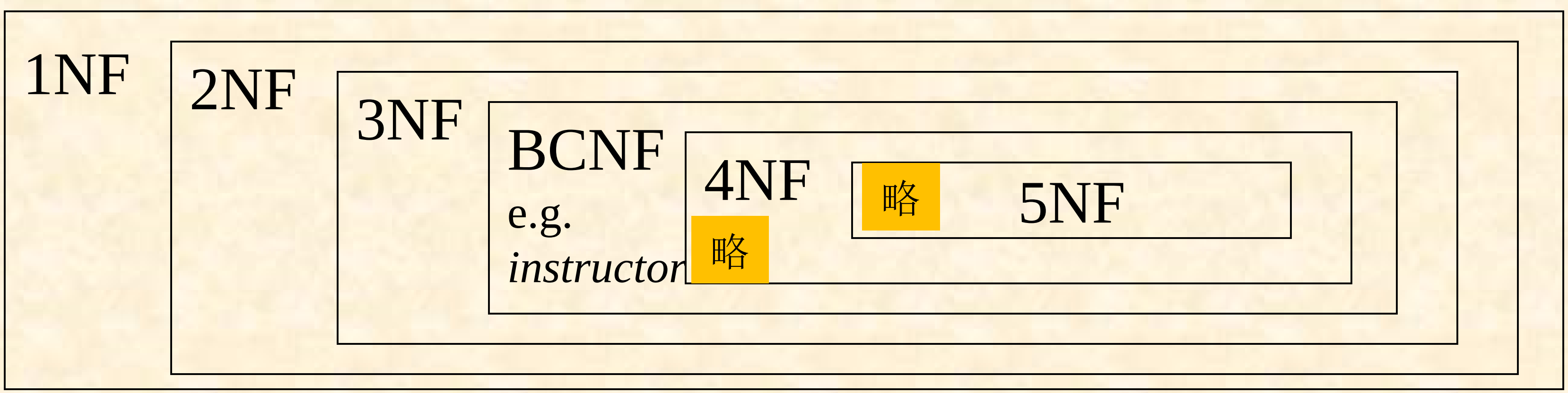
- 对具有函数依赖和多值依赖的初始关系模式集，采用
  - 模式分解算法，对其进行（等价）分解和变换，将其转换为各种范式形式，包括：
  - 1NF(§ 7.2) 、 2NF (Exercise 7.16) 、 BCNF(§ 7.3.2) 、 3NF (§ 7.3.4) 、 4NF(§ 7.6.2)

  , 以消除模式集中的函数依赖和多值依赖带来的负面影响, 保证数据库系统的完整性
- 关系模式规范化处理的基本要求为:
  - 静态关系具有第一范式形式
  - （理论上）动态关系最好具有3NF或BCNF形式

Principles of Relation Normalization (cont.)

- 3种数据依赖间的关系
  - 函数依赖是特殊的多值依赖
  - 多值依赖又是连接依赖的特例
- 范式1NF、2NF、3NF、BCNF可以看作由符合范式要求的各种关系模式组成的关系模式的集合

  e.g. 1NF = { R | R满足第一范式的定义}
- 各种范式间的关系，参见Fig.8.0.3



1NF

↓ 消除非主属性对键的部分函数依赖

2NF

↓ 消除非主属性对键的传递函数依赖

3NF

↓ 消除主属性对键的部分和传递函数依赖

BCNF

↓ 消除非平凡且非函数依赖的多值依赖

4NF

↓ 消除非平凡连接依赖

5NF

函数依赖保持性（1NF–BCNF）

无损连接性（4NF–5NF）

Fig.8.0.3 关系范式间相互关系

Principles of Relation Normalization (cont.)

- 给定一个关系模式，可以采用规范化算法将其转换为1NF、2NF、3NF、BCNF
- 对连接依赖和第五范式，无相应的模式规范化算法

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

7.2 Atomic Domains and 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

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.

7.3 Decomposition Using Functional Dependencies

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 $\{R_1, R_2, \ldots, R_n\}$ such that
  - each relation is in good form
  - the decomposition is a lossless-join decomposition
- Our theory is based on:
  - functional dependencies
  - multivalued dependencies

7.3.1 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*.

- Function: $f: X \rightarrow Y$,

  $x \in X$, $y \in Y$, $y = f(x)$

  e.g. $y = 2x$

  - **for $x_1, x_2 \in X$, if $x_1 = x_2$, then $f(x_1) = f(x_2)$**

Functional Dependencies (Cont.)

- Let R be a relation schema

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

- The **functional dependency**

$$\alpha \to \beta$$

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, $A \to B$ does **NOT** hold, but $B \to A$ does hold.

Functional Dependencies (Cont.)

- **Definition1. Functional dependency(FD)** holds on *R*

/*函数依赖**FD**在关系模式**R**上成立/保持 */

Let *R* be a relation schema, and

$$\alpha \subseteq R \text{ , } \beta \subseteq R$$

, the functional dependency $\alpha \to \beta$ holds on schema ***R***

if and only if

for ***any*** legal relation (or *relation instance*) *r*(*R*), whenever any two tuples t_i and t_j in *r* ***agree on*** the attributes α, they also *agree on* the attributes *β*, that is,

$$t_i[\alpha] = t_j[\alpha] \;\Rightarrow\; t_i[\beta] = t_j[\beta]$$

Functional Dependencies (Cont.)

- **Keys** in relational schema can be defined in terms of FD
  - 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.)

- 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:

$$\boldsymbol{dept\_name \rightarrow building}$$

and $\boldsymbol{ID \rightarrow building}$

but would not expect the following to hold:

$$\boldsymbol{dept\_name \rightarrow salary}$$

Functional Dependencies (Cont.)

- **Definition2. (***A particular relation instance***)** $r(R)$ satisfy FD, or FD is satisfied by $r(R)$

/*关系 **r(R)** 满足函数依赖集**FD, FD** 被*r*满足***/**

Given **FD** $=\{\alpha \rightarrow \beta\}$ holding on R, and a relation (instance) $\boldsymbol{r(R)}$ on R,

  - r is said to satisfy FD, if $\boldsymbol{r}$ is ***legal*** under functional dependency set FD
  - note:

    r is legal under FD$=\{\alpha \rightarrow \beta\}$ means

    - for t_i, $t_j \in r(R)$, if $t_i[\alpha]= t_j[\alpha]$, then $t_i[\beta]= t_j[\beta]$

- FD requires that the values for a certain set of attributes determines uniquely the value for another set of attributes

Example One

- Given relation r(R) shown below, which FD is satisfied by r
  - A. A →B B. **AC → B** C. BC →A D. **B → C**

	A	B	C
t1	1	*4*	2
t2	*3*	5	*6*
t3	*3*	*4*	*6*
t4	7	3	8
t5	9	1	0

Fig. 7.0.4

Example One (cont.)

- $t2[A]=t3[A]=3$, $t2[B]=5 \neq t3[B]=4$,

  $A \rightarrow B$ is not satisfied

- $t2[AC]=t3[AC]=36$, $t2[B]=5 \neq t3[B]=4$,

  $AC \rightarrow B$ is not satisfied

- $BC \rightarrow A$ is satisfied

- $t1[B]=t3[B]=4$, $t1[C]=2 \neq t3[C]=6$,

  $B \rightarrow C$ is not satisfied

Example Two

- Consider the schema *R*=(*employee_ID, date*, *turnover per-day*, *department_name*, *manager*) that describes the information about the turnover per-day (日营业额) for each employee everyday, the department that each employee works at, and the manager of the department the employee works at.

  it is assumed that

  - at every *day*, each *employee* has only one *turnover per-day*
  - each *employee* works at only one department
  - each *department* is managed by only one *manager*

- According to the descriptions mentioned above, list all the functional dependencies that hold on *R*
- *Q&A: How about the E-R diagram for this example?*

Example Two (cont.)

- Answer
  - F = { employee_ID, **date** → turnover per-day,

    employee_ID →department_name,

    department_name →manager

    }

- While E-R diagrams describe the objects in practice and the associations among them, functional dependencies (FDs) illustrate the relationships among the features/attributes of the objects (i.e. entities or relationships in E-R diagrams)

- How to guarantee the FD in DBS?

  integrity mechanisms in DBMS, i.e. keys, assertions, triggers

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*.
- Note: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances.
  - For example, a specific instance of *instructor* may, by chance, satisfy

$$name \rightarrow ID.$$

FD holds on R vs FD is satisfied by r(R)

- FD holds on *R*
  - 定义在R的属性间的语义约束，或R的属性间体现出的语义约束
  - 从设计角度，*R*应满足的约束
- FD is satisfied by r(R)
  - 根据 *R*构造的实际数据r(R)是否满足语义约束FD

FD holds on R vs FD is satisfied by r(R)

- For a schema R, there may be more than one relation instance $r(R)$, i.e. $r_1(R)$, $r_2(R)$, $r_3(R)$,…, $r_m(R)$, defined on R
  - e.g. consider $R= (A, B, C, D)$, and with respect to the instances $r_1(R)$ and $r_2(R)$ in Fig.8.0.5

- Relation $r(R)$ satisfies $\alpha \rightarrow \beta$ vs. $\alpha \rightarrow \beta$ holds on schema R
  - if $\alpha \rightarrow \beta$ holds on R, then every **legal** r(R) satisfies this R
  - but for schema R and $\alpha \rightarrow \beta$**,** if only some $r_i(R)$ satisfies R, $\alpha \rightarrow \beta$ may not hold on R.
  - e.g. in Fig.8.0.5, $A \rightarrow C$ and $AB \rightarrow D$ are satisfied by $r_1(R)$, but $A \rightarrow C$ is not satisfied by $r_2(R)$, so $A \rightarrow C$ does not holds on R

$FD_1 = \{A \rightarrow C, AB \rightarrow D\}$, satisfied by r_1

$FD_2 = \{AB \rightarrow D\}$, satisfied by r_2

r_1

	A	B	C	D
	a_1	b_1	c_1	d_1
t2	a_1	b_2	c_1	d_2
	a_2	b_2	c_2	d_2
	a_2	b_3	c_2	d_3
	a_3	b_3	c_2	d_4

r_2

	A	B	C	D
	a_1	b_1	c_1	d_1
t2	a_1	b_2	c_1	d_2
	a_2	b_2	c_2	d_2
	a_2	b_3	c_2	d_3
	a_3	b_3	c_2	d_4
t6	a_1	b_4	c_2	d_5

$A \rightarrow C$ does not hold on R, *because of* t6

Fig.8.0.5 instance r_1 and r_2 defined on schema R

- E.g. *True or false ?*
  - for a relation $r(R)$ defined on schema R, if r satisfies functional dependency FD $=\{\alpha \rightarrow \beta\}$, then FD holds on schema R
  - false

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 $\beta \subseteq \alpha$

VII-2-1 Basic Concepts (cont.)

- Def. **Transitive dependency** (传递函数依赖)
  - a functional dependency $\alpha \to \gamma$ is *transitive* if:

    $\alpha \to \beta$ ， （$\beta \not\subseteq \alpha$）， $\beta \nrightarrow \alpha$ ， $\beta \to \gamma$

    ; and γ is called transitive dependent on α

- E.g. *Student(<u>sno</u>, sname, address, depart*)
  - for *transitive* dependency

    sno→address

  , there are

    sno→sname, sname→ address

VII-2-1 Basic Concepts (cont.)

- **Def.** **Partial dependency** (部分函数依赖)
  - a functional dependency $\alpha \rightarrow \beta$ is *partial* if there is a proper subset γ of α, i.e. $\gamma \subset \alpha$, such that

    $$\gamma \rightarrow \beta$$

    ; and β is partially dependent on α
    - /*α非最小化/冗余

- E.g. Student(<u>*sno*</u>, *sname*, *address*, *depart*)
  - for *partial* dependency

    (*sno*, *sname*)→*address*

    , there are *sname*→*address*, *sno*→*address*

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, denoted as $\boldsymbol{F^+}$.

7.3.1 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*

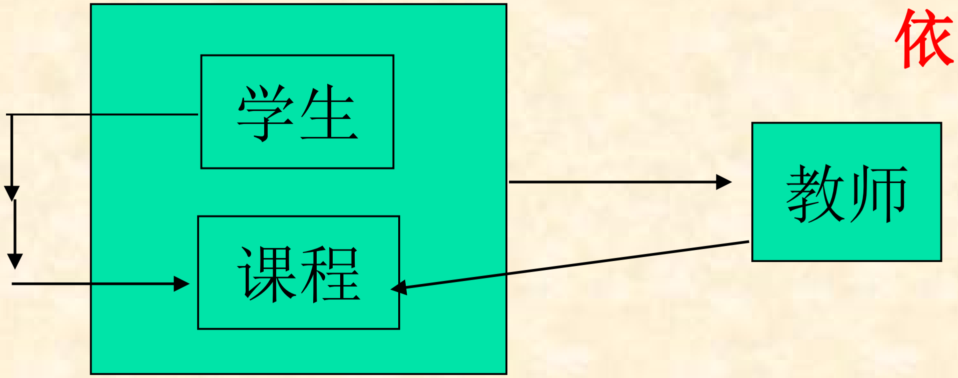
Example schema *not* in BCNF:

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

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

3NF: (学生, 教师, 课程)
教师只教一门课,每门课有若干教师,某一学生选定某门课,就对应一个固定的教师
主键(学生,课程)或(学生,教师)

主属性对键的部分和传递函数依赖



主属性对键的传递依赖: 课程依赖学生

BCNF (cont.)

$$3NF \xrightarrow{\text{消除主属性对键的部分和传递函数依赖}} BCNF$$

- BCNF性质
  - 不存在属性（主属性，非主属性）对候选键的传递和部分依赖
- If *R* is in BCNF, *R* is also in 3NF

- 以FD作为模式规范化程度度量，BCNF达到了最高规范化程度。
- 如果DBS中所有模式都属于BCNF，则消除了*插入*和*删除*异常

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,
  - 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*)

7.3.3 BCNF and Dependency Preservation

- Constraints, including functional dependencies, are costly to check in practice unless they pertain to only one relation
- If it is sufficient to test only those dependencies on each individual relation of a decomposition in order to ensure that *all* functional dependencies hold, then that decomposition is *dependency preserving*.
- Because it is not always possible to achieve both BCNF and dependency preservation, we consider a weaker normal form, known as *third normal form* (3NF).

7.3.4 Third Normal Form

- **Def 1:** A relation schema *R* is in **third normal form (3NF)** if for all:

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

at least one of the following holds:

  - $\alpha \to \beta$ 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)

E.g. B →ABC,
(ABC) – (B) =AC

- 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).

VII-3-3 Third Normal Form (cont.)

Def.2. 如果关系模式R 是2NF，而且每一个非主属性都不传递依赖于R的任何候选键，则R是3NF

消除*非主*属性对键的*传递*函数依赖

2NF $\longrightarrow$ 3NF

- If *R* is in 3NF,

- 3NF 性质
  - *R* is also in 2NF
  - 不存在非主属性对候选键的*部分*和*传递*依赖，i.e.每一个非主属性都不传递依赖于*R*的任何候选键

Example 1 about 3NF

- **Scheme** **SSS (<u>S#</u> , SD , SL)**

  $$F_{SSS}= \{S\# \rightarrow SD, SD \rightarrow SL, S\# \rightarrow SL\}$$ 闭包

- **On basis of 3NF definition1, SSS is not in 3NF, because**
  - **for *SD* → *SL*, RHS *SL* in {SL}-{SD}= {SL} *is not in* candidate key {S#}**

(/*each attribute *A* in $\beta-\alpha$ is contained in a candidate key for *R*)

- **On basis of 3NF definition2, SSS is not in 3NF, because**
  - **for *S#* → *SL*, RHS *SL* is non-primacy attribute, and is *transitive dependent on* candidate key {S#}**

  (*非主*属性对键的*传递*函数依赖)

Practice Exe. 7.19 Second Normal Form

- **Def.** A schema ***R*** is in second normal form (2NF) if
  - *R* is in 1NF, and
  - each attribute *A* in *R* meets ***one of*** the following criteria
    - it appears in a candidate key, ***that is*** *A* is a *prime attribute*
    - it is not ***partially*** dependent on a candidate key

      /* *A* is completely dependent on a candidate key */

VII-3-2 Second Normal Form (cont.)

$$1NF \xrightarrow{\text{消除非主属性对候选键的部分函数依赖}} 2NF$$

- 2NF 性质
  - 不存在非主属性对候选键的*部分*依赖
  - 每一个非主属性都完全依赖于R的**候选**键

2NF Example

- E.g. SLC(S# , SD, SL, C# , G)
  - (S#, C#) → G, *SD → SL,*

    S#→ SD, (S#, C#) → SD,

    S#→ SL, (S#, C#) → SL}

- SLC is not in 2NF, because
  - for non-primary attribute *SD*, there exits ***S#→ SD***, so *SD* is partially dependent on the key (S#, C#)
  - for non-primary attribute *SL*, there exits ***S#→ SL***, so *SL* is partially dependent on the key (S#, C#)

2NF Example (cont.)

- Decompose SLC(S# , SD , SL, C# , G) into

  SC(S# , C# , G), *SSS*(S# , SD , SL)

  - F_{SC} = { (S#, C#) → G}

    F_{SSS}= {S#→ SD, *SD → SL*, ***S#→ SL*** }
  - schema *SC* and *SSS* are in 2NF

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.

How good is BCNF?

- There are database schemas in BCNF that do not seem to be sufficiently normalized
- Consider a relation

  inst_info (ID, child_name, phone)

  - where an instructor may have more than one phone and can have multiple children

ID	*child_name*	*phone*
99999	David	512-555-1234
99999	David	512-555-4321
99999	William	512-555-1234
99999	Willian	512-555-4321

inst_info

How good is BCNF? (Cont.)

- There are no non-trivial functional dependencies and therefore the relation is in BCNF
- Insertion anomalies – i.e., if we add a phone 981-992-3443 to 99999, we need to add two tuples

(99999, David, 981-992-3443)
(99999, William, 981-992-3443)

How good is BCNF? (Cont.)

- Therefore, it is better to decompose *inst_info* into:

inst_child

ID	*child_name*
99999	David
99999	David
99999	William
99999	Willian

inst_phone

ID	*phone*
99999	512-555-1234
99999	512-555-4321
99999	512-555-1234
99999	512-555-4321

This suggests the need for higher normal forms, such as Fourth Normal Form (4NF), which we shall see later.