

数据库技术-规范化方法(NF)

赵亚伟

zhaoyw@ucas.ac.cn

中国科学院大学 大数据分析技术实验室

2017.11.26

目录

- 问题导引
- 数据依赖
- Armstrong公理系统
- 模式分解
- 规范化方法
 - 1NF
 - 2NF
 - 3NF
 - BCNF
 - 4NF（自学）
- 数据库设计问题

问题导引

- 假设通过ER模型转化得到一个关系模式（泛关系）：
**Lending_schema(Branch_name、Branch_city、Assets、
Customer_name、Loan_number、Amount)**

一个实例

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
Mianus	Houseneck	400000	Curry	L-93	500
Round hill	Horseneck	8000000	Smith	L-11	900
Downtown	Brooklyn	9000000	Williams	L-17	1000
Perryridge	Horseneck	1700000	Adams	L-16	1300

分析-存在的问题

□ 存在的问题

- 数据冗余
- 对属性“Assets”的修改，必须对所有“Branch_name”的“Assets”属性也必须做相应的修改，否则将会导致数据的不一致。如两个Downtown的“Assets”值必须保持一致。
- 其他还有会出现数据更新异常。如以“Loan_number”为主码，当增加一个新的支行，而该支行还没有贷款用户，这在现实中是存在的情况，但因没有主码则无法正常插入。
- 一些领导喜欢一览表，但一览表不适合RDB，冗余、异常

□ 如何进行优化处理？如何对属性进行分解？

对关系Lending分解

表branch_customer

Branch_name	Branch_city	Assets	Customer_name
Downtown	Brooklyn	9000000	Jones
Redwood	Palo Alto	2100000	Smith
Perryridge	Horseneck	1700000	Hayes
Downtown	Brooklyn	9000000	Jackson
Mianus	Houseneck	400000	Curry
Round hill	Horseneck	8000000	Smith
Downtown	Brooklyn	9000000	Williams
Perryridge	Horseneck	1700000	Adams

表customer_loan

Customer_name	Loan_number	Amount
Jones	L-17	1000
Smith	L-23	2000
Hayes	L-15	1500
Jackson	L-14	1500
Curry	L-93	500
Smith	L-11	900
Williams	L-17	1000
Adams	L-16	1300

Branch_customer ⋈ customer_loan

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
Round hill	Horseneck	8000000	Smith	L-23	2000
Perryridge	Horseneck	1700000	Hayes	L-15	1500
Downtown	Brooklyn	9000000	Jackson	L-14	1500
Mianus	Houseneck	400000	Curry	L-93	500
Redwood	Palo Alto	2100000	Smith	L-11	900
Round hill	Horseneck	8000000	Smith	L-11	900
Downtown	Brooklyn	9000000	Williams	L-17	1000
Perryridge	Horseneck	1700000	Adams	L-16	1300

查询：找出所有金额小于1000美元的贷款

对上表的查询结果：Mianus, Redwood, Round hill

对表lending的查询结果为：Mianus, Round hill

原因及解决办法

- ❑ 上述分解是错误的，显然，分解不当会导致错误的结果！
- ❑ 关系模式的更新异常问题是由属性间的不好的数据依赖引起的，数据依赖是指数据之间存在着某种内在的联系。
- ❑ 如，每个支行的资产的唯一的，即每个支行的名字唯一地确定了一个资产总额。这种现象称之为数据依赖，解决这些问题的关键是处理好数据依赖。

目录

- 问题导引
- 数据依赖
- Armstrong公理系统
- 模式分解
- 规范化方法
 - 1NF
 - 2NF
 - 3NF
 - BCNF
 - 4NF（自学）
- 数据库设计问题

关系模型设计理论-函数依赖

- 数据依赖主要包括
 - 函数依赖（单值依赖）
 - 多值依赖
- 函数依赖(Functional Dependency, FD)的概念。
设有关系 $r(R)$ ， R 的子集 X 、 Y 。如果对关系 r 的任何两个元组 t_1 、 t_2 ，只要 $t_1[X]=t_2[X]$ ，就有 $t_1[Y]=t_2[Y]$ ，则称在关系 r 上 X 函数决定 Y ，或 Y 函数依赖 X ，记为 $X \rightarrow Y$ 。
- 函数依赖又称为函数相关性，函数依赖是单值依赖，使一些元组（多值）不能出现在关系中，导致插入删除的异常现象。
- 函数依赖是关系模式级的，是关系模型设计理论的核心。

函数依赖

- 如何理解函数依赖？
 - X 、 Y 是属性集合
 - $X \rightarrow Y$ 有几个含义：
 - $t1[X]=t2[X]$ ，就有 $t1[Y]=t2[Y]$
 - $t1[X] \neq t2[X]$ ，可以 $t1[Y]=t2[Y]$ ，也可以 $t1[Y] \neq t2[Y]$
 - 但是， 绝不允许： $t1[X]=t2[X]$ ，而 $t1[Y] \neq t2[Y]$
 - 只要 $t1[X]=t2[X]$ ，就必须考察 $t1[Y]=t2[Y]$ 是否成立，对 $t1[X] \neq t2[X]$ 则不考察。
 - 函数依赖是指具有数学函数的特征，函数依赖是语义范畴的概念，因此，只能根据语义来确定函数依赖，说白了，就是一组属性的值决定另外 一组属性的值。
- 平凡的函数依赖（所有的关系全满足）和非平凡的函数依赖（重点考察）

函数依赖是语义上的一种约束

例子

A	B	C	D
a1	b1	c1	d1
a1	b2	c1	d2
a2	b2	c2	d2
a2	b3	c2	d3
a3	b3	c2	d4

Which is true?

$A \rightarrow B$

$A \rightarrow C$

$C \rightarrow A$

$A \rightarrow D$

$B \rightarrow D$

$AB \rightarrow D$

$A \rightarrow C$
$AB \rightarrow D$

注意：这是在一个确定关系上的依赖，我们关心的是模式上的依赖

如何确定函数依赖的去留？

- 有些函数依赖必须保证，有些则需要消除。
 - “一个助理只能在一个支行工作” 就是一个需要保证的函数依赖
 - “一个客户必须有且只有一笔贷款” 就是一个不好的函数依赖，该依赖导致没有贷款但有存款的用户无法插入到模式中。
- 函数依赖保持或消除主要考虑以下因素：
 - 应用需求：必须满足，这是第一原则。
 - 设计要求：存在更新异常则需要消除，存在冗余则根据情况确定，冗余严重则需要消除，不严重则可以保留。
 - 系统要求：主要考虑效率问题，保持函数依赖有时需要付出系统开销。

函数依赖集的闭包

- 给定关系模式 R 上的函数依赖集 F ，如果每一个满足 F 的关系实例 $r(R)$ 也满足函数依赖 f (如 $X \rightarrow Y$)，则称 F 逻辑蕴涵 f (蕴含==导出)
- 令 F 为一个函数依赖集，则 F 的闭包(**closure**)是指 F 逻辑蕴涵的所有函数依赖的集合，记为 F^+
- 如果能找到 F^+ 就能够得到一个模式上的所有函数依赖。
- 如何找到 F^+ ?

目录

- 问题导引
- 数据依赖
- **Armstrong**公理系统
- 模式分解
- 规范化方法
 - 1NF
 - 2NF
 - 3NF
 - BCNF
 - 4NF（自学）
- 数据库设计问题

Armstrong公理系统

□ Armstrong公理(1974):

- 自反律: 若 X 为一属性集且 $Y \subseteq X$, 则 $X \rightarrow Y$
- 增补律: 若有 $X \rightarrow Y$ 且 W 为一属性集, 则 $WX \rightarrow WY$
- 传递律: 若有 $X \rightarrow Y$ 及 $Y \rightarrow W$, 则有 $X \rightarrow W$

□ 三个推理

- 合并律: 若 $X \rightarrow Y$, $X \rightarrow Z$ 成立, 则 $X \rightarrow YZ$ 成立
- 分解律: 若 $X \rightarrow YZ$ 成立, 则 $X \rightarrow Y$ 及 $X \rightarrow Z$ 成立
- 伪传递律: 若 $X \rightarrow Y$ 及 $WY \rightarrow Z$ 成立, 则 $WX \rightarrow Z$ 成立

□ 保真性（推出来的都正确），完备性（正确的都能推出来），因此给定一个 F ，可以产生完整的 F^+ 。

例子：Armstrong公理

- 给定模式 $R=(A, B, C, G, H, I)$ 及函数依赖集 $F=\{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$ ，使用上述规则可以得到 F^+ ：
 - 传递律：由 $A \rightarrow B, B \rightarrow H$ 得 $A \rightarrow H$
 - 合并律：由 $CG \rightarrow H, CG \rightarrow I$ 得 $CG \rightarrow HI$
 - 伪传递律：由 $A \rightarrow C, CG \rightarrow H$ 得 $AG \rightarrow H$ ，同理得 $AG \rightarrow I$
- 除了最后一次外，每次执行算法 **repeat** 循环都至少往 F^+ 中加入一个函数依赖，直至不变化为止。

理解：闭包 F^+ 求解思想

- ❑ 思想：亲戚的闭包，亲戚的亲戚还是亲戚，滚雪球，靠什么滚？Armstrong公理系统
- ❑ A是一个中国人， $(A\text{的亲戚})^+ = \text{中华民族}$
- ❑ 比喻：滚雪球，亲戚是闭包，找亲戚的亲戚（也是亲戚）
- ❑ 用法：ERWin（E-R模型设计工具）等自动设计关系模式的软件快而省

目录

- 问题导引
- 数据依赖
- Armstrong公理系统
- 模式分解
- 规范化方法
 - 1NF
 - 2NF
 - 3NF
 - BCNF
 - 4NF（自学）
- 数据库设计问题

模式分解

- ❑ 模式分解是指用关系模式**R**的一组子模式等价地替换**R**，使子模式的结构比较简单。
- ❑ 模式分解是一种重要的关系规范方法,可以达到规范的目的。
- ❑ 需要注意的是，分解不当可能会导致另一种不好的设计。
- ❑ 例如：将**lending-schema**分解为两个模式（上述的例子）

**Branch-customer-schema=(branch_name, branch_city,
assets, customer_name)**

**Customer-loan-schema=(customer_name,
loan_number,amount)**

无损分解与有损分解

- 令 R 为一个关系模式， F 为 R 上的函数依赖集，令 R_1 和 R_2 是 R 的分解，如果 $r(R)$ 是模式 R 上的一个关系，若总是存在

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

- 即，分解后的两个模式通过连接后与原关系等价，没有增加元组，我们称该分解为无损连接分解。如果增加了元组，则增加了不确定性（如上述分解中损失了**Redwood**不合格这一信息），则成为有损连接分解。

如何实现无损分解？

- 令 R 为一关系模式， F 为 R 上的函数依赖集。令 R_1 和 R_2 为 R 的分解，该分解为 R 的无损连接分解的条件是， F^+ 中至少存在如下函数依赖中的一个：

$$R_1 \cap R_2 \rightarrow R_1$$

$$R_1 \cap R_2 \rightarrow R_2$$

或者说 $R_1 \cap R_2$ 是 R_1 或 R_2 的超码

- 例子：将lending-schema分解为两个模式：

Branch-schema=(branch_name,branch_city,assets)

Loan-info-schema=(branch_name, customer_name,
loan_number, amount)

由于 $\text{branch_name} \rightarrow \text{branch_name}, \text{branch_city}, \text{assets}$ ，因此该分解为无损分解

R_1

R_2

$R_1 \cap R_2$

R_1

□ 将Loan-info-schema分解为:

Loan-schema=(loan_number, branch_name, amount)

Borrower-schema=(customer_name, loan_number)

由于 $\text{Loan-schema} \cap \text{Borrower-schema} = \{\text{loan_number}\}$,
且 $\text{loan_number} \rightarrow \text{loan_number, branch_name, amount}$,
所以该分解为无损分解

□ 最后得到的如下3个关系:

Branch-schema=(branch_name, branch_city, assets)

Loan-schema=(loan_number, branch_name, amount)

Borrower-schema=(customer_name, loan_number)

该三个模式正是教材给定的设计结果（教材的例子）。

表loan

Loan_number	Amount	Branch_name
L-11	900	Round Hill
L-14	1500	Downtown
L-15	1500	Perryridge
L-16	1300	Perryridge
L-17	1000	Downtown
L-23	2000	Redwood
L-93	500	Mianus

表borrower

Customer_name	Loan_number
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17

表branch

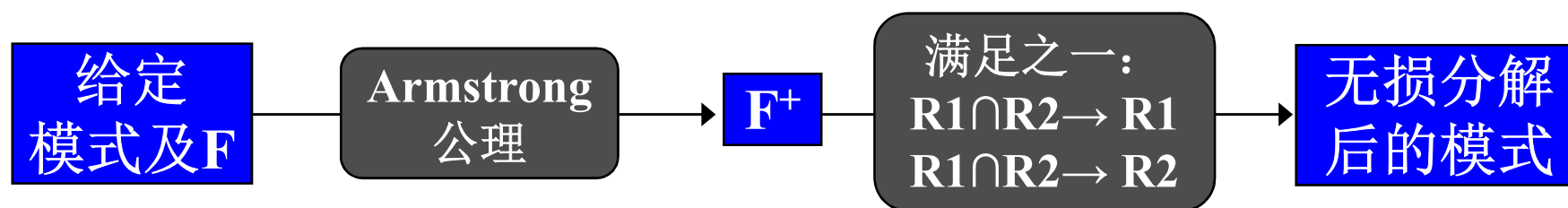
Branch_name	Branch_city	Assets
Brighton	Brooklyn	7100000
Downtown	Brooklyn	9000000
Mianus	Houseneck	400000
North Town	Rys	3700000
Perryridge	Horseneck	1700000
Pownal	Bennington	300000
Redwood	Palo Alto	2100000
Round hill	Horseneck	8000000

分解应具有的特性

- 对一个模式的分解是多种多样的,但是分解后产生的新模式应与原模式等价,因此模式分解应具有如下特性:
 - 无损连接分解（无损）
保证分解后信息不丢失
 - 保持函数依赖（保依）
保证分解后函数依赖不丢失,分解后的模式的函数依赖闭包的并与分解前的模式的函数依赖闭包相等
 - 消除信息重复（消冗）
分解后的模式不存在信息冗余的问题或冗余减少

Armstrong公理及无损分解

□ 实现无损分解的路线总结



□ 按这种方式分解太麻烦，不适合工程应用，工程中应该有更简单的方法，工程化的方法

如何按照一定的规则
进行模式分解？

目录

- 问题导引
- 数据依赖
- Armstrong公理系统
- 模式分解
- 规范化方法
 - 1NF
 - 2NF
 - 3NF
 - BCNF
 - 4NF（自学）
- 数据库设计问题

规范化引论

- ❑ 关系模式设计方案可能存在多个，多个关系模式设计方案有好坏之分，因此，关系模式设计是使得所设计的方案是好的或较好的。
- ❑ 要设计一个好的关系模式方案，关键是要摸清属性间的内在语义联系，即数据依赖。
- ❑ 在一个好的关系数据库模式设计方案中，每个关系的属性一定满足某种内在语义条件，也就是说要按一定的规范构造关系，这就是关系的规范化。
- ❑ 规范化可根据不同的要求而分成若干个级别。因此，一个关系数据库，它的每个关系一定得按规范化要求构造，这样才能得出一个好的数据库。

关于范式及规范化

- ❑ 范式本来用于表示关系得某一个级别，所以经常称某一个关系**R**为第几范式。
- ❑ 现在把范式概念理解为符合某一级别得关系模式集合，则**R**为第几范式可以写为 **$R \in xNF$** 。
- ❑ 各范式之间的关系有
 $5NF \subset 4NF \subset BCNF \subset 3NF \subset 2NF \subset 1NF$
- ❑ 规范化：一个低一级范式的关系模式，通过模式分解可以转换为若干个高一级的范式的关系模式的集合，这种过程称之为规范化。

目录

- 问题导引
- 数据依赖
- Armstrong公理系统
- 模式分解
- 规范化方法
 - 1NF
 - 2NF
 - 3NF
 - BCNF
 - 4NF（自学）
- 数据库设计问题

1NF

- 如果某个域的元素被认为是不可分解的单元，那么这个域就是原子的(**atomic**)。如果一个关系模式**R**的所有属性的域都是原子的，则称关系模式**R**属于**1NF**。
- 每一个分量都是不可分的，这是最基本的规范化，符合这个标准的关系模式称为符合**1NF**。
- 显然，第一范式没有用到函数依赖。
- 满足**1NF**是一个关系最基本的要求。

目录

- 问题导引
- 数据依赖
- Armstrong公理系统
- 模式分解
- 规范化方法
 - 1NF
 - 2NF
 - 3NF
 - BCNF
 - 4NF（自学）
- 数据库设计问题

2NF

- 定义：若 $R \in 1NF$ ，且每个非主属性完全函数依赖于码，则 $R \in 2NF$ 。(非主属性间可以有函数依赖，也可以存在传递函数依赖, 3NF解决)(主属性之间也可以存在依赖，BCNF解决)
- S-L-C(学号, 课程号, 所在系, 住所, 成绩)是一个不符合2NF的模式，约定：
 - 每个系的学生住在同一个楼，
 - 主码：(学号, 课程号)
 - 函数依赖：(学号, 课程号) \rightarrow 成绩，学号 \rightarrow 所在系，学号 \rightarrow 住所，所在系 \rightarrow 住所，(学号, 课程号) \rightarrow 所在系，(学号, 课程号) \rightarrow 住所
- 问题：冗余和更新异常

问题出在“所在系”、“住所”，它只是部分函数依赖于码(学号, 课程号)，即非主属性“所在系”、“住所”不完全函数依赖码(学号, 课程号)，因此不符合2NF

对码部分依赖

解决方法

- 解决办法是采用模式分解，将S-L-C分解成两个关系模式：

S-C(学号,课程号,成绩)

S-L(学号,所在系,住所)

- 模式分解后，两个关系都消除了非主属性部分依赖码的问题，S-C和S-L均属于2NF。
- 2NF只有历史意义，一般不做深入探讨。

目录

- 问题导引
- 数据依赖
- Armstrong公理系统
- 模式分解
- 规范化方法
 - 1NF
 - 2NF
 - 3NF
 - BCNF
 - 4NF（自学）
- 数据库设计问题

3NF

- 关系模式 $R(U, F)$ 中若不存在这样的码 X ，属性组 Y 及非主属性 $Z(Z \subseteq Y)$ ，使得 $X \rightarrow Y (Y \nrightarrow X)$ ， $Y \rightarrow Z$ 成立，则称 $R \in 3NF$ (所有非主属性不存在对码的部分依赖(2NF)，不存在传递函数依赖)。
- 即若一个关系是2NF的且不存在非主属性间的函数依赖（无传递依赖），则说它是第三范式的，记为3NF。
- （主属性间的依赖没有消除，BCNF解决）

例子：3NF

- 考察: S-C(学号,课程号,成绩), S-L(学号,所在系,住所)
 - S-C没有传递依赖, 因此 $S-C \in 3NF$ 。
 - 对于S-L, 由于学号 \rightarrow 所在系, 所在系 \rightarrow 住所, 则学号 \rightarrow 住所, 存在传递依赖, 所以, $S-L \notin 3NF$ 。当一个新生的学号没有给定, 即使报到了, 所在系及住所确定了也无法插入。现实中, 经常会有学生放弃就学机会的事情, 因此, 一般是学生报到后才会分配学号。
- 解决办法仍然是进行模式分解, 分解为:
 - S-D(学号, 所在系)
 - D-L(所在系, 住所)
 - 不存在非主属性之间的函数依赖, 即不存在传递依赖。分解后保证了只要一个新生报到, 即使没有学号也会有他的信息在关系D-L中。

函数依赖情况下，
3NF消除了传递函数依赖，
进一步消除了删除和插入的异常

目录

- 问题导引
- 数据依赖
- **Armstrong**公理系统
- 模式分解
- 规范化方法
 - 1NF
 - 2NF
 - 3NF
 - **BCNF**
 - 4NF（自学）
- 数据库设计问题

BCNF

- **BCNF (Boyce Codd Normal Form)**是由 **Boyce**和 **Codd**提出的，它比**3NF**又进一步，通常认为**BCNF**是修正的第三范式，有时也称为**3NF**。
- 定义：关系模式 $R(U, F) \in 1NF$ ，若 $X \rightarrow Y$ 且 $Y \not\subseteq X$ 时， X 必含有码，则 $R \in BCNF$ 。即， R 中每一个决定因素都包含码，则 $R \in BCNF$ 。
 - 所有非主属性对每一个码都是完全函数依赖。(2NF)
 - 所有主属性对每一个不包含它的码，也是完全函数依赖。
(主属性间不存在部分依赖)
 - 没有任何属性完全函数依赖于非码的任何一组属性。
(3NF)
- $R \in BCNF$ 消除了任何属性(包括主属性)对码的传递依赖与部分依赖，所以 $R \in 3NF$ 。

例子

□ 考察下列关系

G-R=(学号, 课程号, 成绩, 教师)设一个教师只承担一门课程, 一门课可由多个教师承担

候选码: (学号, 课程号), (学号, 教师)

由于不存在传递函数依赖, 所以**G-R** \in 3NF。由于教师 \rightarrow 课程号, 教师不是码, 主属性间存在部分函数依赖, 所以**G-R** \notin BCNF

学号	课程号	成绩	教师
0403170	CS210	92	李白
0409251	CS210	85	李白
0409251	IS350	78	杜甫
0516114	IS350	96	杜牧
0516114	IS460	67	白居易
...

□ 解决办法是将**G-R**进一步分解为**S-G**=(学号, 课程号, 成绩)和**T-C**=(教师, 课程号)

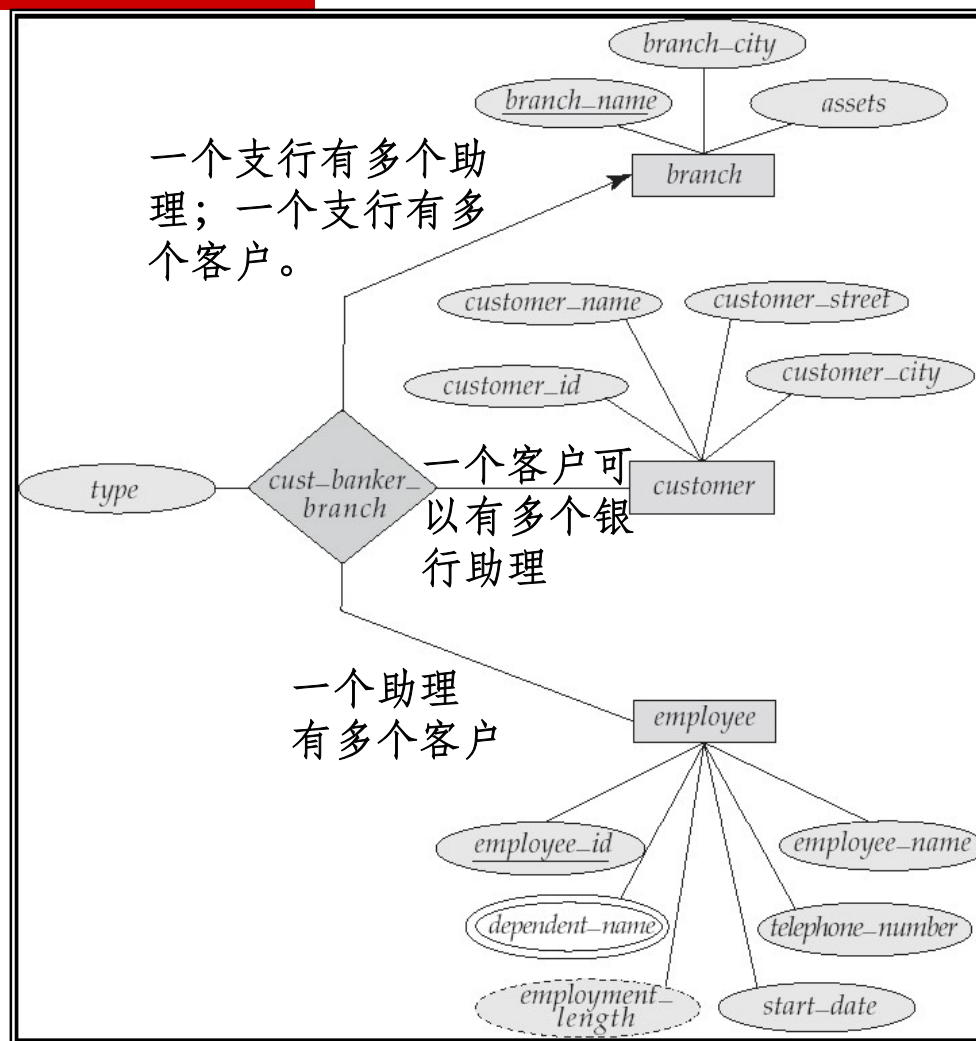
函数依赖范畴内，
BCNF消除了主属性对
码的部分函数依赖，
消除了插入删除异常的问题

3NF与BCNF的比较

- 我们对具有函数依赖的数据库设计的目标是：
 - 1.无损
 - 2.无冗, **BCNF**或**3NF**
 - 3.保依, **3NF**
- **3NF**的一个优点是满足无损并保持依赖(无损+保依), 缺点是有时不得不用空值表示数据项间的某些可能有意义的联系, 而且还存在信息冗余。
- **BCNF**没有做到保持依赖, 检查**FD**的效率低下, 要求过于严格, 工程上不适用
- **3NF or BCNF, which one?**

例子：BCNF不能保依

- 一个客户可以有多个银行助理；一个助理有多个客户；一个支行有多个助理；一个支行有多个客户。
- 设计的E-R图中用三元联系 **cust_banker_branch** 描述上述需求。




❑ **cust_banker_branch**的模式（逻辑设计）为：

cust_banker_branch = (customer_id, employee_id, branch_name, type)
employee_id → *branch_name* （主属性间存在依赖）

- ❑ 分析：由于不存在传递函数依赖，因此 **cust_banker_branch** ∈ 3NF（注：branch_name为主属性（branch_name为关系 branch 的码，customer_id, employee_id, branch_name为cust_banker_branch的超码(参ER模型部分)，而根据语义customer_id, branch_name应该为候选码，见下页），因此，存在主属性间的部分依赖问题，但是不存在非主属性对码的部分依赖，所以cust_banker_branch ∈ 2NF是成立的）
- ❑ 由于存在主属性对码的部分依赖（上述函数依赖），因此**cust_banker_branch** ∉ BCNF

- 上述两个约束（主码、函数依赖）不能保证“一个客户在一个支行最多有一个助理”的要求（假设存在这样的需求）。如下表

Customer_id	Employee_id	Branch_name	type
01	HB001	海淀	普通支行
01	ZB003	中关村	对公
01	ZB004	中关村	对公
02	HB002	海淀	普通支行
02	BB003	保福寺	对私
03	HB001	海淀	普通支行



- 但是，在该模式上很容易增加一个约束（如下）来实现上述需求
 $customer_id, branch_name \rightarrow employee_id$ (使 $customer_id, branch_name$ 为候选码)
这是一个非常重要的约束：对于同一个支行，只要 $customer_id$ 相同（一个客户），则 $employee_id$ 一定相同（只有一个雇员）。即一个客户在一个支行只有一个雇员。

-
- ❑ 如果将模式**cust_banker_branch**分解为以下两个模式
 $(customer_id, employee_id, type)$
 $(\underline{employee_id}, branch_name)$
 - ❑ 则上述两个模式满足**BCNF**，但是无法实现“一个客户在一个支行最多有一个助理”的要求。即
 $customer_id, branch_name \rightarrow employee_id$ 丢失了。
 - ❑ 可见，**BCNF**没有保持函数依赖
 $customer_id, branch_name \rightarrow employee_id$
 - ❑ 而满足**3NF**的模式**cust_banker_branch**却保持了这一函数依赖，实现了上述现实需求。
 - ❑ 因此，“少用属性多用表”是感性认识，**BCNF**会导致“分家后，原来的规矩不成立了”。分解不能过度，分解过度会导致一些属性间的约束无法保证。

选择3NF

- ❑ 在函数依赖范畴内，**BCNF**已经实现了彻底的分离，是最高范式，彻底消除了插入删除异常的问题。
- ❑ 一般设计希望满足**BCNF**，但**BCNF**过于严格，工程上一般难以实现。
- ❑ 另外，由于**BCNF**不保依，会导致一些约束难以实现，如上述例子。
- ❑ 所以一般情况下，降低一点要求（允许存在对主属性的部分依赖，允许存在一定的冗余），选用**3NF**。
- ❑ 但是由于存在对码的部分依赖，**3NF**还存在一定程度的插入删除异常问题，实际应用中可以通过对码的限制加以克服。

目录

- 问题导引
- 数据依赖
- **Armstrong**公理系统
- 模式分解
- 规范化方法
 - 1NF
 - 2NF
 - 3NF
 - BCNF
 - **4NF**（自学）
- 数据库设计问题

高级别范式

- 有些关系模式虽然属于**BCNF**，但从某种意义上说仍存在信息重复问题，所以没有被充分规范化
- 解决信息冗余的问题一般采用更高级别的范式，如**4NF**。
- **4NF**主要解决的是多值依赖问题，**BC**范式前解决的都是函数依赖（单值依赖）问题。

多值依赖

- 定义：设 $R(U)$ 是属性集 U 上的一个关系模式， X, Y, Z 是 U 的子集，并且 $Z=U-X-Y$ ，多值依赖 $X \twoheadrightarrow Y$ 成立，当且仅当对 $R(U)$ 的任一关系 R ，给定的一组 (X, Z) 值，有一组 Y 的值，这组 Y 的值仅取决于 X 值，而与 Z 值无关。
- 也就是说，对于 $R(U)$, X, Y 是 U 的子集, $Z=U-X-Y$, 对 R 任一关系 r ，如果 r 中存在元组 $(x1, y1, z1)$ 和 $(x1, y2, z2)$ 时，就也存在元组 $(x1, y2, z1)$ 和 $(x1, y1, z2)$ ，则称 Y 多值依赖于 X
- 若 $X \twoheadrightarrow Y$ ，而 $Z=\emptyset$ （即 Z 为空），则称 $X \twoheadrightarrow Y$ 为平凡的多值依赖
- 对比多值属性，注意多值依赖也是语义上的

MVD（多值依赖）

□ Tabular representation of $\alpha \twoheadrightarrow \beta$

	战士	枪	刀
	α	β	$R - \alpha - \beta$
t_1	a_1 张飞 a_i	a_i 蛇矛枪 a_j	a_j 剪刀 a_n
t_2	a_1 张飞 a_i	b_i 红缨枪 b_j	b_j 菜刀 b_n
t_3	a_1 张飞 a_i	a_i 蛇矛枪 a_j	b_j 菜刀 b_n
t_4	a_1 张飞 a_i	b_i 红缨枪 b_j	a_j 剪刀 a_n

□ 战士 \twoheadrightarrow 枪，张飞有两杆枪，两把刀，有4种组合

理解多值依赖

- 函数依赖规定了某些元组不能出现在关系中。如果 $A \rightarrow B$ 成立，就不能有两个元组在 A 上的值相同而在 B 上的值不同。所以函数依赖是单值依赖。
- 多值依赖则不是排除某些元组的存在，而是要求某种形式的其他元组必须出现在关系中。即两个元组在 A 上的值相同而在 B 上的值可以不同，但必须成组出现，只要 A 上的值出现，则对应的 B 上的值所有值都必须出现。
- 由于这个原因，函数依赖有时被称为相等产生依赖，而多值依赖被称为元组产生依赖。

例子：多值依赖

- 多值依赖可以用衣裤搭配的例子说明

人(α)	衣(β)	裤($R-\beta$)
李白	毛衣	牛仔
李白	T恤	休闲
李白	毛衣	休闲
李白	T恤	牛仔

说明：
衣服（毛衣、T恤衫）
裤子（牛仔、休闲裤）
可以相互搭配

- 若插入(‘李白’，‘毛衣’)，(‘李白’，‘T恤’)则必须插入上述4个元组，删除(‘李白’，‘毛衣’，‘休闲’)则必同时删除1、2元组之一。
- 人(α) \twoheadrightarrow 衣(β)， 人(α) \twoheadrightarrow 裤($R-\beta$)， 二者等价

多值依赖的应用

- 与函数依赖相同，可以用两种方式使用多值依赖
 - 验证关系，确定在给定的函数依赖集或多值依赖集下是否合法
 - 指定合法关系集上的约束
- 由多值依赖的定义，可以得出以下规则
 - 若 $X \rightarrow Y$ ，则 $X \twoheadrightarrow Y$
 - 即，每一个函数依赖都是多值依赖，函数依赖是单值依赖，是多值依赖的一个特例。

4NF定义

- 关系模式 $R(U, F) \in 1NF$ ，对于存在的每一个非平凡的多值依赖 $X \twoheadrightarrow Y (Y \not\subseteq X)$ ，都满足 X 必含有码这一要求，则称 $R \in 4NF$ 。（对于存在的平凡的多值依赖没有要求）（消除非平凡且非函数依赖的多值依赖）
- 注意：4NF的定义与BCNF定义的唯一不同是用多值依赖替代了函数依赖。
- $4NF \in BCNF$ ，因为如果模式 R 不属于BCNF，则 R 上存在非平凡的函数依赖 $X \rightarrow Y$ 且 X 不是超码，由于 $X \rightarrow Y$ 蕴涵 $X \twoheadrightarrow Y$ ，所以 R 不属于4NF

例子：4NF

- 考虑前面的衣裤例子，由于模式 R （人(α), 衣(β), 裤($R-\beta$)) 没有传递函数依赖，也没有主属性对码的部分依赖（本例为全码），因此该模式 $R \in BCNF$ 。
- 但是，由于 R 中存在人(α) $\rightarrow\rightarrow$ 衣(β), 人(α) $\rightarrow\rightarrow$ 裤($R-\beta$), 且为“人(α)”不是码，所以存在非平凡的多值依赖，所以 $R \notin 4NF$
- 进一步分解，分解的一个技巧是将关系分解为仅具有平凡的多值依赖（ $R-\beta$ 为空）的关系。

R分解后满足4NF

- 将R分解为:
 - R1 (人(α), 衣(β))
 - R2 (人(α), 裤($R-\beta$))
- 这样, 人(α) $\rightarrow\rightarrow$ 衣(β) , 人(α) $\rightarrow\rightarrow$ 裤($R-\beta$) 分别在R1和R2中为平凡的多值依赖, 不存在非平凡的多值依赖 (注意: R1、R2均为全码), 所以R1 \in 4NF, R2 \in 4NF

R1

人(α)	衣(β)
李白	毛衣
李白	T恤

R2

人(α)	裤($R-\beta$)
李白	牛仔
李白	休闲

仅有平凡的多值依赖

4NF消除非函数依赖的多值依赖

更多范式

- ❑ 第四范式不是最终的范式
- ❑ 还有一些类型的概括多值依赖的约束称为连接依赖，引出另一种范式称为投影-连接范式，又称**5NF**
- ❑ 还有一类更一般的约束，引出另一种范式称为域-码范式
- ❑ 存在的问题：难以推导，没有形成一套具有保真性和完备性的推理规则用于约束的推导
- ❑ 很少应用

目录

- 问题导引
- 数据依赖
- **Armstrong**公理系统
- 模式分解
- 规范化方法
 - 1NF
 - 2NF
 - 3NF
 - BCNF
 - 4NF（自学）
- 数据库设计问题

数据库设计问题

- 规范化是如何糅合在数据库设计中的？
- 规范化是假定一个关系模式**R**已经存在，那么，通过什么方式得到关系模式**R**呢？
 - 由**ER**图转换而来，需要进行范化
 - 由规范化得来的，但可能需要进一步规范化
 - 即席设计的结果，验证是否满足期望的范式

ER模型与规范化

- ❑ 好的ER模型设计可以直接转化为满足3NF的关系模式，高手设计的ER模型。
- ❑ 一般的ER，设计模式后，需要分解，才能达到3NF。
- ❑ 规范化既可以留给数据库设计者在ER建模时靠直觉实现，也可以从ER模型生成的关系上形式化地进行，二者可任选其一。

反规范化

- ❑ 为了性能的提高，有时会牺牲规范化。
- ❑ 规范化不能过分，适可而止，分得太小，查询时连接太多
- ❑ 100个字段的一览表不好，但分成50个小关系，每个关系两关字段的效率低，而且还可能丢失一些数据依赖。
- ❑ 必要时有一点传递依赖，部分依赖，具体情况下的冗余不太多，也可允许。存取可快一点。

小结

- ❑ 数据依赖是关系模型设计理论的核心, 数据依赖分为多值依赖和函数依赖, 函数依赖是基础
- ❑ $5NF \subset 4NF \subset BCNF \subset 3NF \subset 2NF \subset 1NF$
- ❑ 一个低一级范式的关系模式向高一级范式转换是通过模式分解完成的, 模式分解的依据是数据依赖
- ❑ 在函数依赖范畴, **BCNF**解决了更新异常问题, 但要求过于严格, 折中处理, 一个数据库设计至少应该满足**3NF**, **3NF**一定程度上解决了数据的更新异常问题, 以后的范式更多的是减少了数据冗余。更新异常是不可接受的, 而数据冗余在多数情况下则是可以接受的。
- ❑ 不能说规范化程度越高的关系模式就越好, 上面的规范化步骤可以在其中任何一步终止

小结-范式转换

