Introduction to Databases
《数据库引论》

Lecture 6: Relational Database Design Theory
第6讲：关系数据库设计理论

周水庚 / Shuigeng Zhou

邮件: sgzhou@fudan.edu.cn 网址：admis.fudan.edu.cn/sgzhou

复旦大学计算机科学技术学院

Outline of the Course

- **Part 0: Overview**
  - Lect. 1 (Feb. 29) - Ch1: Introduction
- **Part 1 Relational Databases**
  - Lect. 2 (Mar. 7) - Ch2: Relational model (data model, relational algebra)
  - Lect. 3 (Mar. 14) - Ch3: SQL (Introduction)
  - Lect. 4 (Mar. 21) – Ch4/5: Intermediate and Advanced SQL
- ☞ **Part 2 Database Design**
  - Lect. 5 (Mar. 28) - Ch6: Database design based on E-R model
  - **Apr. 4 (Tomb-Sweeping Day): no course**
  - Lect. 6 (Apr. 11/18) - Ch7: Relational database design
- **Midterm exam: Apr. 25**
  - **13: 00-15: 00, H3109**
- **Part 3 Data Storage & Indexing**
  - Lect. 7 (May 2 -> Apr. 28) - Ch12/13: Storage systems & structures
  - Lect. 8 (May 10) - Ch14: Indexing
- **Part 4 Query Processing & Optimization**
  - Lect. 9 (May 17) - Ch15: Query processing
  - Lect. 10 (May 24) - Ch16: Query optimization
- **Part 5 Transaction Management**
  - Lect. 11 (May 31) - Ch17: Transactions
  - Lect. 12 (Jun. 7) - Ch18: Concurrency control
  - Lect. 13 (Jun. 14) - Ch19: Recovery system

Final exam: 13:00-15:00, Jun. 26

University Database

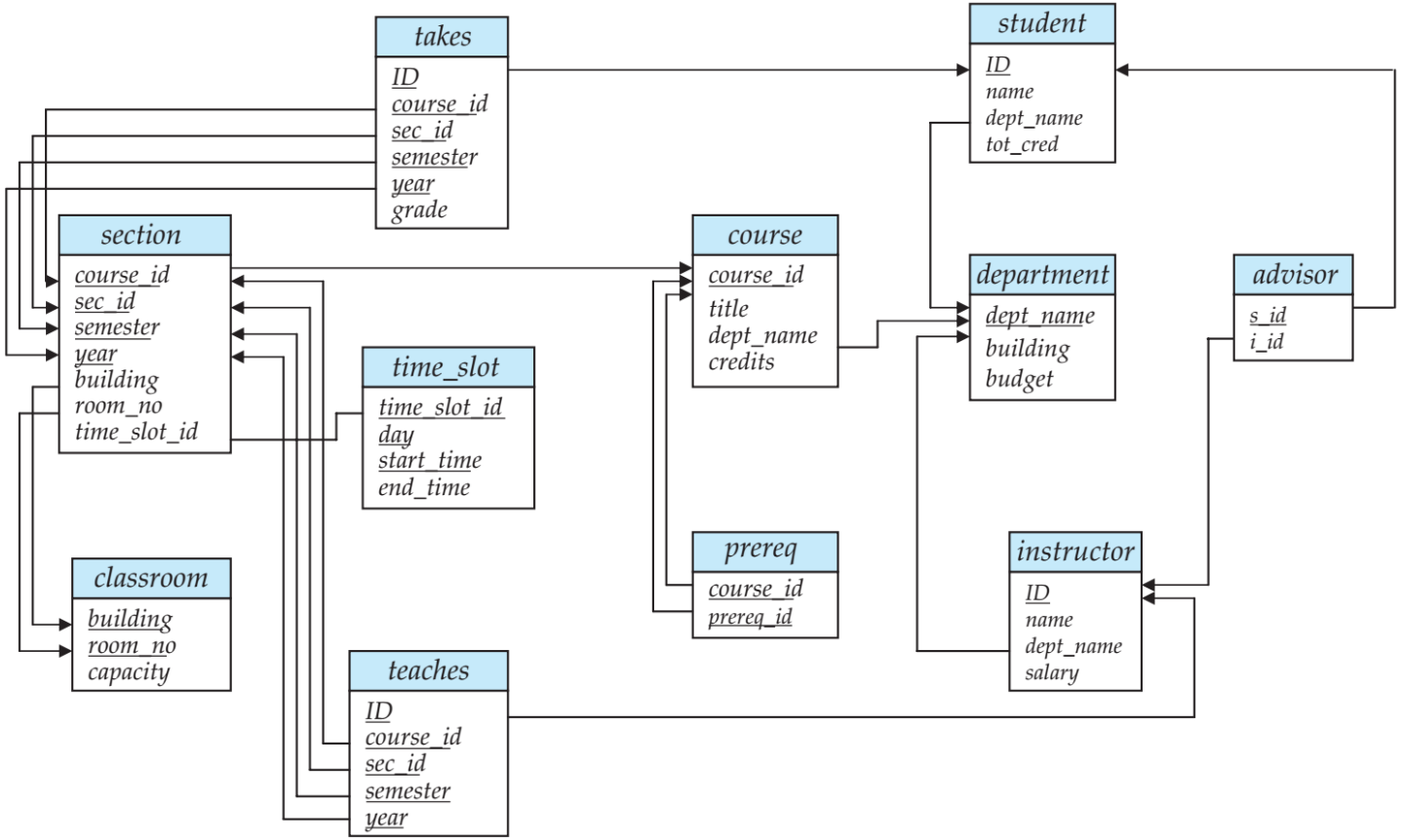
ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Instructor table

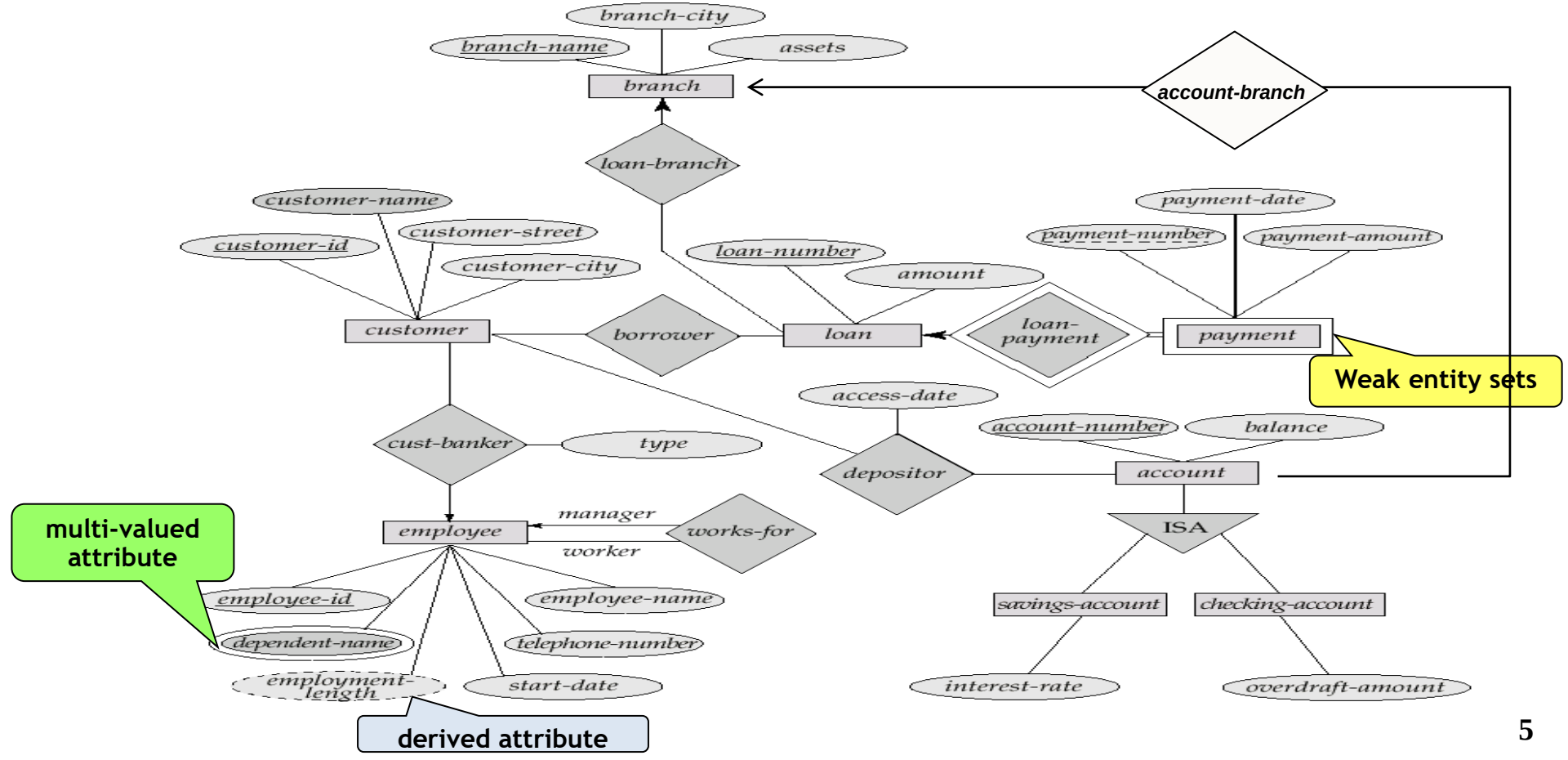
ID	*name*	*dept_name*	*tot_cred*
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
44553	Peltier	Physics	56
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98
98988	Tanaka	Biology	120

Student table

University Database



E-R Diagram for a Banking Enterprise



The Banking Schema

- branch = (branch_name, branch_city, assets)
- customer = (customer_id, customer_name, customer_street, customer_city)
- loan = (loan_number, amount)
- account = (account_number, balance)
- employee = (employee_id, employee_name, telephone_number, start_date)

- dependent_name = (employee_id, dname) (derived from a multivalued attribute)

- account_branch = (account_number, branch_name)
- loan_branch = (loan_number, branch_name)
- cust_banker = (customer_id, employee_id, type)
- borrower = (customer_id, loan_number)
- depositor = (customer_id, account_number, access_date)
- works_for = (worker_employee_id, manager_employee_id)

- payment =(loan_number,payment_number,payment_date,payment_amount)

- savings_account = (account_number, interest_rate)
- checking_account = (account_number, overdraft_amount)

Outline

- ☞ **Features of Good Relational Designs**
- **Functional Dependency（函数依赖）**
  - Functional dependency: why and what?
  - Closure of functional dependency （函数依赖闭包）
  - Closure of attribute sets （属性集闭包）
  - Canonical cover （最小覆盖）
  - Lossless-join decomposition （无损链接分解）
  - Dependency preservation （依赖保持）
- **Normalization （规范化） & Normal Forms (范式)**
- **Multivalued Dependencies* （多值依赖）**
- **Database Design Process**

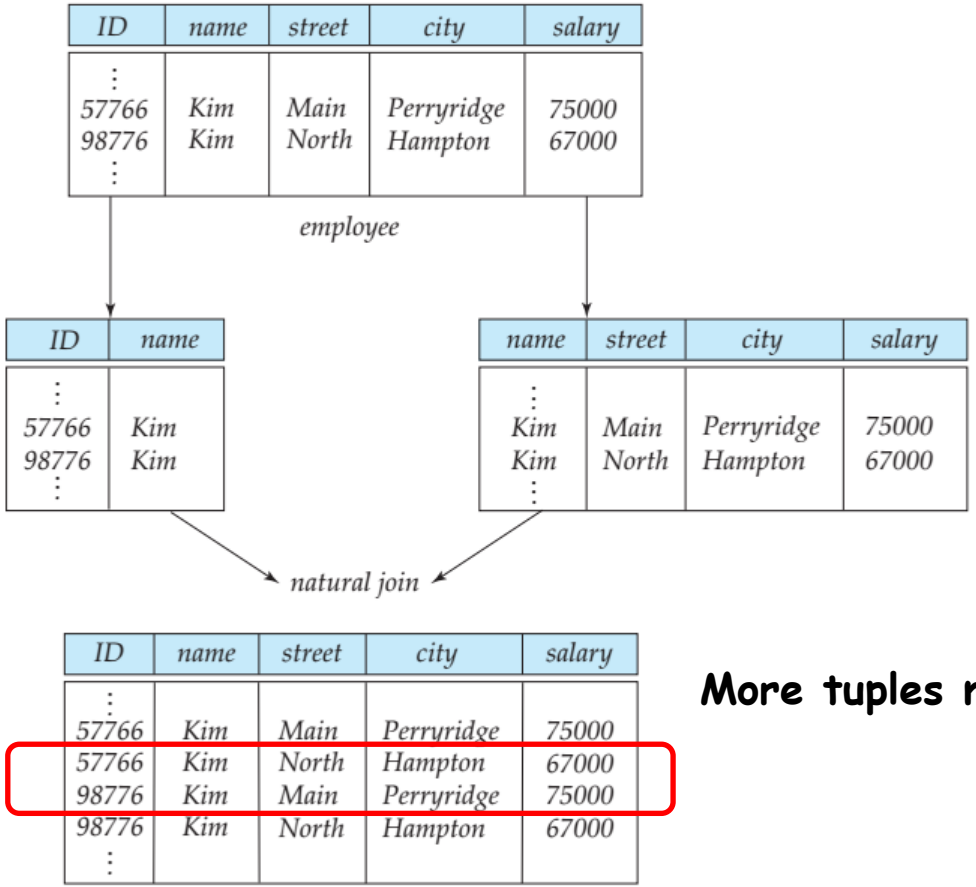
Larger Relation Schema/更大的模式

- **inst_dept (ID, name, salary, dept_name, building, budget)**
  - Redundant（冗余）: dept_name, building, budget
    - Fudan's School of CS has about 200 faculty members and staffs
  - Inconsistent（不一致）: dept_name, building, budget
  - Insert failure: cannot insert a tuple without ID, name, salary
- **Functional dependency** is needed

  dept_name → budget
- **Decomposition**

  inst_dept
  - **instructor**(ID, name, salary, dept_name)
  - **department**(dept_name, building, budget)

Smaller Relation Schema/更小的模式



More tuples mean lossy decompositions

Good Relation Schema

- **RDB design is to find a "good" collection of schemas. A bad design may lead to**
  - Repetition of information
  - Inability to represent certain information
    - e.g. representing a new department without faculty
    -
- **Design goals**
  - **Avoid redundant data**
  - **Ensure that relationships among attributes are represented**
  - **Ensuring no information loss**
  - **Facilitate the checking of updates for violation of database integrity constraints**

Outline

- **Features of Good Relational Designs**
- **Functional Dependency（函数依赖）**
  - Functional dependency: why and what?
  - Closure of functional dependency （函数依赖闭包）
  - Closure of attribute sets （属性集闭包）
  - Canonical cover （最小覆盖）
  - Lossless-join decomposition （无损链接分解）
  - Dependency preservation （依赖保持）
- **Normalization （规范化） & Normal Forms (范式)**
- **Multivalued Dependencies* （多值依赖）**
- **Database Design Process**

Example

- Consider the relation schema:
  lending_schema = (branch_name, branch_city, assets, customer_name, loan_number, amount)

branch-name	*branch-city*	*assets*	*customer-name*	*loan-number*	*amount*
Downtown	Brooklyn	9000000	Jones	L-17	1000
Redwood	Palo Alto	2100000	Smith	L-23	2000
Perryridge	Horseneck	1700000	Hayes	L-15	1500
Downtown	Brooklyn	9000000	Jackson	L-14	1500

- **Redundancy**
  - Data for branch_name, branch_city, and assets are repeated for each loan that a branch makes
  - Waste space, complicate updating, and introduce possibility of inconsistency of assets value
- **Null values**
  - Cannot store information about a branch if no loans exist
  - Can use null values, but they are difficult to handle

Decomposition

- Decompose the relation schema lending_schema into:

  branch_schema = (branch_name, branch_city, assets)

  loan_info_schema = (customer_name, loan_number, branch_name, amount)

- **All attributes** of an original schema R **must appear in** the **decomposition** (R_1, R_2):

  $$R = R_1 \cup R_2$$

- **Lossless-join decomposition (无损连接分解)**:
  - For all possible relations r on schema R: $r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r)$

Example of Non Lossless-Join Decomposition

- Decomposition of R = (A, B, C)
  - R_1 = (A,C), R_2 = (B,C)

lossy

R_1 = (A,B) R_2 = (B,C) ?

lossless

r

A	B	C
α	1	1
α	2	1
β	1	1

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

A	C
α	1
β	1

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

B	C
1	1
2	1

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

A	B
α	1
α	2
β	1

Π_{AC} (r) ⋈ Π_{BC} (r)

A	B	C
α	1	1
α	2	1
β	1	1
β	2	1

Π_{AB} (r) ⋈ Π_{BC} (r)

r

A	B	C
α	1	1
α	2	1
β	1	1

Goal - Devise a Theory for the Following

- Decide whether a particular relation R is in good form

- In the case that 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（无损连接分解）**
  - the decomposition is **dependency-preservation（保持依赖）**

- Our theory is based on:
  - **functional dependencies (函数依赖)**
  - multi-valued dependencies

Functional Dependencies (函数依赖)

- 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
  - Or a set of attributes are determined by another set of attributes

- A functional dependency is a generalization of the notion of a **key**
  - **Or key is a specific form of functional dependency**

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
  - 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
  - i.e., $t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$
- E.g.,
  - Consider $r(A, B)$ with the following instance of r

1	4
1	5
3	7

  - the $A \rightarrow B$ does **NOT hold,** but $B \rightarrow A$ **does hold**

Functional Dependencies (Cont.)

- K is a **superkey** for relation schema R **iff** $K \rightarrow R$
- K is a **candidate key** for R **iff**
  - $K \rightarrow R$, and
  - No $\alpha \subset K$, $\alpha \rightarrow R$
- FDs allow us to express constraints that cannot be expressed using superkeys. Consider the schema:

  loan_info_schema = (customer_name, loan_number, branch_name, amount)

  We expect this set of FDs to hold:

  loan_number → amount
  loan_number → branch_name

  but would not expect the following to hold:

  loan_number → customer_name

Applications of Functional Dependencies

- We use functional dependencies to:
  - **test relations** to see if they are legal under a given set of functional dependencies,
  - **specify constraints** on the set of legal relations
- **Note:** A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not holds on all legal instances.
  - For example, a specific instance of loan_schema may satisfy

    loan_number → *customer_name*

Functional Dependencies (Cont.)

- A functional dependency is **trivial(平凡的)** if it is satisfied by all instances of a relation, e.g.,

  customer_name, loan_number → customer_name

  customer_name → customer_name

  - In general, $\alpha \rightarrow \beta$ **is trivial if** $\beta \subseteq \alpha$

- **Full dependency and partially dependency**
  - β is **fully dependent** on α, if there is no proper subset α' of α such that $\alpha' \rightarrow \beta$. Otherwise, β is **partially dependent** on α

Outline

- **Features of Good Relational Designs**
- ☞ **Functional Dependency（函数依赖）**
  - Functional dependency: why and what?
  - ➢ Closure of functional dependency （函数依赖闭包）
  - Closure of attribute sets （属性集闭包）
  - Canonical cover （最小覆盖）
  - Lossless-join decomposition （无损链接分解）
  - Dependency preservation （依赖保持）
- **Normalization （规范化） & Normal Forms (范式)**
- **Multivalued Dependencies* （多值依赖）**
- **Database Design Process**

Closure of a Set of Functional Dependencies

- Given a set F of FDs, there are some other FDs that are **logically implied (逻辑蕴涵) by F**
  - E.g., if $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
  - The set of all FDs logically implied by F is the **closure (闭包) of F**
  - We denote the closure of F by F^+
- Can find all of F^+ by applying **Armstrong's Axiom (公理)**:
  - **If $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$** (**reflexivity:自反律**)
  - **If $\alpha \rightarrow \beta$, then $\gamma\alpha \rightarrow \gamma\beta$** (**augmentation:增广律**)
  - **If $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$** (**transitivity:传递律**)
- These rules are (正确且完备)
  - **sound (generate only FDs that actually hold) and**
  - **complete (generate all FDs that hold).**

Closure of Functional Dependencies (Cont.)

- We can further simplify manual computation of F^+ by using the following additional rules.
  - If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds (**union：合并规则**)
  - If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds (**decomposition：分解规则**)
  - If $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds (**pseudotransitivity：伪传递规则**)

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

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 $CG \rightarrow I$
  - $CG \rightarrow HI$
    - from $CG \rightarrow H$ and $CG \rightarrow I$: **union rule** can be inferred from
      - definition of functional dependencies, or
      - **augmentation** of $CG \rightarrow I$ to infer $CG \rightarrow CGI$, **augmentation** of $CG \rightarrow H$ to infer $CGI \rightarrow HI$, and then **transitivity**

Procedure for Computing F^+

- **To compute the closure of a set of FDs F:**

  $F^+ = F$

  apply **reflexivity (自反律)** /* Generates all trivial dependencies */

  repeat

  for each FD f in F^+

  apply **augmentation (增广律)** rules on f

  add the resulting FDs to F^+

  for each pair of FDs f_1 and f_2 in F^+

  if f_1 and f_2 can be combined using **transitivity (传递律)**

  then add the resulting FD to F^+

  until F^+ does not change any further

NOTE: We will see an alternative procedure for this task later

F^+

R(X,Y,Z), F = {X→Y, Y→Z}, F⁺ ?

F⁺ = {

X→ Φ, X→X, X→Y, X→Z, X→XY, X→XZ, X→YZ, X→XYZ,

Y→ Φ, Y→Y, Y→Z, Y→YZ,

Z→ Φ, Z→Z,

XY→ Φ, XY→X, XY→Y, XY→Z, XY→XY, XY→YZ, XY→XZ, XY→XYZ,

XZ→ Φ, XZ→X, XZ→Y, XZ→Z, XZ→XY, XZ→XZ, XZ→YZ, XZ→XYZ,

YZ→ Φ, YZ→Y, YZ→Z, YZ→YZ,

XYZ→ Φ, XYZ→X, XYZ→Y, XYZ→Z, XYZ→XY, XYZ→YZ, XYZ→XZ, XYZ→XYZ}

F={X→A1, …… , X→An}, to compute F⁺ is a NP problem

Outline

- **Features of Good Relational Designs**
- ☞ **Functional Dependency（函数依赖）**
  - Functional dependency: why and what?
  - Closure of functional dependency （函数依赖闭包）
  - ➢ Closure of attribute sets （属性集闭包）
  - Canonical cover （最小覆盖）
  - Lossless-join decomposition （无损链接分解）
  - Dependency preservation （依赖保持）
- **Normalization （规范化） & Normal Forms (范式)**
- **Multivalued Dependencies* （多值依赖）**
- **Database Design Process**

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:

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

- Algorithm to compute α^+ :

```
result:=α;
while (changes to result) do
    for each β → γ in F do
        begin
            if β ⊆result, then result:=result ∪ γ
        end
```

Example of Attribute Set Closure

Given R<U,F>, U = {A,B,C,D,E}, F={AB→C,B→D,C→E,EC→B,AC→B};

Compute: $(AB)_F^+$,$(AC)_F^+$,$(EC)_F^+$

$X^{(0)}$={A, B};

First loop:

$X^{(1)}$: for each FD in F, find FDs that the left hand side(LHS) is A,B or AB, then AB→C,B→D, and $X^{(1)}$={A,B}∪{C,D}={A,B,C,D};

Second loop:

$X^{(1)} \neq X^{(0)}$, find FDs that the left hand side is the subset of {ABCD}, then AB→C,B→D,C→E,AC→B, and $X^{(2)}$=$X^{(1)}$∪{C,D,E,B}={A,B,C,D,E};

$X^{(2)}$=U, all attributes are in $X^{(2)}$, the attribute set closure computing is end.

So $(AB)_F^+$ = {A,B,C,D,E}.

$(AC)_F^+$ = ??? $(EC)_F^+$ = ???

$(AC)_F^+$ = {A,B,C,D,E}; $(EC)_F^+$ = {B,C,D,E}

Note: 观察属性在函数依赖集中的情况，如何确定超码、候选码，有何规律?

Example of Attribute Set Closure

- $R = (A, B, C, G, H, I)$, $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$
- Calculate $(AG)^+$
  - result = AG
  - result = ABCG ($A \rightarrow C$ and $A \rightarrow B$)
  - result = ABCGH ($CG \rightarrow H$ and $CG \subseteq ABCG$)
  - result = ABCGHI = R ($CG \rightarrow I$ and $CG \subseteq ABCGH$)
- Is **AG** a **candidate key**?
  - Is **AG** a **superkey**?
    - Does $AG \rightarrow R$? == Is $(AG)^+ \supseteq R$
  - Is **any subset of AG a superkey**?
    - Does $A \rightarrow R$? == Is $(A)^+ \supseteq R$ $(A)^+$=ABCH
    - Does $G \rightarrow R$? == Is $(G)^+ \supseteq R$ $(G)^+$=G （观察属性A、G）

Applications of Attribute Closure

- **Testing for superkey**
- **Testing functional dependencies**
  - To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is in F^+), **just check if** $\beta \subseteq \alpha^+$
  - Compute α^+ by using attribute closure, then check if it contains β
  - A simple and cheap test

- **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 $\gamma \rightarrow S$

Outline

- **Features of Good Relational Designs**
- ☞ **Functional Dependency（函数依赖）**
  - Functional dependency: why and what?
  - Closure of functional dependency （函数依赖闭包）
  - Closure of attribute sets （属性集闭包）
  - ➢ Canonical cover （最小覆盖）
  - Lossless-join decomposition （无损链接分解）
  - Dependency preservation （依赖保持）
- **Normalization （规范化） & Normal Forms (范式)**
- **Multivalued Dependencies* （多值依赖）**
- **Database Design Process**

Canonical Cover (正则覆盖/最小覆盖)

- Sets of FDs may have **redundant FDs** that **can be inferred from the others**
  - E.g., $A \rightarrow C$ is redundant in: $\{A \rightarrow B,\ B \rightarrow C,\ A \rightarrow C\}$
  - Parts of a FD 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 FDs equivalent to F, having no redundant FDs or redundant parts of FDs

Extraneous Attributes (无关属性)

- Consider a set F of FDs and the FD $\alpha \rightarrow \beta$ in F
  - **Attribute A is extraneous (无关的) in α (左侧)** if $A \in \alpha$ and F logically implies $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$
  - **Attribute A is extraneous in β (右侧)** if $A \in \beta$ and the set of FDs $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha \rightarrow (\beta - A)\}$ logically implies F
- **Note:** implication in the opposite direction is trivial in each of the cases above
- 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 $AB \rightarrow CD$, it can be inferred from $= \{A \rightarrow C, AB \rightarrow D\}$

Testing if an Attribute is Extraneous

- Consider a set F of FDs and $\alpha \rightarrow \beta$ in F.
- To test if attribute $A \in \alpha$ is extraneous in α (左侧LHS)
  1. compute $(\{\alpha\} - A)^+$ using the dependencies in F
  2. check that $(\{\alpha\} - A)^+$ contains β; if it does, A is extraneous
- To test if attribute $A \in \beta$ is extraneous in β (右侧RHS)
  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

Canonical Cover

- **A canonical cover** for **F** is a set of FDs F_c such that
  - F logically implies all dependencies in F_c, and
  - F_c logically implies all dependencies in F, and
  - No FD in F_c contains an extraneous attribute, and
  - Each left side of FD in F_c is unique, i.e., there are no two FDs $\alpha_1 \rightarrow \beta_1$ and $\alpha_2 \rightarrow \beta_2$ such that $\alpha_1 = \alpha_2$
- 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 FD $\alpha \rightarrow \beta$ *with an extraneous attr. either in* α *or in* β

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

until *F does not change*

Example of Computing a Canonical Cover

- $R = (A, B, C)$ $F = \{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$, Fc=?
  - 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 $AB \rightarrow C$
    - Check if the result of deleting A from $AB \rightarrow C$ is implied by the other dependencies $B \rightarrow C$
    - Set is now $\{A \rightarrow BC, B \rightarrow C\}$
  - C is extraneous in $A \rightarrow BC$
    - Check if $A \rightarrow C$ is logically implied by $A \rightarrow B$ and the other dependencies $B \rightarrow C$
  - The canonical cover is: Fc= $\{A \rightarrow B, B \rightarrow C\}$
    - A canonical cover might not be unique. For $\{A \rightarrow C, B \rightarrow AC, C \rightarrow AB\}$, $F_c = \{A \rightarrow C, B \rightarrow C, C \rightarrow AB\}$ or $F_c = \{A \rightarrow C, B \rightarrow AC, C \rightarrow B\}$

Example of Computing a Canonical Cover

R<U,F>, U={X,Y,Z,W},

F={W→Y,Y→W,X→WY,Z→WY,XZ→W}, F_c?

(1) F={W→Y,Y→W,X→W,X→Y,Z→W,Z→Y,XZ→W}

(2) For LHS, F={W→Y,Y→W,X→W,X→Y,Z→W,Z→Y}

(3) Delete redundant FDs,F={W→Y,Y→W,X→Y,Z→Y}

Fc = {W→Y,Y→W,X→Y,Z→Y}

or Fc = {W→Y,Y→W,X→W,Z→W}

Example of Computing a Canonical Cover

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

$F_{c1} = \{A \rightarrow B,\ B \rightarrow C,\ C \rightarrow A\}$

$F_{c2} = \{A \rightarrow B,\ B \rightarrow A,\ A \rightarrow C,\ C \rightarrow A\}$

- F_{c1}、F_{c2} are all canonical covers for F
- So, a canonical cover might not be unique

More Examples

- R<U,F>, U={A,B,C,D,E,G},

F={AB→C, C→A, BC→D, ACD→B, D→EG, BE→C, CG→BD, CE→AG},

Compute $(AB)^+$, $(AC)^+$, $(CD)^+$, Fc

  - $(AB)^+$={A,B,C,D,E,G}=U, $(AC)^+$? $(CD)^+$?
  - $(AC)^+$={A,C}, $(CD)^+$={A,B,C,D,E,G}=U
  - Fc={AB→C,C→A,BC→D,CD→B,D→E,D→G,BE→C,CG→D,CE→G}
  - $(CG)^+$={A,B,C,D,E,G}=U, $(CE)^+$={A,B,C,D,E,G}=U

Find Candidate Keys

- For $R(A_1, A_2, \ldots, A_n)$ and FDs in F, all attributes can be classified into **4 types**:
  - **L**: only exists in LHS
  - **R**: only exists in RHS
  - **N**: not exists in either LHS or RHS
  - **LR**: exists in LHS and RHS both

Find Candidate Keys (Cont.)

- **Algorithm:** find candidate keys for R
- **Input:** R and its FDs set F
- **Output:** All candidate keys for R

(1) Classify all attributes into two parts: X represents for L and N types, Y for LR type

(2) Compute X^+, if X^+ contains all attributes of R, then X is the only candidate key for R, then goes to (5); otherwise goes to (3)

(3) Take attribute A from Y, compute $(XA)^+$. If $(XA)^+$ contains all attributes of R, then XA is a candidate key for R. Then take another attribute from Y, continue with the process until all attributes in Y are tested

(4) If all candidate keys are found in step (3), then goes to (5); otherwise take 2 or 3 or more attributes from Y, and compute the corresponding attribute closure (the attribute group should not contain any candidate keys already found), till the attribute closure contains all attributes of R

(5) Finished, and output the result

Find Candidate Keys (Cont.)

- Given R<U, F>, U={X, Y, Z, W}, and F={W→Y, Y→W, X→WY, Z→WY, XZ→W}, find all candidate keys of R

  a) F_c = {W→Y, Y→W, X→Y, Z→Y}

  b) $X_{LN} = X_L$ = XZ, Y_{LR} = YW

  c) X_{LN}^+ = {X,Y,Z,W} = U, so (XZ) is the only candidate key of R

Find Candidate Keys (Cont.)

- Given R<U,F>, U={A,B,C,D}, and F={AB→C, C→D, D→A}, find all candidate keys of R

  a) $F_c = \{AB \rightarrow C, C \rightarrow D, D \rightarrow A\}$

  b) $X_{LN} = X_L = B$, $Y_{LR} = ACD$

  c) $X_{LN}^+ = \{B\} \neq U$

  d) $(AB)^+ = \{ABCD\} = U$, $(BC)^+ = \{ABCD\} = U$, $(BD)^+ = \{ABCD\} = U$, then **(AB)、(BC)、(BD)** are all candidate keys of R

Find Candidate Keys (Cont.)

- Given R<U,F>, U={OBISQD}, F={S→D, D→S, I→B, B→I, B→O, O→B}, find all candidate keys of R

  (1) F_c= { ? }

  (2) X_{LN}= ? , Y_{LR}= ?

  (3) X_{LN}^+={ ? } = or ≠U?

  (4) …… , ……

  candidate keys of R ?

  (QSO)、(QDO)、(QSB)、(QDB)、(QSI)、(QDI)

Find Candidate Keys (Cont.)

- Given R<U,F>, U={OBISQD}, F={S→D, D→S, I→B, B→I, B→O, O→B}, find all candidate keys of R

(1) Fc={S→D, D→S, I→B, B→I, B→O, O→B}=F

(2) X_{LN}= Q, Y_{LR}= SDBIO

(3) X_{LN}^{+}={Q} ≠U

(4)$(QS)^{+}$={QSD},$(QD)^{+}$={QSD},$(QB)^{+}$={QBIO},$(QI)^{+}$={QBIO},$(QO)^{+}$={QBIO};

≠U

$(QSO)^{+}$、$(QSB)^{+}$、$(QSI)^{+}$、$(QSD)^{+}$、$(QDO)^{+}$、$(QDB)^{+}$、$(QDI)^{+}$、$(QDS)^{+}$、$(QBO)^{+}$、$(QBI)^{+}$、$(QBS)^{+}$、$(QBD)^{+}$、$(QIO)^{+}$、$(QIB)^{+}$、$(QSI)^{+}$、$(QID)^{+}$、$(QOB)^{+}$、$(QOI)^{+}$、$(QOS)^{+}$、$(QOD)^{+}$、

candidate keys of R:

(QSO)、(QSB)、(QSI)、(QDO)、(QDB)、(QDI)

Outline

- **Features of Good Relational Designs**
- ☞ **Functional Dependency（函数依赖）**
  - Functional dependency: why and what?
  - Closure of functional dependency （函数依赖闭包）
  - Closure of attribute sets （属性集闭包）
  - Canonical cover （最小覆盖）
  - ➢ Lossless-join decomposition （无损链接分解）
  - Dependency preservation （依赖保持）
- **Normalization （规范化） & Normal Forms (范式)**
- **Multivalued Dependencies* （多值依赖）**
- **Database Design Process**

Goals of Normalization

- Decide whether a particular relation R is in good form
- In the case that 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
  - the decomposition is dependency-preservation

- Our theory is based on:
  - functional dependencies
  - Multi-valued dependencies

Decomposition

- Decompose the relation schema *Lending_schema* into:
  Branch_schema = (branch_name, branch_city,assets)
  Loan_info_schema = (customer_name, loan_number, branch_name, amount)
- All attributes of an original schema (*R*) must appear in the decomposition (R_1, R_2):
  $$R = R_1 \cup R_2$$
- Lossless-join decomposition. For all possible relations *r* on schema *R*
  $$r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r)$$
- Theorem: A decomposition of R into R_1 and R_2 is lossless join iff at least one of the following dependencies is in F^+:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$

Example of Non Lossless-Join Decomposition

- Decomposition of R = (A, B, C), F = {A → C, B → C)

R1 = (A,C), R2 = (B,C)

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

lossy

r

A	B	C
α	1	1
α	2	1
β	1	1

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

A	C
α	1
β	1

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

B	C
1	1
2	1

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

A	B
α	1
α	2
β	1

lossless

$R_1 \cap R_2 \rightarrow R_2$

$\Pi_{AC}(r) \bowtie \Pi_{BC}(r)$

A	B	C
α	1	1
α	2	1
β	1	1
β	2	1

$R_1 \cap R_2 \rightarrow R_1$?

$R_1 \cap R_2 \rightarrow R_2$?

$\Pi_{AB}(r) \bowtie \Pi_{BC}(r)$

r

A	B	C
α	1	1
α	2	1
β	1	1

Example

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

Example

- Given R<U,F>, U={A,B,C,D,E}, F={AB→C, C→D, D→E}, and a decomposition ρ of R into:

  R1(A,B,C), R2(C,D), R3(D,E).

  ρ is a lossless-join decomposition or a lossy one?

  - (A,B, C, D, E) -> (A, B, C, D) +(D, E) (LJD)
  - (A,B,C,D) -> (A,B,C) + (C, D) (LJD)
  - ρ is LJD

Test for Lossless-join Decomposition

- **Input:** $R < U, F >$, $U = \{A_1, A_2, \ldots, A_n\}$, F, a decomposition of R: $\rho = \{R_1 < U_1, F_1 >, R_2 < U_2, F_2 >, \ldots, R_k < U_k, F_k >\}$
- **Output:** ρ is a lossless-join decomposition or a lossy one

(1) **Construct a table L** with k rows and n columns, and each column corresponds to an attribute $A_j (1 \le j \le n)$, and each row corresponds to a schema $R_i (1 \le i \le k)$. If A_j is in R_i $(A_j \in R_i)$, then fill the form with a_j at $L_{i,j}$, otherwise fill it with $b_{i,j}$.

(2) Regard table **L** as a relation on schema **R**, and **check for each FD in F** whether the FD is satisfied or not. If the FD is not satisfied, **rewrite the table** as:

- For a **FD** in **F**: **X→Y**, if **t[x1]=t[x2]**, and **t[y1]≠t[y2]**, then **rewrite y** with the same value;
  - If there is an a_j for **y**, then another **y** is set to a_j;
  - If there is not an a_j, then use one b_{ij} to replace the other **y**;
- Till no changes occur on form L

(3) If **there is a row of all a_i** (i.e. $a_1 a_2 \ldots a_n$), then **ρ is a lossless-join decomposition.** Otherwise, **ρ** is a lossy decomposition.

Example

- Given R<U,F>, U={A,B,C,D,E}, F={AB→C, C→D, D→E}, and a decomposition ρ of R into: R1(A, B, C), R2(C, D), R3(D, E). **ρ is a lossless-join decomposition or a lossy one?**

  (1) First, construct a table as:

	A	B	C	D	E
R1(A,B,C)	a_1	a_2	a_3	b_{14}	b_{15}
R2(C,D)	b_{21}	b_{22}	a_3	a_4	b_{25}
R3(D,E)	b_{31}	b_{32}	b_{33}	a_4	a_5

Example (cont.)

(2) For AB→C in F, no change occurs; for C→D, rewrite b_{14} with a_4, and for D→E, rewrite b_{15} and b_{25} as a_5. Then we have a row as: a_1, a_2, a_3, a_4, a_5. The decomposition of R into R1, R2, and R3 is a lossless-join one.

	A	B	C	D	E
R1(A,B,C)	a_1	a_2	a_3	b14→a_4	b15→a_5
R2(C,D)	b_{21}	b_{22}	a_3	a_4	b25→a_5
R3(D,E)	b_{31}	b_{32}	b_{33}	a_4	a_5

Example of Non Lossless-Join Decomposition

- Decomposition of R = (A, B, C), F = {A → C, B → C)

R1 = (A,C), R2 = (B,C) $R_1 = (A,B)$ $R_2 = (B,C)$?

lossy

r

A	B	C
α	1	1
α	2	1
β	1	1

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

A	C
α	1
β	1

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

B	C
1	1
2	1

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

A	B
α	1
α	2
β	1

lossless

$R_1 \cap R_2 \rightarrow R_2$

$\Pi_{AC}(r) \bowtie \Pi_{BC}(r)$

A	B	C
α	1	1
α	2	1
β	1	1
β	2	1

$R_1 \cap R_2 \rightarrow R_1$?
$R_1 \cap R_2 \rightarrow R_2$?

$\Pi_{AB}(r) \bowtie \Pi_{BC}(r)$

r

A	B	C
α	1	1
α	2	1
β	1	1

Example

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

Outline

- **Features of Good Relational Designs**
- ☞ **Functional Dependency（函数依赖）**
  - Functional dependency: why and what?
  - Closure of functional dependency （函数依赖闭包）
  - Closure of attribute sets （属性集闭包）
  - Canonical cover （最小覆盖）
  - Lossless-join decomposition （无损链接分解）
  - ➢ Dependency preservation （依赖保持）
- **Normalization （规范化） & Normal Forms (范式)**
- **Multivalued Dependencies* （多值依赖）**
- **Database Design Process**

Normalization using Functional Dependencies

- When we decompose a relation schema R with a set of FDs **F** into $R_1, R_2, .., R_n$ we want
  - **Lossless-join decomposition:** Otherwise decomposition would result in information loss
  - **No redundancy:** The relations R_i preferably should be in either **BCNF** or **3NF**
  - **Dependency preservation:** Let F_i be the subset of dependencies F^+ that include only attributes in R_i
    - $(F_1 \cup F_2 \cup \cdots \cup F_n)^+ = F^+$
    - Otherwise, checking updates for violation of FDs may require computing joins, which is expensive

Testing for Dependency Preservation

- To check if FD $\alpha \rightarrow \beta$ is preserved in a decomposition of R into R_1, R_2, …, R_n, we apply the following simplified test

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

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

- We apply the test on all dependencies in F to check if a decomposition is dependency preserving
- This procedure takes polynomial time, instead of the exponential time required to compute **F**$^+$ and $(F_1 \cup F_2 \cup \cdots \cup F_n)^+$

Example

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

Outline

- **Features of Good Relational Designs**
- **Functional Dependency（函数依赖）**
  - Functional dependency: why and what?
  - Closure of functional dependency （函数依赖闭包）
  - Closure of attribute sets （属性集闭包）
  - Canonical cover （最小覆盖）
  - Lossless-join decomposition （无损链接分解）
  - Dependency preservation （依赖保持）
- ☞ **Normalization （规范化） & Normal Forms (范式)**
- **Multivalued Dependencies* （多值依赖）**
- **Database Design Process**

Data Normalization (规范化)

- The process of decomposing relations with anomalies to produce smaller and well-structured relations

- To validate and improve a logical design so that it satisfies certain constraints that avoid unnecessary duplication of data

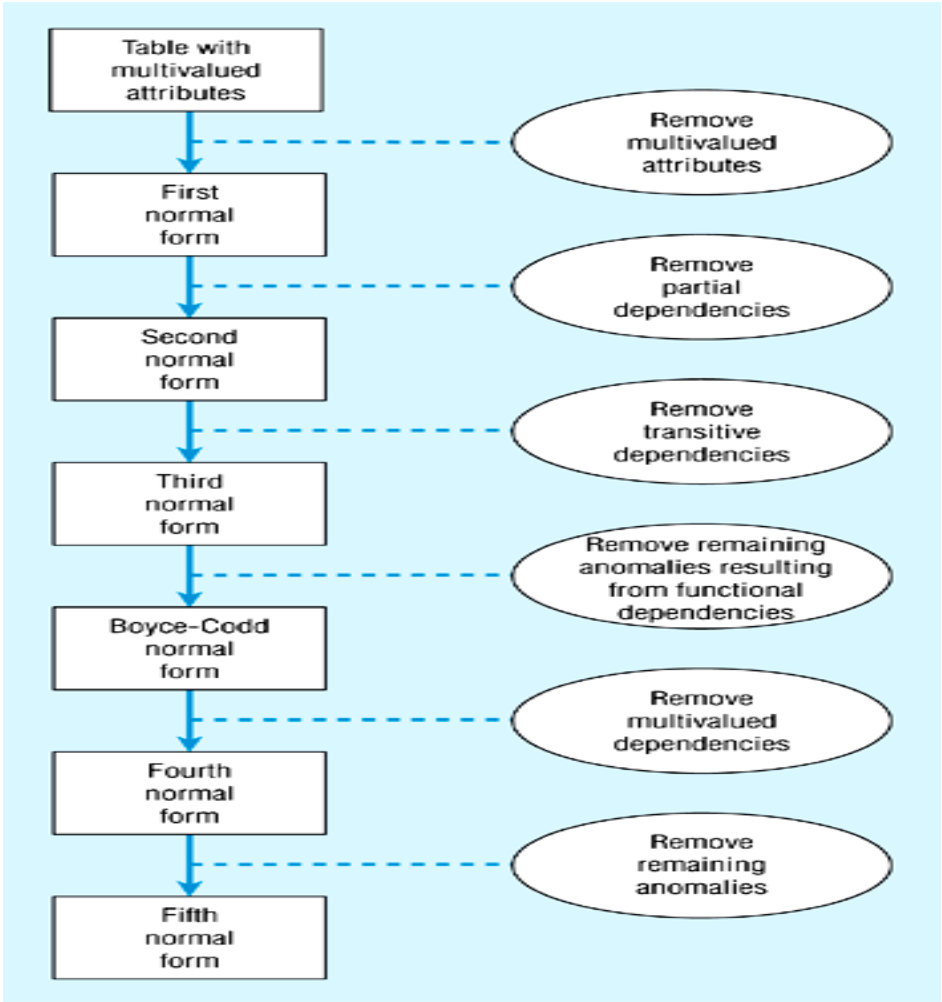
- The problems of having duplication of data
  - Waste of space
  - Difficulty in **consistency control**

Well-structured Relations

- A relation that contains minimal data redundancy and allows users to insert, delete, and update rows without causing data inconsistencies
- Goal is to avoid anomalies
  - **Insertion Anomaly** – adding new rows forces user to create duplicate data
  - **Deletion Anomaly** – deleting rows may cause a loss of data that would be needed for other future rows
  - **Modification Anomaly** – changing data in a row forces changes to other rows because of duplication

General rule of thumb: a table should not pertain to more than one entity type

Steps in Normalization



Atomic Domains and First Normal Form

- Domain is **atomic** if its elements are considered to be indivisible units
  - attributes do not have any substructure
- A relational **schema R is in 1NF** if the domains of all attributes of R are atomic
- Non-atomic values complicate storage and encourage redundant storage of data
  - E.g. composite attribute/ multivalued attributes

First Normal Form (1NF, Cont.)

- **Atomicity** is actually a property of how the elements of the domain are used
  - E.g. Strings would normally be considered indivisible
    - Suppose that students are given roll numbers which are strings of the form 0372001
  - If the first four 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

First Normal Form (1NF)

- **Requirements**
  - **No multivalued attributes**
  - **Every attribute value is atomic**

- E.g.,
  - Fig. 1 is not in 1st Normal Form (multivalued attributes)
  - Fig. 2 is in 1st Normal form

- All relations should be in 1st Normal Form

Figure 1

not in 1NF (multivalued attributes)

Emp_ID	Name	Dept_Name	Salary	Course_Title	Date_Completed
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/200X
				Surveys	10/7/200X
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/200X
110	Chris Lucero	Info Systems	43,000	Visual Basic	1/12/200X
				C++	4/22/200X
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/16/200X
				Java	8/12/200X

Figure 2

in 1NF

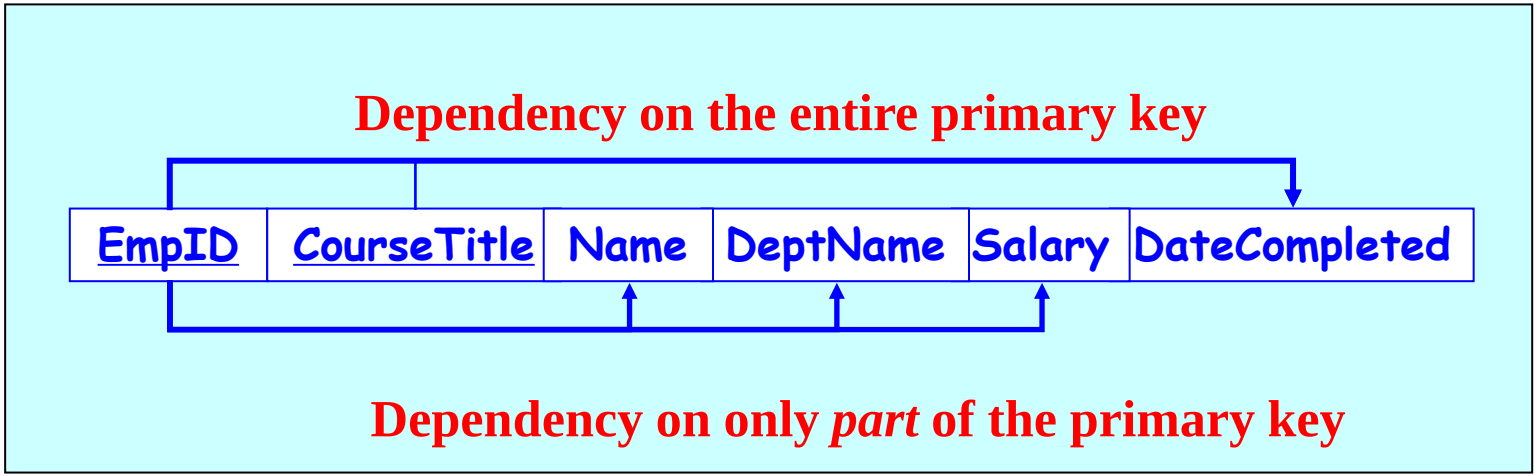
EMPLOYEE2

Emp_ID	Name	Dept_Name	Salary	Course_Title	Date_Completed
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/200X
100	Margaret Simpson	Marketing	48,000	Surveys	10/7/200X
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/200X
110	Chris Lucero	Info Systems	43,000	Visual Basic	1/12/200X
110	Chris Lucero	Info Systems	43,000	C++	4/22/200X
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/19/200X
150	Susan Martin	Marketing	42,000	Java	8/12/200X

Second Normal Form

- **2nd Normal Form**
  - 1NF
  - **Every non-key attribute is fully functionally dependent** on the **ENTIRE primary key**, i.e., **no partial functional dependencies**
- **Partial functional dependency**
  - A function dependency in which one or more non-key attributes are functionally dependent on part (but not in all) of the primary key

Functional Dependencies in Employee



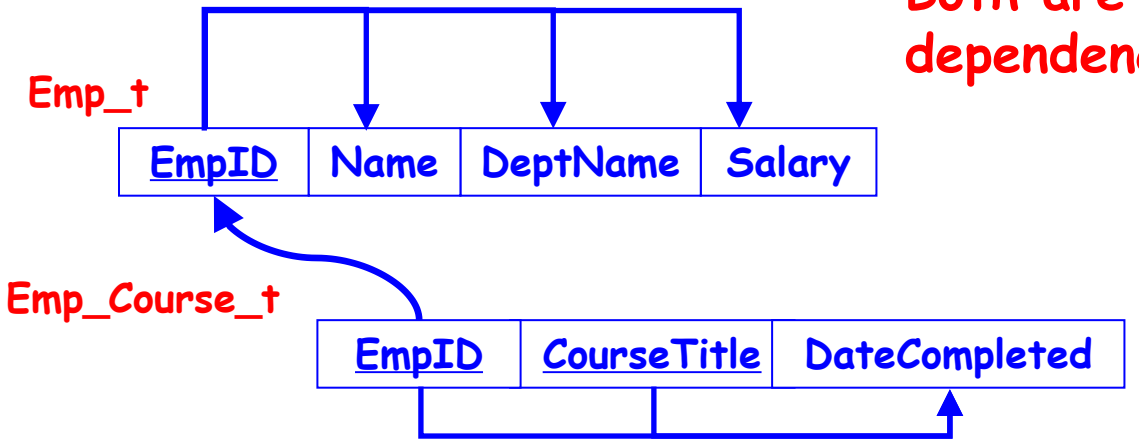
EmpID, CourseTitle ➔ DateCompleted

EmpID ➔ Name, DeptName, Salary

As such, NOT in 2nd Normal Form!

Decompose a Relation to 2nd Normal Form

- Decompose the relation into two separate relations



Both are full functional dependencies

Third Normal Form

- **Requirements**
  - **2NF**
  - **No transitive dependencies**
- A **transitive dependency** is a functional dependency between two (or more) non-key attributes.

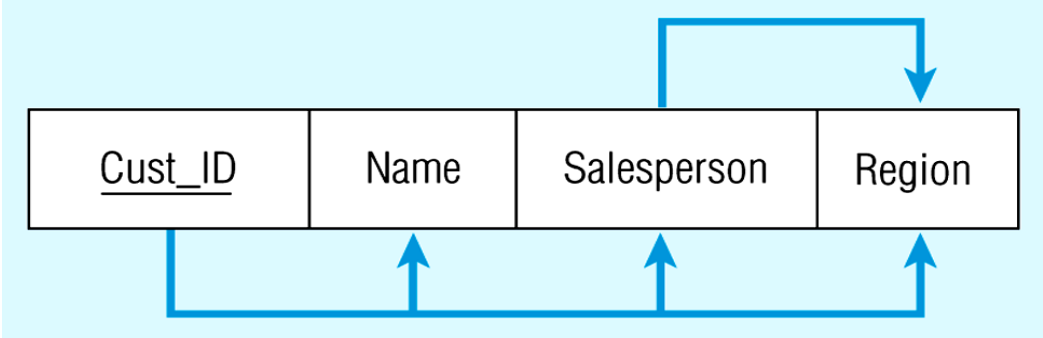
Relation with Transitive Dependency

SALES

Cust_ID	Name	Salesperson	Region
8023	Anderson	Smith	South
9167	Bancroft	Hicks	West
7924	Hobbs	Smith	South
6837	Tucker	Hernandez	East
8596	Eckersley	Hicks	West
7018	Arnold	Faulb	North

SALES relation

Relation with Transitive Dependency



Cust_ID → Name
Cust_ID → Salesperson
Cust_ID → Region

All this is OK
(2nd NF)

BUT

Cust_ID → Salesperson → Region

Transitive dependency
(not 3rd NF)

Relation with Transitive Dependency

SALES1

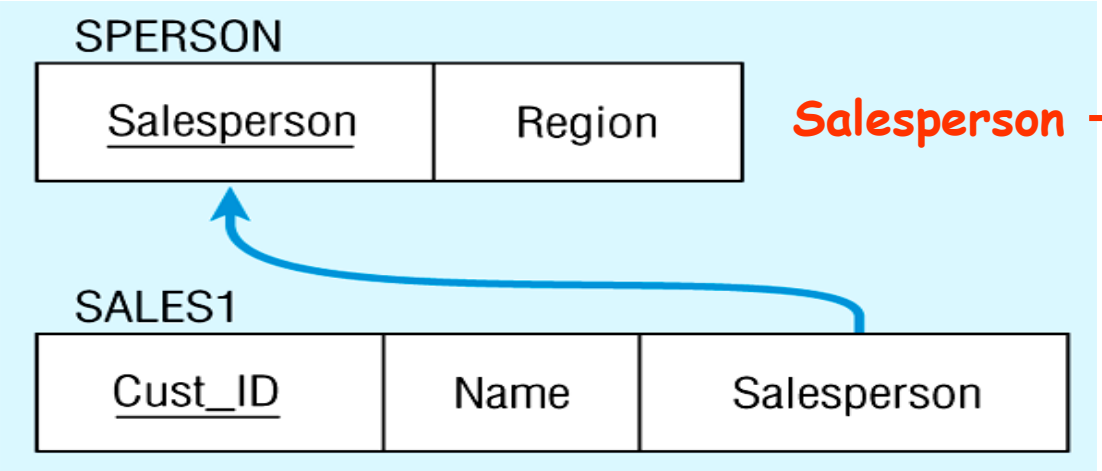
Cust_ID	Name	Salesperson
8023	Anderson	Smith
9167	Bancroft	Hicks
7924	Hobbs	Smith
6837	Tucker	Hernandez
8596	Eckersley	Hicks
7018	Arnold	Faulb

SPERSON

Salesperson	Region
Smith	South
Hicks	West
Hernandez	East
Faulb	North

Decompose the SALES relation

Relations in 3NF



Salesperson → Region

Cust_ID → Name

Cust_ID → Salesperson

Now, there are no transitive dependencies…
Both relations are in 3rd NF

Data Normalization

- 1st Normal Form
  - No multivalued attributes, and every attribute value is atomic
  - All relations are in 1st Normal Form
- 2nd Normal Form
  - 1NF + every non-key attribute is fully functionally dependent on the ENTIRE primary key
  - Decomposing the relation into two new relations
- 3rd Normal Form
  - 2NF + no transitive dependencies
  - Decomposing the relation into two new relations

Other Normal Forms

- **Boyce-Codd NF**
  - **All determinants are superkeys**
- **4th NF**
  - No multivalued dependencies
- **5th NF**
  - Join dependencies generalize MVDs
    - Lead to the project-join normal form (PJNF), or the 5th NF
- A class of even more general constraints, leads to a normal form called **domain-key normal form**
- Problem with these generalized constraints: are hard to reason with, and no set of sound and complete set of inference rules exists

Boyce-Codd Normal Form

- Given relation schema R and FDs F, **R is BCNF** if for every FD $\alpha \rightarrow \beta$ in F^+ ($\alpha \subseteq R$ **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

- **R = (A, B, C), F = {A → B, B → C }, Key = {A}**
  - **R is not in BCNF** since **B → C** but **B** is not the key

- Decomposition R_1 = (A, B), R_2 = (B, C)
  - R_1 and R_2 in BCNF
  - Lossless-join decomposition
  - Dependency preserving

Testing for BCNF

- To check if a non-trivial dependency $\alpha \rightarrow \beta$ in F^+ causes a violation of **BCNF**
  - **compute** α^+ (the attribute closure of α), and
  - verify that it includes all attributes of R, i.e., **a superkey of R**

- **Simplified test**
  - To check if a relation schema **R** is in **BCNF**, it suffices to check only the **FDs 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

Testing for BCNF (Cont.)

- Using only F is **incorrect** when testing a relation in a **decomposition** of **R**
- E.g., consider **R (A, B, C, D)** with **F = {A → B, B → C}**
  - Decompose **R** into $R_1(A,B)$ and $R_2(A,C,D)$
  - Neither of the dependencies in **F** contain only attributes from **(A,C,D)** so we might be mislead into thinking that R_2 satisfies **BCNF**
  - In fact, **dependency A → C in F^+** shows that **R_2 is not in BCNF**

Testing Decomposition for BCNF

- To check if a relation R_i in a decomposition of R **is in BCNF**
  - Either **test R_i for BCNF** w.r.t. the **restriction of F to R_i** (that is, all FDs in F^+ that contain only attributes from R_i)
  - or 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^+ either includes no attributes of $R_i - \alpha$（要么不是决定属性）, or includes all attributes of R_i （要么是R_i超键）.
    - If the condition is violated by some $\alpha \rightarrow \beta$ in **F**, the FD $\alpha \rightarrow (\alpha^+ - \alpha) \cap R_i$ can be shown to hold on R_i, and R_i violates **BCNF**
    - We use above dependency to decompose R_i

BCNF Decomposition Algorithm

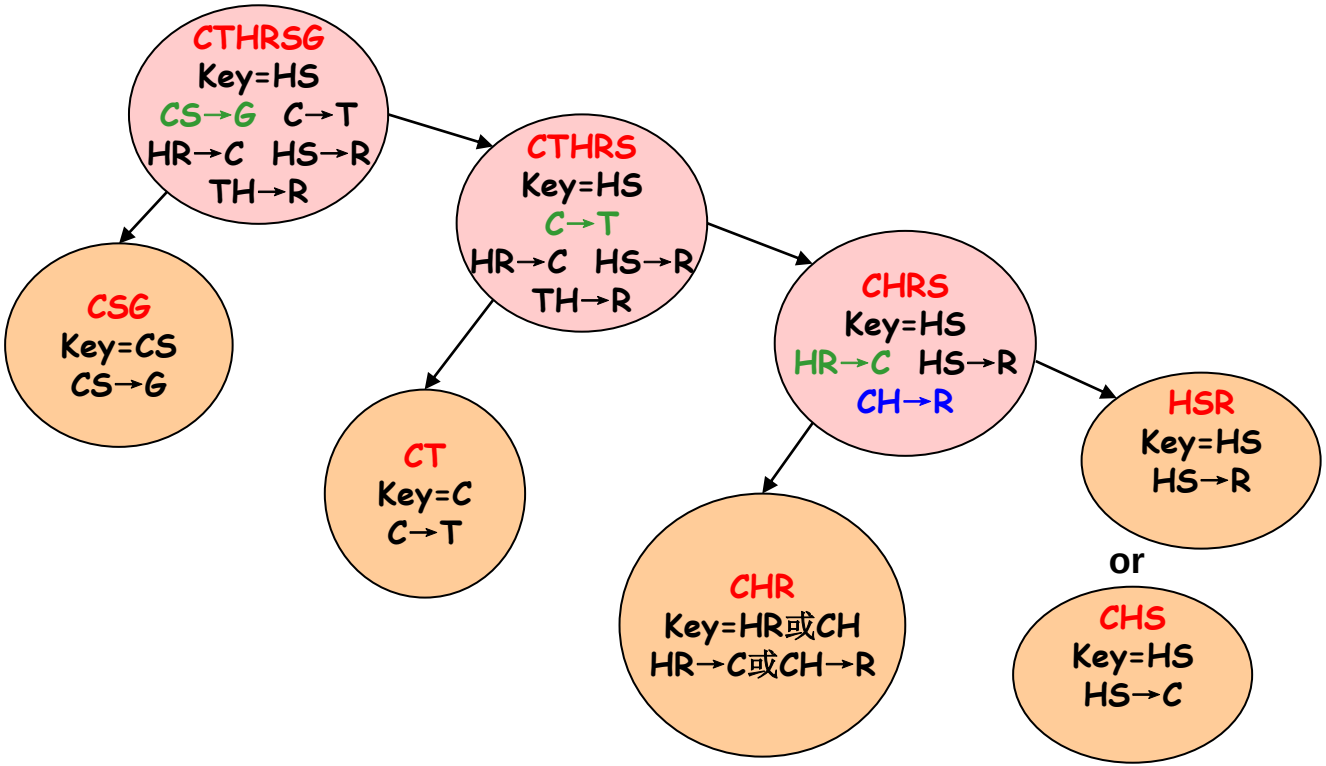
```
result := {R};
done := false;
while (not done) do
    if (there is a schema R_i in result that is not in BCNF)
      then begin
              let α → β be a nontrivial functional dependency that holds
              on R_i such that α+ does not contain R_i and α ∩ β = ∅;
              result := (result − R_i) ∪ (R_i − β) ∪ ( α, β);
           end
      else done := true;
```

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

Example

- Consider the relation scheme **CTHRSG**, where **C**=course, **T**=teacher, **H**=hour, **R**=room, **S**=student, and **G**=grade. The functional dependencies **F** we assume are:
  - **CS→G**: each student has one grade in each course
  - **C→T**: each course has one teacher
  - **HR→C**: only one course can meet in a room at one time
  - **HS→R**: a student can be in only one room at one time
  - **TH→R**: a teacher can be in only one room at one time

Decomposition Tree



BCNF and Dependency Preservation

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

- R = (J, K, L), F = {JK → L, L → K}, two candidate keys = JK and JL
  - R is not in BCNF
- Any decomposition of R will fail to preserve
  - JK → L 或者 L → K

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, i.e., **Third Normal Form**
  - Allows some redundancy
  - 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 3NF** if for **all $\alpha \rightarrow \beta$ in F^+** at least one of the following holds:
  - **$\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)**
  - **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

3NF (Cont.)

- **Example**
  - R = (J, K, L), F = {JK → L, L → K}, two candidate keys: JK and JL
  - **R is in 3NF**
    - JK → L JK is a superkey/ candidate key
    - L → K K is contained in a candidate key
  - BCNF decomposition has (JL) and (LK), and testing for JK → L requires a join
- There is some redundancy in this schema
- Equivalent to example:

  Banker-schema = (branch-name, customer-name, banker-name)

  banker-name → branch name,

  branch-name, customer-name → banker-name

Testing for 3NF

- **Optimization:** Need to check only FDs in F
- Use **attribute closure** to check for each dependency $\alpha \rightarrow \beta$, if α is a **superkey**.
- 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 involve finding candidate keys
  - testing for **3NF** has been shown to be NP-hard
  - Interestingly, **decomposition into 3NF can be done in polynomial time**

3NF Decomposition Algorithm

Let F_c be a canonical cover for F;
$i := 0$;
for each FD $\alpha \rightarrow \beta$ in F_c do
if none of the schemas R_j, $1 \le j \le i$ contains α, β
then begin
$i := i + 1$;
$R_i := \alpha \beta$
end

end
if none of the schemas R_j, $1 \le j \le i$ contains a candidate key for R
then begin
$i := i + 1$;
R_i := any candidate key for R;
end
return $(R_1, R_2, \ldots, R_i)$

The algorithm ensures that each relation schema R_i is in **3NF**, and decomposition is **dependency preserving** and **lossless-join**

3NF Decomposition Example 1

- **R<U, F>, U={A,B,C,D,E}, F={AB→CDE, AC→BDE, B→C, C→D, B→E}**
  - R is in which NF? Decompose R into 3NF, and the decomposition is dependency preserving and lossless-join

1) **Fc**={AC→B, B→CE, C→D};

2) Find **candidate keys**: **AC、AB**;
   - key-attributes are: **A、B、C;**
   - for C→D, non-key attribute D is partial dependent on key AC, so R ∉ 2NF, **R∈1NF**.

3) **Decompose R into 3NF**:
   - So decompose R into (**Same LHS attributes**):
     - **U1={A,B,C}, F1={AC → B }**
     - **U2={B,C,E}, F2={B → CE }**
     - **U3={C,D}, F3={C → D }**
   - **ρ={R1<U1,F1>, R2<U2,F2>, R3<U3,F3>},** the decomposition is **dependency preserving**. And **candidate keys AC、AB are all in U1**, so a row can be found as **a1, a2, a3, a4, a5** for testing lossless-join form, so **ρ is lossless-join**.

3NF Decomposition Example 2

- **R<U, F>, U={A,B,C,D}, F={A→C, C→A, B→AC, D→AC, BD→A}.**
  - R is in which NF? Decompose R into 3NF, and the decomposition is dependency preserving and lossless-join

1) **Fc**={A → C, C → A, B → A, D → A}

2) **Candidate keys** of R: BD; key-attributes: B、D;
   - For B→A and D→A, non-key attribute A is partial dependent on key BD, so R∉2NF, R∈ 1NF

3) **Decompose R into 3NF**:
   - All attributes exist in F, and does not exist X→A ∈ F and XA=U
   - So decompose R into (Same LHS attributes):
     - U1={A,C}, F1={ A → C, C → A }
     - U2={A,B}, F2={ B → A }
     - U3={A,D}, F3={ D → A }
   - ρ={R1<U1,F1>, R2<U2,F2>, R3<U3,F3>}, the decomposition is **dependency preserving**. But **candidate key BD is not in any Ui, so τ =ρ ∪ {R*<X,Fx>} = ρ ∪ {R4<{B,D},Φ>}, and τ is the decomposition that is dependency preserving and lossless-join**
   - **(ABCD)->(AC), (ABD) -> (AC), (AB), (AD), (BD)**

Example

- Relation schema:

  Banker-info-schema = (branch-name, customer-name, banker-name, office-number)

- The FDs for this relation schema are:

  banker-name → branch-name, office-number
  customer-name, branch-name → banker-name

- The key is:

  {customer-name, branch-name}

Applying 3NF to *Banker-info-schema*

- The for loop in the algorithm causes us to include the following schemas in our decomposition:

  Banker-office-schema = (banker-name, branch-name, office-number)
  Banker-schema = (customer-name, branch-name, banker-name)

- Since Banker-schema contains a candidate key for Banker-info-schema, we are done with the decomposition process

Comparison of BCNF and 3NF

- It is always possible to decompose a relation into relations in **3NF** and
  - **the decomposition is lossless**
  - **the dependencies are preserved**

- It is always possible to decompose a relation into relations in **BCNF** and
  - **the decomposition is lossless**
  - it **may not be possible to preserve dependencies**.

Comparison of BCNF and 3NF (Cont.)

- Example of problems due to redundancy in **3NF**
  - **R = (J, K, L)**
    F = {JK → L, L → 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*)

Design Goals

- Goal for a relational database design:
  - BCNF
  - Lossless join
  - Dependency preservation

- If we cannot achieve this, we accept one of
  - Lack of dependency preservation
  - Redundancy due to use of 3NF

Design Goals (Cont.)

- Interestingly, SQL does not provide a direct way of specifying FDs other than superkeys.
  - Can specify FDs using assertions, but they are expensive to test

- Even if we had a dependency preserving decomposition, using SQL we would **not** be able to efficiently test a FD whose left hand side is not a key.

Testing for FDs Across Relations

- If decomposition is not dependency preserving, we can have an extra materialized view for each dependency $\alpha \rightarrow \beta$ in F_c that is not preserved in the decomposition
- The materialized view is defined as a projection on $\alpha\beta$ of the join of the relations in the decomposition
- Many newer database systems support materialized views and database system maintains the view when the relations are updated.
  - No extra coding effort for programmer

Testing for FDs Across Relations (Cont.)

- The functional dependency $\alpha \rightarrow \beta$ is expressed by declaring α as a candidate key on the materialized view
- Checking for candidate key cheaper than checking $\alpha \rightarrow \beta$
- **BUT:**
  - **Space overhead**: for storing the materialized view
  - **Time overhead**: Need to keep materialized view up to date when relations are updated
  - Database system may not support key declarations on materialized views

Outline

- **Features of Good Relational Designs**
- **Functional Dependency（函数依赖）**
  - Functional dependency: why and what?
  - Closure of functional dependency （函数依赖闭包）
  - Closure of attribute sets （属性集闭包）
  - Canonical cover （最小覆盖）
  - Lossless-join decomposition （无损链接分解）
  - Dependency preservation （依赖保持）
- **Normalization （规范化） & Normal Forms (范式)**
- ☞ **Multivalued Dependencies* （多值依赖）**
- **Database Design Process**

Multivalued Dependencies

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

  classes(course, teacher, book)

  such that **(c, t, b)** $\in$ **classes** means that ***t*** is qualified to teach ***c***, and ***b*** is a required textbook for ***c***
- The database is supposed to list for each course the set of teachers any one of which can be the course's instructor, and the set of books, all of which are required for the course

course	teacher	book
database	Avi	DB Concepts
database	Avi	Ullman
database	Hank	DB Concepts
database	Hank	Ullman
database	Sudarshan	DB Concepts
database	Sudarshan	Ullman
operating systems	Avi	OS Concepts
operating systems	Avi	Shaw
operating systems	Jim	OS Concepts
operating systems	Jim	Shaw

classes

- There are no non-trivial functional dependencies and therefore the relation is in **BCNF**
- **Insertion anomalies** – i.e., if Sara is a new teacher that can teach database, two tuples need to be inserted
  - (database, Sara, DB Concepts)
  - (database, Sara, Ullman)

Multivalued Dependencies (Cont.)

- Therefore, it is better to decompose classes into:

course	*teacher*
database	Avi
database	Hank
database	Sudarshan
operating systems	Avi
operating systems	Jim

teaches

We shall see that these two relations are in 4NF

course	*book*
database	DB Concepts
database	Ullman
operating systems	OS Concepts
operating systems	Shaw

text

Multivalued Dependencies (MVDs)

- Let **R** be a relation schema and let $\alpha \subseteq R$ and $\beta \subseteq R$. The **multivalued dependency**

$$\alpha \twoheadrightarrow \beta$$

holds on **R** if in any legal relation **r(R),** for all pairs for tuples t_1 and t_2 in **r** such that $t_1[\alpha] = t_2[\alpha]$, there exist tuples t_3 and t_4 in **r** such that:

$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$
$t_3[\beta] = t_1[\beta]$
$t_3[R - \beta] = t_2[R - \beta]$
$t_4[\beta] = t_2[\beta]$
$t_4[R - \beta] = t_1[R - \beta]$

- Why called "multivalued dependency"?
  - because a value of α determine multiple values of β

Why Called Multivalued Dependencies?

- When we say $\alpha \twoheadrightarrow \beta$, it means that a value of α determine multiple values of β

course	teacher	book
database	Avi	DB Concepts
database	Avi	Ullman
database	Hank	DB Concepts
database	Hank	Ullman
database	Sudarshan	DB Concepts
database	Sudarshan	Ullman
operating systems	Avi	OS Concepts
operating systems	Avi	Shaw
operating systems	Jim	OS Concepts
operating systems	Jim	Shaw

classes

We have: course $\twoheadrightarrow$ teacher, course $\twoheadrightarrow$ book

MVD (Cont.)

Tabular representation of $\alpha \twoheadrightarrow \beta$

	α	β	$R - \alpha - \beta$
t_1	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$a_{j+1} \ldots a_n$
t_2	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$b_{j+1} \ldots b_n$
t_3	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$b_{j+1} \ldots b_n$
t_4	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$a_{j+1} \ldots a_n$

Functional dependencies: equality-generating dependencies 相等产生依赖

Multivalued dependencies: tuple-generating dependencies 元组产生依赖

MVD (Cont.)

- Properties of MVD
  - Symmetry: if $X \twoheadrightarrow Y$ then $X \twoheadrightarrow Z$, here $Z=U-X-Y$
  - Transitivity: if $X \twoheadrightarrow Y$, $Y \twoheadrightarrow Z$, then $X \twoheadrightarrow Z-Y$
  - If $X \twoheadrightarrow Y$, $X \twoheadrightarrow Z$, then $X \twoheadrightarrow YZ$
  - If $X \twoheadrightarrow Y$, $X \twoheadrightarrow Z$, then $X \twoheadrightarrow Y \cap Z$
  - If $X \twoheadrightarrow Y$, $X \twoheadrightarrow Z$, then $X \twoheadrightarrow Y-Z$, $X \twoheadrightarrow Z-Y$
  - …

Example

- Let **R** be a relation schema with a set of attributes that are partitioned into 3 nonempty subsets.

  Y, Z, W

- We say that **Y↠Z** (**Y multi-determines Z**)
  iff for all possible relations **r(R)**
  - $< y, z_1, w_1 > \in r$ and $< y, z_2, w_2 > \in r$ then
  - $< y, z_1, w_2 > \in r$ and $< y, z_2, w_1 > \in r$
- Note that since the behavior of **Z** and **W** are identical it follows that **Y↠ Z** if **Y↠W**

Example (Cont.)

- In our example:
  - course ↠ teacher
  - course ↠ book
- The above formal definition is supposed to formalize the notion that given a particular value of **Y (course)** it has associated with it a set of values of **Z (teacher)** and a set of values of **W (book)**, and these two sets are in some sense independent of each other
- Note:
  - If **Y→ Z** then **Y ↠ Z**
  - Indeed we have (in above notation) $z_1 = z_2$
    The claim follows

Use of Multivalued Dependencies

- We use **MVDs** in two ways:
  - 1. **To test relations** to determine whether they are legal under a given set of FDs and MVDs
  - 2. **To specify constraints** on the set of legal relations. We shall concern ourselves with relations that satisfy a given set of FDs and MVDs.
- If a relation r fails to satisfy a given MVD, we can construct a relations r' that does satisfy the MVD by adding tuples to r

Theory of MVDs

- From the definition of multivalued dependency, we can derive the following rule:
  - **If $\alpha \rightarrow \beta$, then $\alpha \twoheadrightarrow \beta$**; That is, every FD is also a MVD
- The closure **D^+** of D is the set of all FDs and MVDs logically implied by D.
- We can compute **D^+** from D, using the formal definitions of FDs and MVDs.
- We can manage with such reasoning for very simple MVDs, which seem to be common in practice
- For complex MVDs, it is better to reason about sets of dependencies using a system of inference rules

Fourth Normal Form

- A relation schema **R** is in **4NF** w.r.t. a set **D** of FDs and MVDs if for all MVDs in **D⁺** of the form $\alpha \twoheadrightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following hold:
  - $\alpha \twoheadrightarrow \beta$ **is trivial** (i.e., $\beta \subseteq \alpha$ or $\alpha \cup \beta = R$)
  - α is a superkey for schema **R**
- If a relation is in **4NF** it is in **BCNF**

Restriction of Multivalued Dependencies

- The restriction of **D** to R_i is the set D_i consisting of
  - All FDs in **D⁺** that include only attributes of **Ri**
  - All MVDs of the form

    $$\alpha \twoheadrightarrow \beta \cap R_i$$

    where $\alpha \subseteq R_i$ and $\alpha \twoheadrightarrow \beta$ is in **D⁺**

4NF Decomposition Algorithm

result: = {R};
done := false;
compute D+;
Let D_i denote the restriction of D+ to R_i

while (not done)
  if (there is a schema R_i in result that is not in 4NF) then
    begin
      let $\alpha \twoheadrightarrow \beta$ be a nontrivial MVD that holds on R_i such that $\alpha \rightarrow R_i$ is not in D_i, and $\alpha \cap \beta = \emptyset$;
      result := (result - R_i) $\cup$ ((R_i - β) $\cup$ (α, β));
    end
  else done:= true;

Note: each R_i is in **4NF**, and decomposition is lossless-join

Example

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

  $F = \{ A \twoheadrightarrow B$

  $B \twoheadrightarrow HI$

  $CG \twoheadrightarrow H \}$

- R is not in 4NF since $A \twoheadrightarrow B$ and A is not a superkey for R
- Decomposition

a) $R_1 = (A, B)$ (R_1 is in 4NF)

b) $R_2 = (A, C, G, H, I)$ (R_2 is not in 4NF)

c) $R_3 = (C, G, H)$ (R_3 is in 4NF)

d) $R_4 = (A, C, G, I)$ (R_4 is not in 4NF)

- Since $A \twoheadrightarrow B$ and $B \twoheadrightarrow HI$, $A \twoheadrightarrow HI$, $A \twoheadrightarrow I$

e) $R_5 = (A, I)$ (R_5 is in 4NF)

f) $R_6 = (A, C, G)$ (R_6 is in 4NF)

Further Normal Forms

- Join dependencies generalize MVDs
  - lead to **project-join normal form (PJNF)** (also called **fifth normal form**)
    投影-连接范式

- A class of even more general constraints, leads to a normal form called domain-key normal form (DKNF) 域-码范式

- Problem with these generalized constraints: are hard to reason with, and no set of sound and complete set of inference rules exists, hence rarely used

Outline

- **Features of Good Relational Designs**
- **Functional Dependency（函数依赖）**
  - Functional dependency: why and what?
  - Closure of functional dependency （函数依赖闭包）
  - Closure of attribute sets （属性集闭包）
  - Canonical cover （最小覆盖）
  - Lossless-join decomposition （无损链接分解）
  - Dependency preservation （依赖保持）
- **Normalization （规范化） & Normal Forms (范式)**
- **Multivalued Dependencies* （多值依赖）**

☞ **Database Design Process**

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, 泛关系)
  - Normalization breaks **R** into smaller relations and normal form

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 are possible, but rare

Universal Relation Approach 泛关系

- **Dangling tuples** – Tuples that "disappear" in computing a join
  - Let $r_1(R_1),\ r_2(R_2),\ \ldots,\ r_n(R_n)$ be a set of relations
  - A tuple t of the relation r_i is a dangling tuple if t is not in the relation:

$$\Pi_{R_i} \bowtie (r_1 \bowtie r_2 \bowtie \cdots \bowtie r_n)$$

- The relation $r_1 \bowtie r_2 \bowtie \cdots \bowtie r_n$ is called a universal relation since it involves all the attributes in the "universe" defined by $R_1 \cup R_2 \cup \cdots \cup R_n$
- If dangling tuples are allowed in the database, instead of decomposing a universal relation, we may prefer to synthesize a collection of normal form schemas from a given set of attributes.

Universal Relation Approach

- Dangling tuples may occur in practical database applications
- They represent incomplete information
- **E.g.,** may want to break up information about loans into:
  - (branch-name, loan-number)
  - (loan-number, amount)
  - (loan-number, customer-name)
- Universal relation would require null values, and have dangling tuples

Universal Relation Approach (Cont.)

- A particular decomposition defines a restricted form of incomplete information that is acceptable in our database.
  - Above decomposition requires at least one of customer-name, branch-name or amount in order to enter a loan number without using null values
  - Rules out storing of customer-name, amount without an appropriate loan-number (since it is a key, it can't be null either!)
- **Universal relation requires unique attribute names unique role assumption**
- Reuse of attribute names is natural in SQL since relation names can be prefixed to disambiguate names

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 ⋈ 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 a new table each year
  - company-year(company-id, earnings-2000, earnings-2001, earnings-2002)
    - Also in BCNF, but makes querying across years difficult and requires new attribute each year.
    - Is an example of a crosstab (交叉表), where values for one attribute become column names
    - Used in spreadsheets, and in data analysis tools

Quiz

- **Given the relational schema R<U, F>, U={A,B,C,D,E }, F={AC→BD, B → C, C → D, B → E}**
  a) Use Armstrong axioms and related rules to prove the functional dependency AC → E
  b) Compute $(A)^+$ and $(AC)^+$
  c) Find a canonical cover F_c of F
  d) Find all candidate keys, and point out R is in which normal form
  e) Decompose R into 3NF, which the decomposition is lossless-join and dependency preserving.
  f) Give related explanation or proof that the above decomposition is lossless-join and dependency preserving
  g) *Decompose the relation into relations in BCNF

Homework

- **Further Reading**
  - Chapter 7
- **Exercises**
  - 7.1, 7.2, 7.6
  - Any two from (7.30, 7.31, 7.32, 7.33, 7.34)
- **Submission**
  - Deadline: April 23 , 2024

End of Lecture 6