Introduction to Databases
《数据库引论》

Lecture 6: Relational Database Design Theory (1)
第6讲：关系数据库设计理论 (1)

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

Content of the Course

- **Part 0: Overview**
  - Lect. 0/1 (Feb. 20) - Ch1: Introduction
- **Part 1 Relational Databases**
  - Lect. 2 (Feb. 27) - Ch2: Relational model (data model, relational algebra)
  - Lect. 3 (Mar. 6) - Ch3: SQL (Introduction)
  - Lect. 4 (Mar. 13) – Ch4 & 5: Intermediate & Advanced SQL
- **Part 2 Database Design**
  - Lect. 5 (Mar. 20) - Ch6: Database design based on E-R model
  - Lect. 6 (Mar. 27) - Ch7: Relational database design (Part I)
  - Lect. 7 (Apr. 3) - Ch7: Relational database design (Part II)
- **Midterm exam: Apr. 10**
- **Part 3 Data Storage & Indexing**
  - Lect. 7 (Apr. 17) - Ch12/13: Storage systems & structures
  - Lect. 8 (Apr. 24) - Ch14: Indexing
- **Part 4 Query Processing & Optimization**
  - May 1, holiday, no classes
  - Lect. 9 (May 8) - Ch15: Query processing
  - Lect. 10 (May 15) - Ch16: Query optimization
- **Part 5 Transaction Management**
  - Lect. 11 (May 22) - Ch17: Transactions
  - Lect. 12 (May 29) - Ch18: Concurrency control
  - Lect. 13 (Jun. 5) - Ch19: Recovery system
  - Lect. 14 (Jun. 5) – Course review

Final exam: 13:00-15:00, Jun. 18

University Database

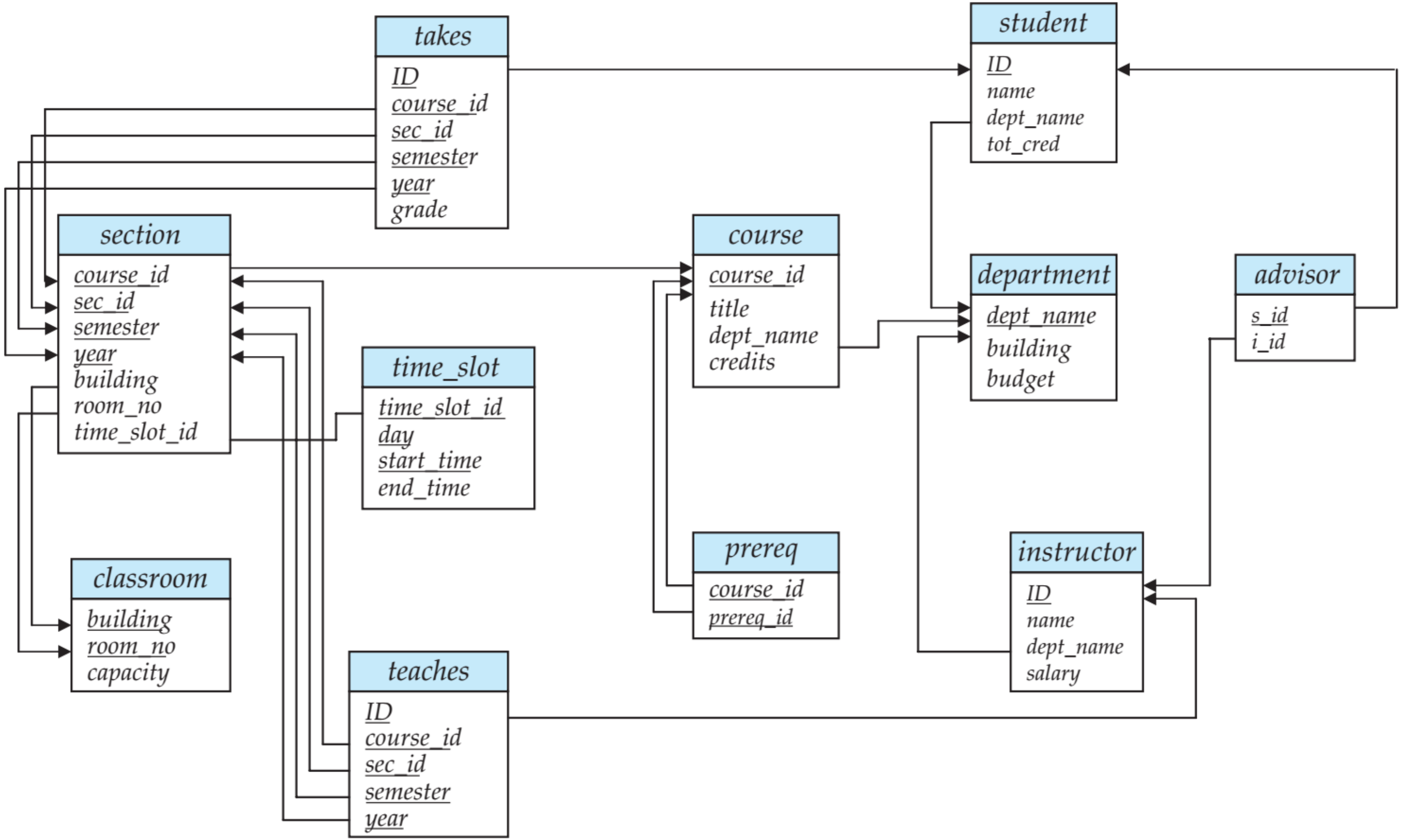
ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music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

Instructor table

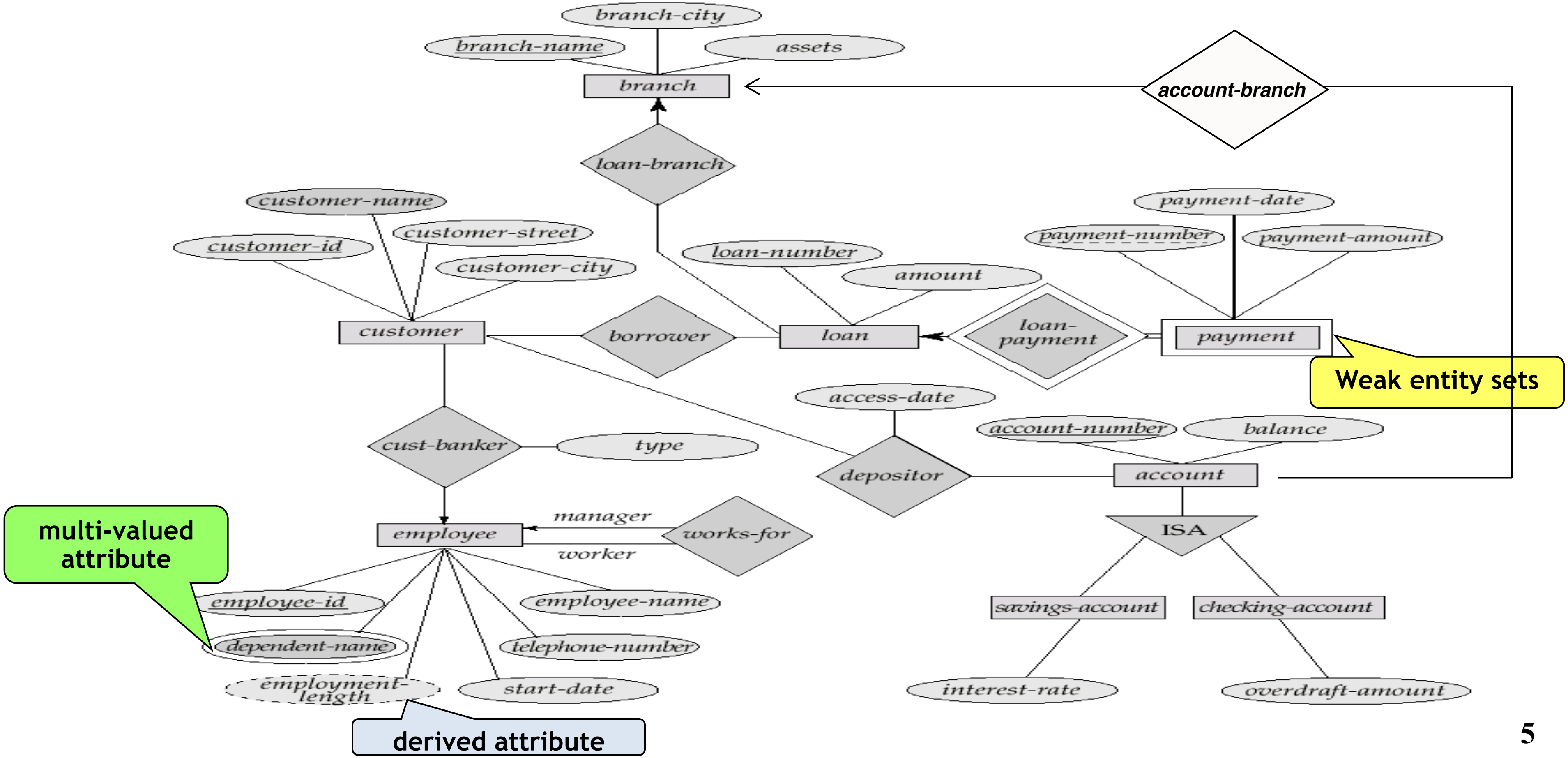
ID	*name*	*dept_name*	*tot_cred*
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
44553	Peltier	Physics	56
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.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

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）

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

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
⋮				

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）

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

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

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

$\neq$

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

A	B	C
α	1	1
α	2	1
β	1	1
β	2	1

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

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 $\{R_1, R_2, \dots, R_n\}$ 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 <u>superkey</u> for relation schema R **iff** $\boldsymbol{K \rightarrow R}$
- K is a <u>candidate key</u> for R **iff**
  - $\boldsymbol{K \rightarrow R}$, and
  - No $\boldsymbol{\alpha \subset K}$, $\boldsymbol{\alpha \rightarrow R}$
- 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

$$loan\_number \rightarrow customer\_name$$

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）

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

- $R = (A, B, C, G, H, I)$ $\quad F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$
- 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）

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:

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

- 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）

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 $\boldsymbol{F_c}$ 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},

$F=\{W\to Y, Y\to W, X\to WY, Z\to WY, XZ\to W\}$, F_c?

(1) $F=\{W\to Y, Y\to W, X\to WY, Z\to WY, XZ\to W\}$

(2) For RHS, $X\to \cancel{W}Y$ => $X\to Y$; $Z\to \cancel{W}Y$ => $Z\to Y$

$F=\{W\to Y, Y\to W, X\to Y, Z\to Y, XZ\to W\}$

(3) For LHS, $X\cancel{Z}\to W$ => $X\to W$

$F=\{W\to Y, Y\to W, X\to Y, Z\to Y, X\to W\}$

(4) Delete redundant FDs, $F=\{W\to Y, Y\to W, X\to Y, Z\to Y, X\to \cancel{W}\}$

$F_c = \{W\to Y, Y\to W, X\to Y, Z\to Y\}$ or $F_c = \{W\to Y, Y\to W, X\to W, Z\to W\}$

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 $R\langle U, F\rangle$, $U=\{X, Y, Z, W\}$, and $F=\{W\to Y, Y\to W, X\to WY, Z\to WY, XZ\to W\}$, find all candidate keys of R

  a) $F_c = \{W\to Y, Y\to W, X\to Y, Z\to Y\}$

  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 $F=\{AB \to C,\ C \to D,\ D \to A\}$, **find all candidate keys of R**

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

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）

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 $\{R_1, R_2, ..., R_n\}$ 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 $R = (A, B, C)$, $F = \{A \rightarrow C, B \rightarrow C\}$

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

lossy

r

A	B	C
α	1	1
α	2	1
β	1	1

$\neq$

A	B	C
α	1	1
α	2	1
β	1	1
β	2	1

$R_1 \cap R_2 \rightarrow R_1$?

$R_1 \cap R_2 \rightarrow R_2$?

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

A	C
α	1
β	1

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

B	C
1	1
2	1

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

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

lossless

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

A	B
α	1
α	2
β	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
- $R_1 = (A, B), \quad R_2 = (B, C)$
  - Lossless-join decomposition:

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

  - Dependency preserving
- $R_1 = (A, B), \quad R_2 = (A, C)$
  - Lossless-join decomposition:

$$R_1 \cap R_2 = \{A\} \text{ and } A \rightarrow AB$$

  - Not dependency preserving
    (cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

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:** $R < U, F >$, $U = \{A_1, A_2, \dots, A_n\}$, F, a decomposition of R: $\rho = \{R_1 < U_1, F_1 >, R_2 < U_2, F_2 >, \dots, R_k < U_k, F_k >\}$
- **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. $a_1 a_2 \dots a_n$), 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 $R = (A, B, C)$, $F = \{A \rightarrow C, B \rightarrow C\}$

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

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

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

$\neq$

A	B	C
α	1	1
α	2	1
β	1	1
β	2	1

$R_1 \cap R_2 \rightarrow R_1$?

$R_1 \cap R_2 \rightarrow R_2$?

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

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

A	B
α	1
α	2
β	1

lossless

$R_1 \cap R_2 \rightarrow R_2$

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

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
- $R_1 = (A, B), \quad R_2 = (B, C)$
  - Lossless-join decomposition:

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

  - Dependency preserving
- $R_1 = (A, B), \quad R_2 = (A, C)$
  - Lossless-join decomposition:

$$R_1 \cap R_2 = \{A\} \text{ and } A \rightarrow AB$$

  - Not dependency preserving
    (cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

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 using Functional Dependencies

- When we decompose a relation schema R with a set of FDs F into $R_1, R_2,\ldots, R_n$ 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 $\boldsymbol{\alpha} \rightarrow \boldsymbol{\beta}$ is preserved in a decomposition of $\boldsymbol{R}$ into $\boldsymbol{R_1}$, $\boldsymbol{R_2}$,…, $\boldsymbol{R_n}$, 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 $\boldsymbol{\beta}$, then the functional dependency $\boldsymbol{\alpha} \rightarrow \boldsymbol{\beta}$ 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 $(\boldsymbol{F_1} \cup \boldsymbol{F_2} \cup \cdots \cup \boldsymbol{F_n})^+$

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 $R_1 \bowtie R_2$)

End of Lecture 6