关系数据库设计优化
Relational Database Design Optimization

李文根/Wengen Li

Email: lwengen@tongji.edu.cn

先进数据与机器智能系统实验室

Advanced Data and Machine Intelligence Systems (ADMIS) Lab

https://admis-tongji.github.io

同济大学 计算机科学与技术学院

2025年03月

课程概要

- **Part 0: Overview**
  - Ch1: Introduction
- **Part 1 Relational Languages**
  - Ch2: Relational model
  - Ch3: Introduction to SQL
  - Ch4: Intermediate SQL
  - Ch5: Advanced SQL
- **Part 2 Database Design**
  - Ch6: Database design via E-R model
  - **Ch7: Relational database design**
- **Part 3 Application Design & Development**
  - Ch8: Complex data types
  - Ch9: Application development
- **Part 4 Big Data Analytics**
  - Ch10: Big data
  - Ch11: Data analytics
- **Part 5 Storage Management & Indexing**
  - Ch12: Physical storage systems
  - Ch13: Data storage structures
  - Ch14: Indexing
- **Part 6 Query Processing & Optimization**
  - Ch15: Query processing
  - Ch16: Query optimization
- **Part 7 Transaction Management**
  - Ch17: Transactions
  - Ch18: Concurrency control
  - Ch19: Recovery system
- **Part 8 Parallel & Distributed Database**
  - Ch20: Database system architecture
  - Ch21-23: Parallel & distributed storage, query processing & transaction processing
- **Advanced topics**
  - DB Platform: **OceanBase**, MongoDB, Neo4J
  - …

目录

大关系模式

- in_dep (ID, name, salary, dept_name, building, budget)
  - instructor + department

ID	name	salary	dept_name	building	budget
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

in_dep

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

course_id	title	dept_name	credits
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

course

大关系模式

- in_dep (ID, name, salary, dept_name, building, budget)
  - **Redundant**（冗余）: building, budget
  - **Inconsistent**（不一致）: building, budget
  - **Insert failure**: cannot insert a tuple without ID, name, salary
  - **Difficult to check attribute dependency**: dept_name → budget

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

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

小关系模式

TONGJI UNIVERSITY

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

良好的关系模式

- **RDB design is to find a collection of "good" schemas. A bad design may lead to**
  - Repetition of information
  - Inability to represent certain information

- **Design goals**
  - Avoid redundant data
  - The relationships among attributes are represented
  - Facilitate the checking for violation of database integrity constraints

- **Normal Forms, 范式**
  - Decide whether a particular relation R is in good form
- **Decomposition**
  - If 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**
- The method is based on:
  - **functional dependency**
  - multi-valued dependency*

目录

- **良好关系的特征**
- **函数依赖**
  - **基于函数依赖的关系分解**
  - 函数依赖闭包
  - 属性闭包
  - 正则覆盖
  - 无损连接分解
  - 依赖保持
- **规范化与范式**
- **多值依赖***
- **数据库设计过程**

函数依赖（Functional Dependency）

- **Functional dependency**
  – The value for a certain set of attributes determines uniquely the value for another set of attributes
  – A functional dependency is a generalization of the notion of key

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

函数依赖（续）

- 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 relation $r(R)$, 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., considering $r(A, B)$ with the following instance of r, the functional dependency $A \rightarrow B$ does NOT hold, but $B \rightarrow A$ holds

A	B
1	4
1	5
3	7

函数依赖（续）

- 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 $\alpha \subset K$, $\alpha \rightarrow R$
- FD allows us to express constraints that cannot be expressed using superkeys. Consider the following schema:

  in_dep (ID, name, salary, dept_name, building, budget)

  We expect the following FDs to hold:

  dept_name $\rightarrow$ *building*
  dept_name $\rightarrow$ *budget*

函数依赖的应用

- Functional dependency can be used 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 *classroom* may satisfy

    room_number → capacity

building	*room_number*	*capacity*
Packard	101	500
Painter	514	10
Taylor	3128	70
Watson	100	30
Watson	120	50

函数依赖（续）

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

  building, room_number → *capacity*

  dept_name, building → *building*

  dept_name → *dept_name*

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

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

分解（Decomposition）

- Decompose the relation schema **in_dep** into:

  instructor(ID, name, salary, dept_name)

  department(dept_name, building, budget)

- 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: $r = \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$

无损连接分解（Lossy-join Decomposition）

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

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

$\neq$

A	B	C
α	1	1
α	2	1
β	1	1
β	2	1

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

目录

函数依赖集的闭包

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

函数依赖集的闭包（续）

- Further simplify the manual computation of F^+ with 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.

例

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

计算F^+

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

$F^+ = F$

repeat
  for each FD f **in** F^+
    apply reflexivity and augmentation 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
      add the resulting FD to F^+
until F^+ does not change any more

NOTE: We will see an alternative way for this task later.

F^+

$R(X,Y,Z)$, $F = \{X\to Y,\ Y\to Z\}$, F^+ ?

$F^+ = \{$

$X\to\Phi$, $Y\to\Phi$, $Z\to\Phi$, $XY\to\Phi$, $XZ\to\Phi$, $YZ\to\Phi$, $XYZ\to\Phi$,

$X\to X$, $Y\to Y$, $Z\to Z$, $XY\to X$, $XZ\to X$, $YZ\to Y$, $XYZ\to X$,

$X\to Y$, $Y\to Z$, $XY\to Y$, $XZ\to Y$, $YZ\to Z$, $XYZ\to Y$, $X\to Z$,

$Y\to YZ$, $XY\to Z$, $XZ\to Z$, $YZ\to YZ$, $XYZ\to Z$, $X\to XY$, $XY\to XY$,

$XZ\to XY$, $XYZ\to XY$, $X\to XZ$, $XY\to YZ$, $XZ\to XZ$, $XYZ\to YZ$, $X\to YZ$,

$XY\to XZ$, $XZ\to YZ$, $XYZ\to XZ$, $X\to XYZ$, $XY\to XYZ$, $XZ\to XYZ$, $XYZ\to XYZ\}$

Note: computing F^+ is an NP-hard problem

目录

TONGJI UNIVERSITY

属性集的闭包（Closure of Attribute Set）

- Given a set of attributes α, the closure of α under F (denoted by α^+) is the set of **attributes** that are functionally determined by α under F:

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

- **Algorithm to compute α^+**

  result:=α
  while (changes to result) **do**
      for each $\beta \rightarrow \gamma$ in F **do**
          begin
              if $\beta \subseteq$result, **then** result:=result $\cup$ γ
          end

属性集的闭包：例1

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 $X^{(2)}$, 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}

属性集的闭包：例2

- $R = (A, B, C, G, H, I)$, $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$
- Calculate $(AG)^+$
  - result = AG
  - result = ABCG ($A \rightarrow B$ and $A \rightarrow C$)
  - result = ABCGHI ($CG \rightarrow H$ and $CG \rightarrow I$)

- **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 $(A)^+ \supseteq R$
    - Does $G \rightarrow R$? == Is $(G)^+ \supseteq R$

属性集闭包的应用

- **Test for superkey**

- **Test functional dependencies**
  - To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is in F^+), just check if $\beta \subseteq \alpha^+$
  - A simple and cheap test

- **Compute the closure of $\boldsymbol{F}$**
  - For each subset $\gamma \subseteq R$, find the attribute closure γ^+, and for each $S \subseteq \gamma^+$, output a functional dependency $\gamma \rightarrow S$

目录

函数依赖中的冗余

- One set of FDs may contain 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 $\{A \rightarrow B, B \rightarrow C, A \rightarrow D\}$

无关属性（Extraneous Attributes）

- Consider a set F of FDs and the FD $\alpha \to \beta$ in F
  - Attribute A is extraneous (无关的) in α if $A \in \alpha$ and F logically implies $(F - \{\alpha \to \beta\}) \cup \{(\alpha - A) \to \beta\}$
  - Attribute A is extraneous in β if $A \in \beta$ and the set of FDs $(F - \{\alpha \to \beta\}) \cup \{(\alpha \to (\beta - A)\}$ logically implies F

- E.g., given $F = \{A \to C, AB \to C\}$
  - B is extraneous in $AB \to C$ because $\{A \to C, AB \to C\}$ logically implies $A \to C$ (i.e., the result of dropping B from $AB \to C$)
- E.g., given $F = \{A \to C, AB \to CD\}$
  - C is extraneous in $AB \to CD$ since it can be inferred from $= \{A \to C, AB \to D\}$

检查无关属性

同济大学 TONGJI UNIVERSITY

- Consider a set F of FDs and $\alpha \rightarrow \beta$ in F
- To test if attribute $A \in \alpha$ is extraneous in α
  - compute $(\{\alpha\} - A)^+$ using the dependencies in F
  - if $(\{\alpha\} - A)^+$ contains β, A is extraneous

- To test if attribute $A \in \beta$ is extraneous in β
  - compute α^+ using only the dependencies in $F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$
  - if α^+ contains A, A is extraneous

正则覆盖（Canonical Cover）

TONGJI UNIVERSITY

- **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 $\alpha_1 = \alpha_2$ (**optional**)

- To compute a canonical cover for F:

$F_c = F$
repeat
    use the union rule to replace $\alpha_1 \rightarrow \beta_1$ *and* $\alpha_1 \rightarrow \beta_2$ *with* $\alpha_1 \rightarrow \beta_1 \beta_2$ *in F*
    find a FD $\alpha \rightarrow \beta$ *with an extraneous attribute either in* α *or in* β
        if an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$
until F_c *does not change*

计算正则覆盖

- $R = (A, B, C)$, $F = \{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$, F_c?
  - Combine $A \rightarrow BC$ and $A \rightarrow B$ into $A \rightarrow BC$
    - F_c 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
    - F_c 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 $B \rightarrow C$
  - The canonical cover is: $F_c = \{A \rightarrow B, B \rightarrow C\}$
  - 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 $F_c = \{A \rightarrow C, B \rightarrow AC, C \rightarrow B\}$

计算正则覆盖：例1

- R<U,F>, U={A,B,C}, F={A→BC, B→C, A→B, AB→C}. F_c?
  a) For each FD in F： F={A→B, A→C, B→C, A→B, AB→C}
  b) One A→B can be deleted, then F={A→B, A→C, B→C, AB→C}
  c) A→C is implied by A→B and B→C. So A→C is extraneous. Then F={A→B, B→C, AB→C}
  d) AB→C is implied by B→C, then AB→C is extraneous, so we have F_c={A→B, B→C}

计算正则覆盖：例2

- $R<U, F>$, $U=\{X, Y, Z, W\}$, $F=\{W\to Y, Y\to W, X\to WY, Z\to WY, XZ\to W\}$, F_c?

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

  (2) For LHS, X in $XZ\to W$ is redundant, $F=\{W\to Y, Y\to W, X\to W, X\to Y, Z\to W, Z\to Y\}$

  (3) Delete redundant FDs,

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

寻找候选码

- For $H(A_1, A_2, \ldots, A_n)$ 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 (**Rare**)
  - LR： exists in both LHS and RHS

寻找候选码（续）

- **Algorithm：** find candidate keys for relation H
- **Input：** H and its FD set F
- **Output：** all candidate keys for H

思考：为什么不考虑仅存在函数依赖右边的属性?

(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 H, then X is the only candidate key for H, then goes to (5); otherwise goes to (3)

(3) Take attribute A from Y, compute $(XA)^+$. If $(XA)^+$ contains all attributes of H, then XA is a candidate key for H. 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 H

(5) Finish and output the result

寻找候选码：例1

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

寻找候选码：例2

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

  a) F_c={AB→C, C→D, D→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 the candidate keys of R

寻找候选码：例3

- 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}^{+}={ ? }

  (4) …

寻找候选码：例3

- 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={S→D, D→S, I→B, B→IO, O→B}

(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)、(QDO)、(QSB)、(QDB)、(QSI)、(QDI)

目录

分解（Decomposition）

in_dep_schema *(ID, name, salary, dept_name, building, budget)*

- Decompose the relation schema *in_dep_schema* into two relations:
  - ***instructor****(ID, name, salary, dept_name)*
  - ***department****(dept_name, building, budget)*

分解（续）

- All attributes of an original schema R must appear in the decomposition (R_1, R_2), i.e., $R = R_1 \cup R_2$
- Lossless-join decomposition(无损分解): For all possible relations r on schema R, $r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r)$
- A decomposition of R into R_1 and R_2 is lossless-join iff at least one of the following dependencies is in F^+:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$

例1

- R = (A, B, C), F = {A →B, B → 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 $B \rightarrow BC$
- $R_1 = (A, B)$, $R_2 = (A, C)$
  - Lossless-join decomposition: $R_1 \cap R_2 = \{A\}$ and $A \rightarrow AB$

例2

- Given R<U,F>, U={A,B,C,D,E}, F={AB→C, C→D, D→E}, and a decomposition ρ of R into:
  - $R_1(A,B,C)$, $R_2(C,D)$, $R_3(D,E)$
  - Is ρ a lossless-join decomposition or a lossy one?

无损连接分解的检测

- **Input:** $R < U, F >$, $U = \{A_1, A_2, \ldots, A_n\}$, 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 $A_j (1 \leq j \leq n)$, and each row corresponds to a schema $R_i (1 \leq i \leq k)$. 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

例2（续）

- Given R<U, F>, U={A, B, C, D, E}, F={AB→C, C→D, D→E}, and a decomposition ρ of R into: R_1(A, B, C), R_2(C, D), R_3(D, E). Is ρ a lossless-join decomposition or a lossy one?

(1) First, construct a table as:

	A	B	C	D	E
R_1(A,B,C)	a_1	a_2	a_3	b_{14}	b_{15}
R_2(C,D)	b_{21}	b_{22}	a_3	a_4	b_{25}
R_3(D,E)	b_{31}	b_{32}	b_{33}	a_4	a_5

例2（续）

- (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 R_1, R_2, and R_3 is a lossless-join one.

	A	B	C	D	E
$R_1(A,B,C)$	a_1	a_2	a_3	B14 → a_4	b15 → a_5
$R_2(C,D)$	b_{21}	b_{22}	a_3	a_4	b25 → a_5
$R_3(D,E)$	b_{31}	b_{32}	b_{33}	a_4	a_5

例3

- R<U,F>, U={A,B,C,D,E}, F={A→C, B→C, C→D, DE→C, CE→A}, and a decomposition of R: ρ={R_1(A,D), R_2(A,B), R_3(B,E), R_4(C,D,E), R_5(A,E)}. ρ is a lossless-join decomposition or a lossy one ?
  - ρ is a lossless-join decomposition

	A	B	C	D	E
R_1	a1	b12	b13	a4	b15
R_2	a1	a2	b13	a4	b25
R_3	a1	a2	a3	a4	a5
R_4	a1	b42	a3	a4	a5
R_5	a1	b52	a3	a4	a5

例4

- R<U,F>, U={A,B,C,D}, F={A→B, A→C, C→D}, ρ={R_1(A,B), R_2(B,C), R_3(C,D)}. ρ is a lossless-join decomposition or a lossy one?
  - ρ is a lossy-join decomposition

	A	B	C	D
R_1	a_1	a_2	b_{13}	b_{14}
R_2	b_{21}	a_2	a_3	$\cancel{b_{24}}a_4$
R_3	b_{31}	b_{32}	a_3	a_4

目录

分解的目标

TONGJI UNIVERSITY

- When decomposing 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

例

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

If each member of *F* can be tested on one of the relations of the decomposition, then the decomposition is dependency preserving.

Sufficient condition!!

依赖保持检测

TONGJI UNIVERSITY

- 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 testing

  result = α
  while (changes to result) do
      for each R_i *in the decomposition*
          t = (*result* ∩ R_i)$^+$ ∩ R_i
          result = *result* ∪ 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 $(F_1 \cup F_2 \cup \cdots \cup F_n)^+$

目录

规范化（Normalization）

- **Normalization**: The process of decomposing relations with anomalies to
  - produce **smaller** and **well-structured** relations
  - improve the logical design so that it satisfies certain constraints to avoid unnecessary **duplication** of data

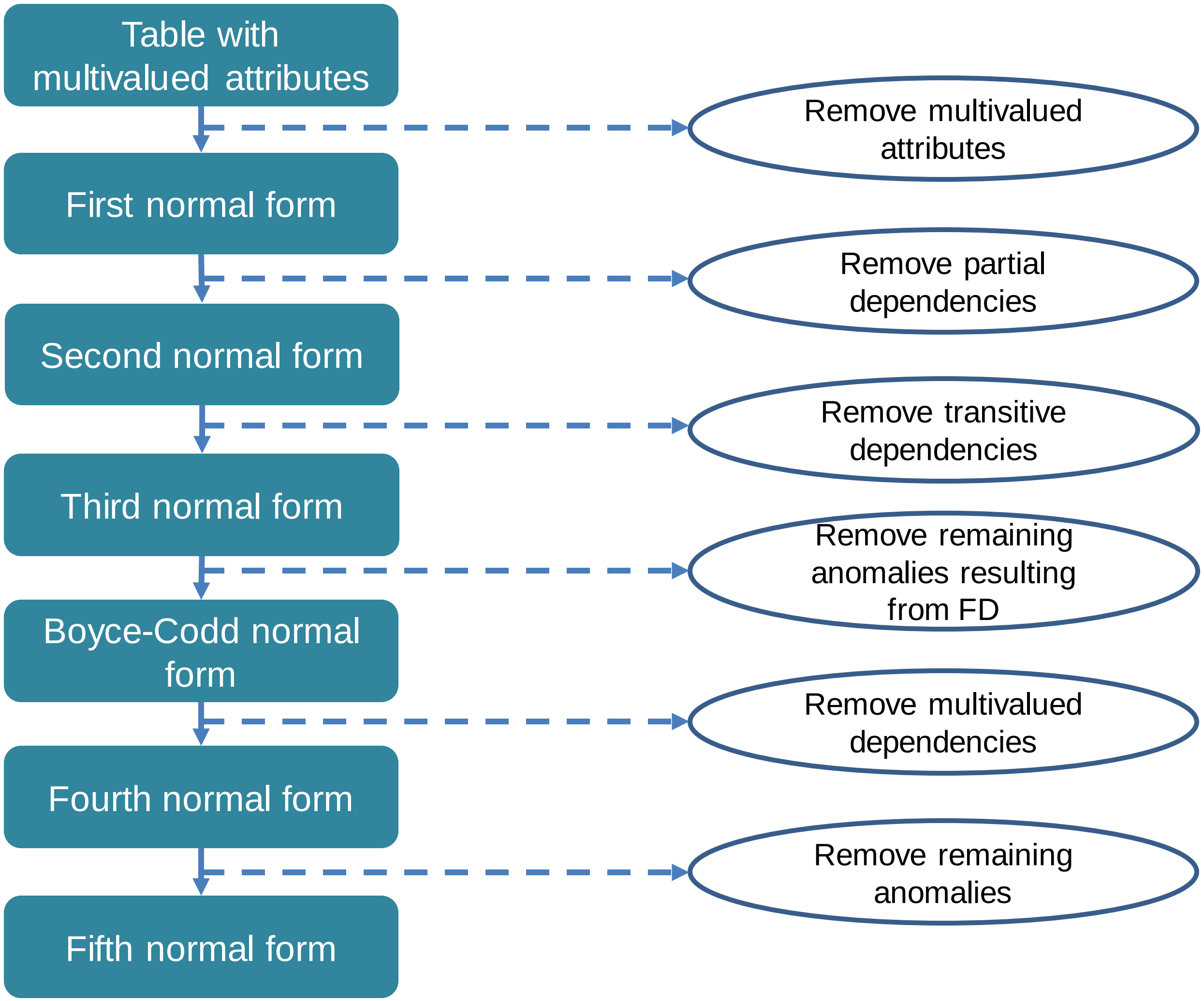
- The problems of having duplication of data
  - waste of space
  - difficulty in consistency control

具有良好结构的关系

- A relation that contains minimal data redundancy and allows users to insert, delete, and update rows without causing data inconsistencies
  - **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

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

Levels of Normalization



第一范式（1st Normal Form）

- **Atomic domain**
  - its elements are considered to be indivisible units
  - attributes do not have any substructures

- **1NF**
  - A relation schema R is in 1NF if the domains of all attributes of R are atomic
  - Non-atomic values, e.g., composite attribute/ multivalued attributes, complicate the storage and may result in redundant storage of data

第一范式（续）

- **Atomicity (原子性) 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 identify the department, the domain of roll numbers is not atomic
      - Doing so is a bad idea: lead to encoding of information in application program rather than in the database

第一范式（续）

- **Requirements**
  - No multivalued attributes
  - Every attribute value is atomic
- **例：**
  - Table 1 is not in 1st Normal Form (multivalued attributes)
  - Table 2 is in 1st Normal form

Table 1

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

Table 2

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

第二范式

- **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 all) of the primary key

关系Employee中的函数依赖

Dependency on the entire primary key

<u>EmpID</u>	<u>CourseTitle</u>	Name	DeptName	Salary	DateCompleted

Dependency on only *part* of the primary key

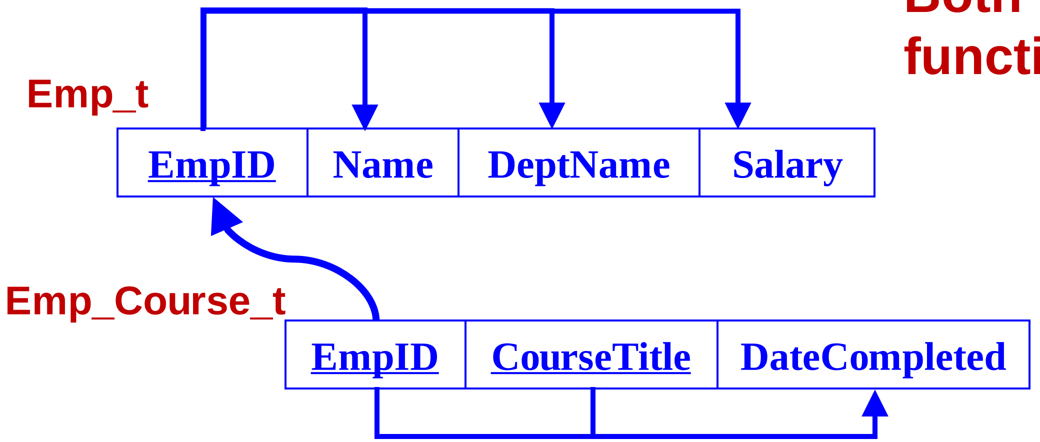
EmpID, CourseTitle ➔ DateCompleted

EmpID ➔ Name, DeptName, Salary

NOT in 2nd Normal Form!

将关系分解为第二范式

- Decompose the relation into two separate relations



Both relations are full functional dependencies

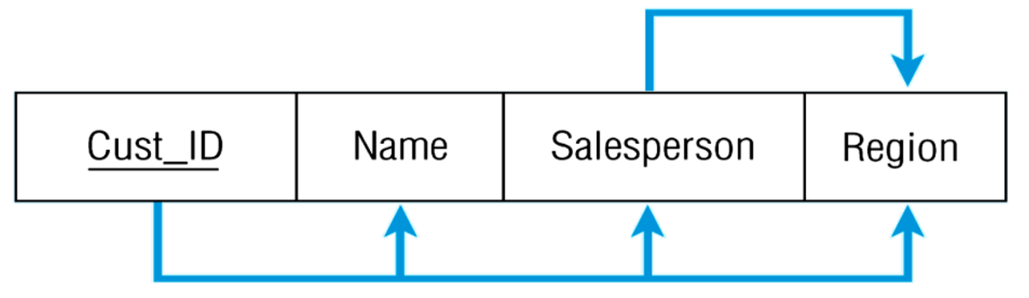
- **Requirements**
  - 2NF
  - **No transitive dependencies**
- **Transitive dependency**: functional dependency between two (or more) non-key attributes

存在传递依赖的关系

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

存在传递依赖的关系（续）



Cust_ID → Name
Cust_ID → Salesperson
Cust_ID → Region

(2nd NF)

BUT

Cust_ID → Salesperson → Region

Transitive dependency
(not 3rd NF)

存在传递依赖的关系（续）

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

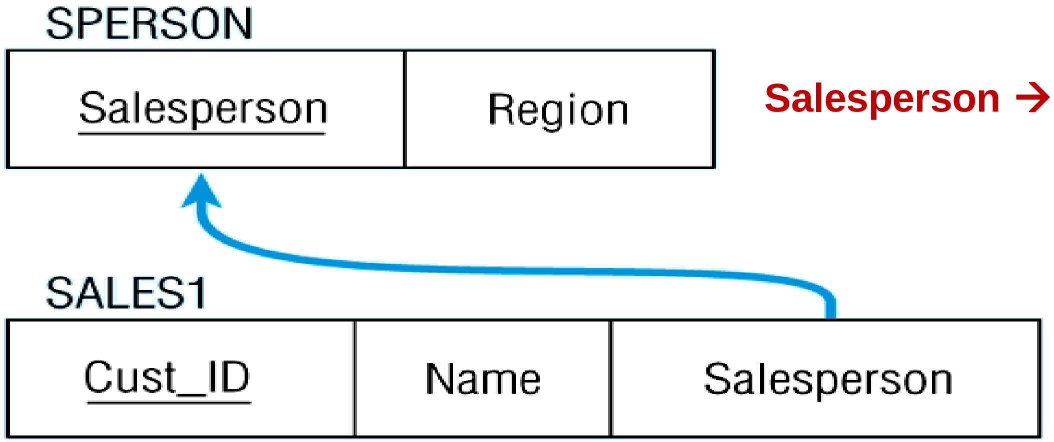
Sales_new

Salesperson	Region
Smith	South
Hicks	West
Hernandez	East
Faulb	North

Salesperson

Decompose the Sales relation

满足3NF的关系



Salesperson → Region

Cust_ID → Name

Cust_ID → Salesperson

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

规范化

- **1st NF**
  - No multivalued attributes, and every attribute value is atomic
  - All relations are in 1st Normal Form
- **2nd NF**
  - 1NF + every non-key attribute is fully functionally dependent on the **ENTIRE** primary key
- **3rd NF**
  - 2NF + no transitive dependency
- **Boyce-Codd NF**
  - All determinants (决定属性) are superkeys
- **4th NF**
  - No multivalued dependencies
- **5th NF**
  - 投影-连接范式（project-join normal from, PJNF）
  - Minimize redundancy by separating semantically connected relationships

BC范式（Boyce-Codd Normal Form）

- Given relation schema R and FD set F, R is in BCNF if at least one of the following condition holds for every FD $\alpha \rightarrow \beta$ in F^+:
  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
  - α is a superkey for R

例

- $R = (A, B, C)$, $F = \{A \rightarrow B, B \rightarrow C\}$, $Key = \{A\}$
  - R is not in BCNF since $B \rightarrow C$ 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

BCNF检测

- To check if a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF
  - Compute α^+, and
  - Verify that it includes all the attributes of R, i.e., a superkey of R

- **Simplified test**
  - Check only the FDs **F** for violation of BCNF, rather than checking all the 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

BCNF检测（续）

- Checking only F is **incorrect** when testing a relation in a **decomposition** of R
  - E.g., consider R(A, B, C, D) with F = {$A \rightarrow B$, $B \rightarrow C$}
    - 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) and we might be mislead into thinking that R_2 satisfies BCNF
    - However, dependency $A \rightarrow C$ in F^+ shows that R_2 is not in BCNF

BCNF分解检测

- **To check if a relation $\boldsymbol{R_i}$ in a decomposition of $\boldsymbol{R}$ is in BCNF**
  - either test R_i for BCNF w.r.t. the restriction of F to R_i (i.e., 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的超码）
    - If the condition is violated by some $\alpha \subseteq R_i$, the FD $\alpha \rightarrow (\alpha^+ - \alpha) \cap R_i$ can be shown to hold on R_i, and R_i violates BCNF

BCNF分解算法

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 $\alpha \rightarrow \beta$ be a non-trivial FD that holds on R_i
      such that α^+ does not contain R_i and $\alpha \cap \beta = \varnothing$;
      result := $(result - R_i) \cup (R_i - \beta) \cup (\alpha, \beta)$;
    end
  else *done* := true;

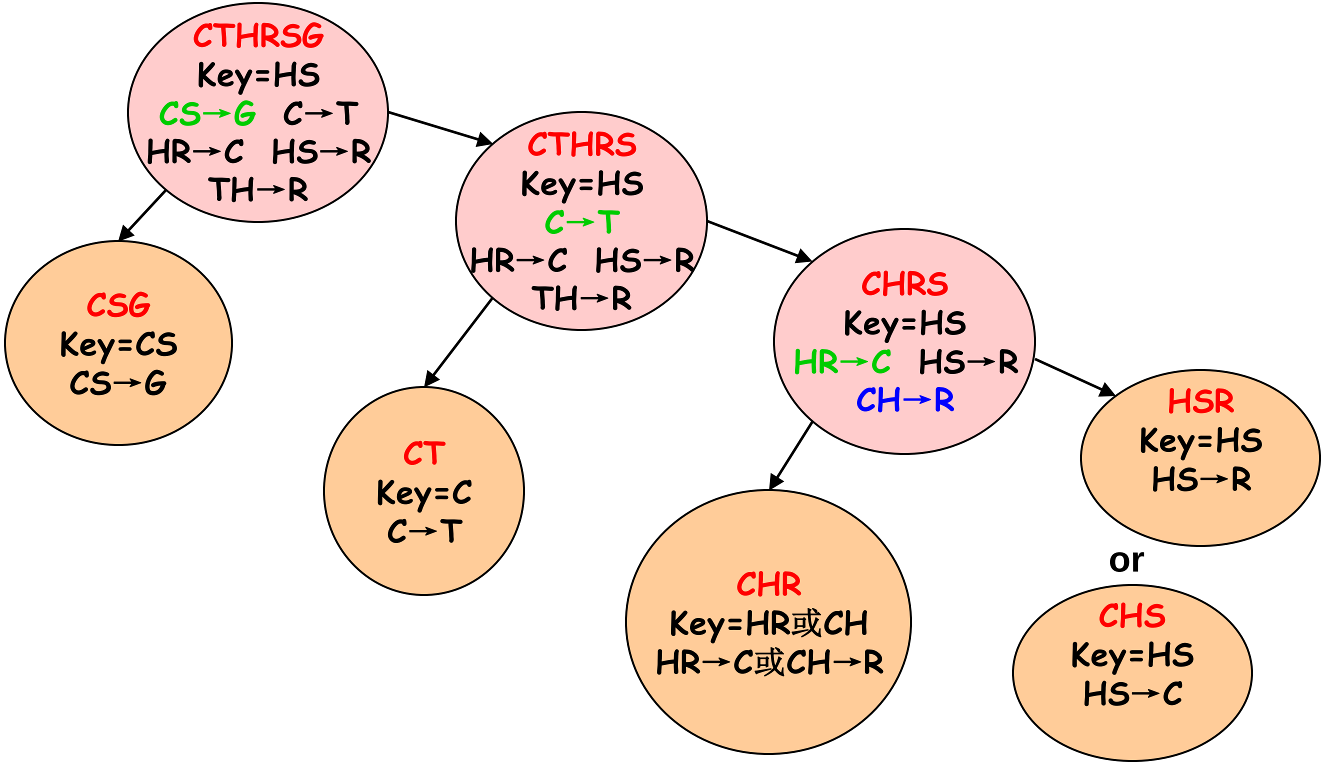
Note: each R_i is in BCNF, and the decomposition is lossless-join.

例

- Consider the relation scheme {C, T, H, R, S, G}, where C=course, T=teacher, H=hour, R=room, S=student, and G=grade.
- The functional dependencies F 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

分解树

- $\alpha \to \beta$ holds on R_i such that α^+ does not contain R_i and $\alpha \cap \beta = \varnothing$; *result* := (*result* – R_i) $\cup$ (R_i – β) $\cup$ (α, β);



BCNF和依赖保持

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

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

第三范式

- There are some situations where
  - BCNF is not dependency preserving, but
  - Efficient checking for FD violation on updates is important
- **Solution: define a weaker normal form, i.e., Third Normal Form**
  - Allow some redundancy
  - But FDs can be checked on individual relations without computing a join
  - **There is always a lossless-join and dependency-preserving decomposition into 3NF**

第三范式（续）

- A relation schema R is in 3NF if for every $\alpha \rightarrow \beta$ in F^+ at least one of the following conditions 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 different candidate keys)
- If a relation is in BCNF, it is in 3NF (since one of the first two conditions above must hold in BCNF)
- The third condition is a relaxation of BCNF to ensure dependency preservation

第三范式（续）

- **Example**
  - R = (J, K, L), F = {JK $\rightarrow$ L, L $\rightarrow$ K}, two candidate keys: JK and JL
  - R is in 3NF

    JK $\rightarrow$ L JK is a superkey

    L $\rightarrow$ K K is contained in a candidate key
- There is some redundancy in this schema
- Equivalent to example:

  banker-schema = (branch_name, customer_name, banker_name)

  banker_name $\rightarrow$ *branch_name*

  branch_name, customer_name $\rightarrow$ *banker_name*

第三范式检测

- Use attribute closure to check for each dependency $\alpha \rightarrow \beta$ to see if α is a superkey.
- If α is not a superkey, further verify if each attribute in β-α is contained in a candidate key of R
  - this test is rather more expensive, since it involve finding candidate keys
  - testing for 3NF is NP-hard
  - decomposition into 3NF can be done in polynomial time

3NF分解算法

- Let F_c be a canonical cover for F;
  $i := 0$;
  for each FD $\alpha \rightarrow \beta$ in F_c do
  $\quad i := i + 1$;
  $\qquad R_i := \alpha\,\beta$
  if none of the schemas R_j, $1 \le j \le i$ contains a candidate key for R
  $\quad i := i + 1$;
  $\qquad R_i :=$ any candidate key for R;
  return $(R_1, R_2, \ldots, R_i)$

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

3NF分解示例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：

- All attributes exist in F, and does not exist X→Y ∈ F and XY=U
- So decompose R into (Same LHS attributes):
  - U1={A,B,C}, F1={AC → B, B → C}
  - 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. Since candidate keys AC、AB are all in U1, a row can be found as a1, a2, a3, a4, a5 for testing lossless-join form. So ρ is lossless-join.

3NF分解示例2

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

  1) $Fc=\{A \to C, C \to A, B \to A, D \to A\}$

  2) Candidate keys of R： BD; key-attributes: B、D;

  - For $B\to A$ and $D\to A$, non-key attribute A is partial dependent on key BD, so $R\notin 2NF$, $R\in 1NF$

  3) Decompose R into 3NF：

  - All attributes exist in F, and does not exist $X\to Y \in F$ and $XY=U$
  - So decompose R into (Same LHS attributes):
    - $U1=\{A,C\}$, $F1=\{ A \to C, C \to A \}$
    - $U2=\{A,B\}$, $F2=\{ B \to A \}$
    - $U3=\{A,D\}$, $F3=\{ D \to A \}$
  - $\rho=\{R1<U1,F1>, R2<U2,F2>, R3<U3,F3>\}$, the decomposition is dependency preserving. But candidate key BD is not in any Ui, so $\tau = \rho \cup \{R4<\{B,D\},\Phi>\}$, and τ is the decomposition that is dependency preserving and lossless-join

例

- Relation schema:

  banker-info-schema = (*branch-name*, *customer-name*, *banker-name*)

- The FDs for this relation schema are:

  banker-name $\rightarrow$ *branch-name*
  customer-name, *branch-name* $\rightarrow$ *banker-name*

- The key is:

  {*customer-name*, *branch-name*}

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

  banker-office-schema = (*banker-name*, *branch-name*)
  banker-schema = (*customer-name*, *branch-name*, *banker-name*)

BCNF和3NF对比

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

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

BCNF和3NF对比（续）

- The redundancy in 3NF
  - $R = (J, K, L)$
    $F = \{JK \rightarrow L,\ L \rightarrow K\}$

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
null	l_2	k_2

- A schema that is in 3NF but not in BCNF has the repetition issue
  - e.g., the relationship l_1, k_1

关系数据库设计目标

- Goal for a relational database design
  - BCNF
  - lossless join
  - dependency preservation

- If cannot achieve this, we accept one of
  - lack of dependency preservation or
  - redundancy due to use of 3NF

关系数据库设计目标（续）

- SQL does not provide a direct way to specify FDs other than superkeys
  - can specify FDs using assertions, but they are expensive to test

- For a dependency preserving decomposition, we would not be able to use SQL to efficiently test a FD whose left hand side is not a key
  - R = (J, K, L), F = {JK $\rightarrow$ L, L $\rightarrow$ K}

跨关系函数依赖检测

- 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 new database systems support materialized views and maintain the view when the relations are updated
  - No extra coding effort for programmers

跨关系函数依赖检测（续）

- The functional dependency $\alpha \rightarrow \beta$ is expressed by declaring α as a candidate key on the materialized view
- Checking for candidate key is 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

目录

- **良好关系的特征**
- **函数依赖**
  - 基于函数依赖的关系分解
  - 函数依赖闭包
  - 属性闭包
  - 正则覆盖
  - 无损连接分解
  - 依赖保持
- **规范化与范式**
- **多值依赖***
- **数据库设计过程**

多值依赖（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)∈ 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

多值依赖（续）

- 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)

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

多值依赖（续）

- Therefore, it is better to decompose classes into:

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

teaches

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

text

We shall see that these two relations are in 4NF

多值依赖（续）

- 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: $t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$

$t_3[\beta] = t_1[\beta]$

$t_3[R-\beta] = t_2[R-\beta]$

$t_4[\beta] = t_2[\beta]$

$t_4[R-\beta] = t_1[R-\beta]$

多值依赖（续）

TONGJI UNIVERSITY

- 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

多值依赖（续）

- **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 $X \twoheadrightarrow Y-Z$, $X \twoheadrightarrow Z-Y$
  - …

例

- 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

例（续）

- In our example:
  - course $\twoheadrightarrow$ teacher
  - course $\twoheadrightarrow$ 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 $\twoheadrightarrow$ Z
  - Indeed we have (in above notation) $z_1 = z_2$
    The claim follows

多值依赖的使用

- We use MVDs in two ways:
  - To test relations to determine whether they are legal under a given set of FDs and MVDs
  - 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

多值依赖理论

- 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

第四范式

- 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

多值依赖的局限

- 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

    $$\alpha \twoheadrightarrow \beta(\beta \cap R_i)$$

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

4NF分解算法

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 α $\twoheadrightarrow \beta$ 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) ∪ ((R_i -β) ∩ (α, β));
   end
  else done:= true;

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

例

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

  $F = \{ A \rightarrow\rightarrow B$

  $B \rightarrow\rightarrow HI$

  $CG \rightarrow\rightarrow H \}$
- R is not in 4NF since $A \rightarrow\rightarrow B$ 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 $A \rightarrow\rightarrow B$ and $B \rightarrow\rightarrow HI$, $A \rightarrow\rightarrow HI$, $A \rightarrow\rightarrow I$

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

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

其他范式

- 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

目录

TONGJI UNIVERSITY

E-R模型和规范化

- 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 dept_name and dept_address, and an FD
    dept_name → *dept_address*
  - Good design would have made department an entity
- FDs from non-key attributes of a relationship set are possible, but rare

去规范化（Denormalization）

- Use non-normalized schema for performance
  - E.g., displaying customer-name along with account-number and balance requires the join of account with depositor
  - **Alternative 1**: use a denormalized relation that contains attributes of both account and depositor
    - faster lookup
    - extra space and extra execution time for updates
    - extra coding work for programmers and possibility of error in extra code
  - **Alternative 2**: use a materialized view defined as account $\bowtie$ depositor
    - benefits and drawbacks are the same as above, except no extra coding work for programmers and avoids possible errors

其他设计问题

- **Some aspects of database design are not caught by normalization**
- E.g., instead of earnings(company-id, year, amount), use
  - earnings-2020, earnings-2021, earnings-2022, 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-2020, earnings-2021, earnings-2022)
    - Also in BCNF, but make querying across years difficult and requires new attribute each year.
    - Is an example of a crosstab, where values for one attribute become column names
    - Used in spreadsheets, and in data analysis tools