

# 数据库系统概论

## An Introduction to Database Systems

---

### 关系数据库规范化理论

The Relational Database Normalization Theory

陆嘉恒

中国人民大学信息学院

[www.jiahenglu.net](http://www.jiahenglu.net)

# 主要学术经历

---

- **2006.9 ~2008.6 美国 University of California, Irvine 博士后研究员 导师：李晨教授**
- **2002.8 ~2006.8 新加坡国立大学 (National University of Singapore) 攻读博士学位 导师：Ling Tok Wang教授**
- **1998.9 ~ 2001.1 上海交通大学硕士学位**

# 承担三次课的教学

---

- 规范化理论（三次）
- 期中考试

# 下个学期双语课

---

- 计算机科学选讲
- **Introduction to advanced database topics**
- 选修课 2学分
- 欢迎选修

# 下个学期双语课

---

- 计算机科学选讲
- 信息检索与搜索引擎技术
- XML数据管理技术
- 关键词模糊搜索技术

# Welcome questions

---

- Q & A

- 下载课程slides

[www.jiahenglu.net](http://www.jiahenglu.net) 中文版

# 概念回顾

---

- **关系：**描述实体及其属性、实体间的联系。  
从形式上看，它是一张二维表，是所涉及属性的笛卡尔积的一个子集。
- **关系模式：**用来定义关系。
- **关系数据库：**基于关系模型的数据库，利用关系来描述现实世界。  
从形式上看，它由一组关系组成。
- **关系数据库的模式：**定义这组关系的关系模式的全体。

# 什么是数据依赖

---

## 数据依赖

- 是通过一个关系中属性间值的相等与否体现出来的数据间的相互关系
- 是现实世界属性间相互联系的抽象
- 是数据内在的性质
- 是语义的体现

# 什么是数据依赖（续）

---

## 数据依赖的类型

很多，其中最重要的是：

- 函数依赖（**Functional Dependency**，简记为**FD**）
- 多值依赖（**Multivalued Dependency**，简记为**MVD**）

# 函数依赖概念

---

- 一、函数依赖
- 二、平凡函数依赖与非平凡函数依赖
- 三、完全函数依赖与部分函数依赖
- 四、传递函数依赖
- 五、码

# 函数依赖

---

**定义5.1** 设 $R(U)$ 是一个关系模式， $U$ 是 $R$ 的属性集合， $X$ 和 $Y$ 是 $U$ 的子集。对于 $R(U)$ 的任意一个可能的关系 $r$ ，如果 $r$ 中不存在两个元组，它们在 $X$ 上的属性值相同，而在 $Y$ 上的属性值不同，则称“ $X$ 函数确定 $Y$ ”或“ $Y$ 函数依赖于 $X$ ”，记作 $X \rightarrow Y$ 。

即若  $t_1$  and  $t_2$  in  $r(R)$ , 则

$$t_1[X] = t_2[X] \longrightarrow t_1[Y] = t_2[Y].$$

# 函数依赖（续）

---

说明：

1. 函数依赖不是指关系模式**R**的某个或某些关系实例满足的约束条件，而是指**R**的所有关系实例均要满足的约束条件。
2. 函数依赖是语义范畴的概念。只能根据数据的语义来确定函数依赖。例如“姓名→年龄”这个函数依赖只有在没有同名人的条件下成立。如果有相同名字的人，则“年龄”就不再函数依赖于“姓名”了。

## 函数依赖（续）

---

3. 数据库设计者可以对现实世界作强制的规定。例如在上例中，设计者可以强行规定不允许同名的人出现，因而使函数依赖“姓名 $\rightarrow$ 年龄”成立。这样当插入某个元组时这个元组上的属性值必须满足规定的函数依赖，若发现有同名的人存在，则拒绝装入该元组。
4. 若 $X \rightarrow Y$ ，则 $X$ 称为这个函数依赖的决定属性集(Determinant)。
5. 若 $X \rightarrow Y$ ，并且 $Y \rightarrow X$ ，则记为 $X \leftrightarrow Y$ 。
6. 若 $Y$ 不函数依赖于 $X$ ，则记为 $X \not\rightarrow Y$ 。

# 平凡函数依赖与非平凡函数依赖

---

- 定义 在关系模式 $R(A_1, \dots, A_n)$ 中，对于 $A_1, \dots, A_n$ 的子集 $X$ 和 $Y$ ，如果 $X \rightarrow Y$ ，但 $Y \subseteq X$ ，则称 $X \rightarrow Y$ 是非平凡的函数依赖。若 $X \rightarrow Y$ ，但 $Y \subseteq X$ 则称 $X \rightarrow Y$ 是平凡的函数依赖。
  - 对于任一关系模式，平凡函数依赖都是必然成立的，它不反映新的语义，因此若不特别声明，我们总是讨论非平凡函数依赖。

# 完全函数依赖与部分函数依赖

---

定义 在关系模式 $R(A_1, \dots, A_n)$ 中，如果 $X \rightarrow Y$ ，并且对于 $X$ 的任何一个真子集 $X'$ ，都有 $X' \not\rightarrow Y$ ，则称 $Y$ 完全函数依赖于 $X$ ，记作 $X \xrightarrow{f} Y$ 。若 $X \rightarrow Y$ ，但 $Y$ 不完全函数依赖于 $X$ ，则称 $Y$ 部分函数依赖于 $X$ ，记作 $X \xrightarrow{p} Y$ 。

例: 在关系 $SC(Sno, Cno, Grade)$ 中，有:

$Sno \rightarrow Grade$ ,  $Cno \rightarrow Grade$ ,

$(Sno, Cno) \xrightarrow{f} Grade$ 。(Sno, Cno)是决定属性集。

# 传递函数依赖

---

定义 在关系模式 $R(A_1, \dots, A_n)$ 中，如果 $X \rightarrow Y$  ( $Y \not\subseteq X$ )， $Y \twoheadrightarrow X$ ， $Y \rightarrow Z$ ，则称 $Z$ 传递函数依赖于 $X$ 。

注：如果 $Y \rightarrow X$ ，即 $X \leftrightarrow Y$ ，则 $Z$ 直接依赖于 $X$ 。

例3：在关系 $Std(Sno, Sdept, Dname)$ 中，有：

$Sno \rightarrow Sdept$ ， $Sdept \rightarrow Dname$ ， $Dname$ 传递函数依赖于 $Sno$ 。

# 函数依赖： 举例

---

例：在关系**Student(Sno, Sname, Ssex, Sage, Sdept)**中，

假设无人重名，则有：

**Sno  $\rightarrow$  Ssex, Sno  $\rightarrow$  Sage, Sno  $\rightarrow$  Sdept,**  
**Sno  $\leftrightarrow$  Sname**

但**Ssex  $\not\rightarrow$  Sage**

# 函数依赖与码

---

- 超码 (Superkey)
  - An attribute or a set of attributes that uniquely identifies a tuple within a relation.
- 候选码 (Candidate Key)
  - A superkey (K) such that no proper subset is a superkey within the relation.
  - In each tuple of R, the values of K uniquely identify that tuple (uniqueness).
  - No proper subset of K has the uniqueness property (irreducibility).
  - 在最简单的情况下，候选码只包含一个属性。在最极端的情况下，关系模式的所有属性组是这个关系模式的候选码，称为全码 (All-key)。

# 第二篇 关系数据规范化理论

---

## ● 问题的提出

### — 数据库逻辑设计

- 关系数据库逻辑设计：针对一个具体问题，应如何构造一个适合于它的数据模式，即应该构造几个关系，每个关系由哪些属性组成等。
- 数据库逻辑设计的方法——关系数据库的规范化理论。

# 规范化

---

问题模式举例

第一范式 (1NF, 5.2.1)

第二范式 (2NF, 5.2.2)

第三范式 (3NF, 5.2.3)

BC范式 (BCNF, 5.2.4)

多值依赖与第四范式 (4NF, 5.2.5)

规范化小结

# 关系模式举例-“问题模式”（1）

---

例：建立一个描述学校的数据库。

涉及的对象包括：

学生的学号（**Sno**）

所在系（**Sdept**）

系主任姓名（**Dname**）

课程名（**Cname**）

成绩（**Grade**）

## 关系模式举例-“问题模式”(2)

---

假设学校的数据库模式由一个单一的关系模式**Student**构成，

则该关系模式的属性集合为：

$$U = \{ Sno, Sdept, Dname, Cname, Grade \}$$

# 关系模式举例-“问题模式”(3)

---

现实世界的已知事实告诉我们：

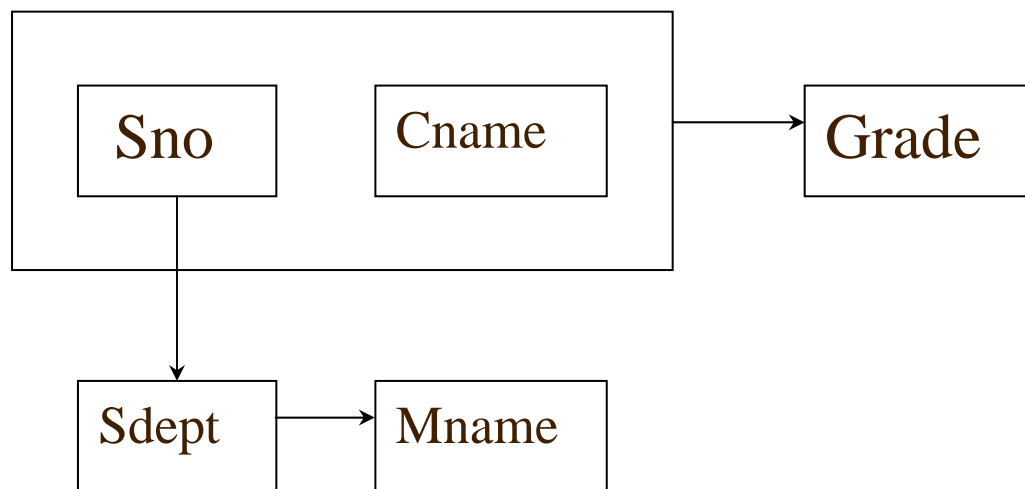
1. 一个系有若干学生， 但一个学生只属于一个系；
2. 一个系只有一名主任；
3. 一个学生可以选修多门课程， 每门课程有若干学生选修；
4. 每个学生所学的每门课程都有一个成绩。

# 关系模式举例-“问题模式”(4)

---

从上述事实我们可以得到属性组U上的一组函数依赖F:

$$F = \{ \text{Sno} \rightarrow \text{Sdept}, \text{Sdept} \rightarrow \text{Dname}, (\text{Sno}, \text{Cname}) \rightarrow \text{Grade} \}$$



# 关系模式举例-“问题模式”(5)

---

仅用一个表存在的问题:

1. 数据冗余太大: 浪费大量的存储空间。  
例, 每一个系主任的姓名重复出现, 重复次数与该系所有学生的所有课程成绩出现次数相同。

# 关系模式举例-“问题模式”(6)

---

## 2. 更新异常（Update Anomalies）：

数据冗余，更新数据时，维护数据完整性代价大。

例，某系更换系主任后，系统必须修改与该系学生有关的每一个元组。

## 3. 插入异常（Insertion Anomalies）：该插的数据插不进去。

例，如果一个系刚成立，尚无学生，我们就无法把这个系及其系主任的信息存入数据库。

# 关系模式举例-“问题模式”(7)

---

4. 删除异常（**Deletion Anomalies**）：不该删除的数据不得不删。

例，如果某个系的学生全部毕业了，我们在删除该系学生信息的同时，把这个系及其系主任的信息也丢掉了。

**结论：**单一表的关系模式不是一个好的模式。

一个“好”的模式应当不会发生插入异常、删除异常、更新异常，数据冗余应尽可能少。

# 关系模式举例-“问题模式”(8)

---

**原因：**由存在于模式中的某些函数依赖引起的。

**解决方法：**通过分解关系模式来消除其中不合适的数据依赖。

规范化理论正是用来改造关系模式，通过分解关系模式来消除其中不合适的数据依赖，以解决插入异常、删除异常、更新异常和数据冗余问题。

# 规范化

---

- 范式是符合某一种级别的关系模式的集合。
- 关系数据库中的关系必须满足一定的要求。满足不同程度要求的为不同范式。
- 范式的种类：
  - 第一范式(1NF)
  - 第二范式(2NF)
  - 第三范式(3NF)
  - BC范式(BCNF)
  - 第四范式(4NF)
  - 第五范式(5NF)

# 规范化（续）

---

- 各种范式之间存在联系：

$1NF \supset 2NF \supset 3NF \supset BCNF \supset 4NF \supset 5NF$

- 某一关系模式 **R** 为第 **n** 范式，可简记为 **R**  $\in$  **nNF**。

# 5.2 规范化

---

**5.2.1 第一范式 (1NF)**

**5.2.2 第二范式 (2NF)**

**5.2.3 第三范式 (3NF)**

**5.2.4 BC范式 (BCNF)**

**5.2.5 多值依赖与第四范式 (4NF)**

**5.2.6 规范化**

## 5.2.1 第一范式 (1NF)

---

- **1NF的定义**

定义5.7 如果一个关系模式**R**的所有属性都是不可分的基本数据项，则 **$R \in 1NF$** 。

- 第一范式是对关系模式的最起码的要求。不满足第一范式的数据库模式不能称为关系数据库。  
。
- 但是满足第一范式的关系模式并不一定是一个好的关系模式。

# 第一范式（续）

---

例: 关系模式 **SLC(Sno, Sdept, Sloc, Cno, Grade)**

**Sloc**为学生住处，假设每个系的学生住在同一个地方。

- 函数依赖包括:

$(\text{Sno}, \text{Cno}) \xrightarrow{f} \text{Grade}$

$\text{Sno} \rightarrow \text{Sdept}$

$(\text{Sno}, \text{Cno}) \xrightarrow{P} \text{Sdept}$

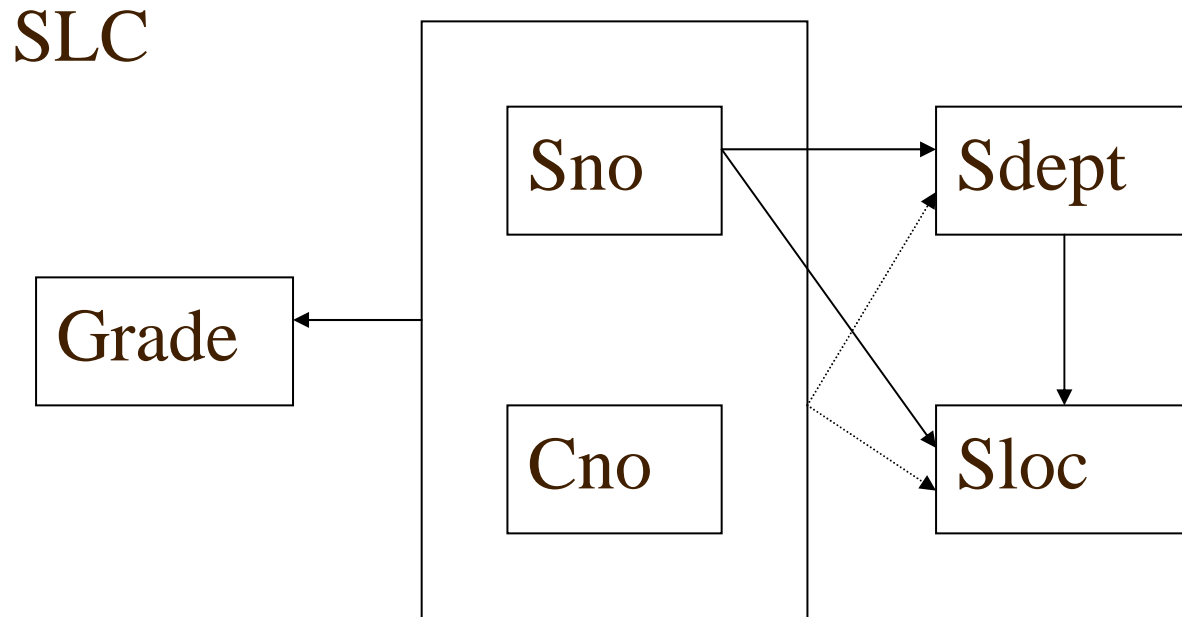
$\text{Sno} \rightarrow \text{Sloc}$

$(\text{Sno}, \text{Cno}) \xrightarrow{P} \text{Sloc}$

$\text{Sdept} \rightarrow \text{Sloc}$

# 第一范式 (续)

---



- SLC的码为(Sno, Cno)

# 第一范式（续）

---

- 结论:

1. SLC满足第一范式。
2. 非主属性Sdept和Sloc部分函数依赖于码(Sno, Cno)

- SLC存在的问题:

- (1) 插入异常

假设Sno=95102, Sdept=IS, Sloc=N的学生还未选课, 因课程号是主属性, 因此该学生的信息无法插入SLC。

# 第一范式（续）

---

## (2) 删除异常

假定某个学生本来只选修了3号课程这一门课。现在因身体不适，他连3号课程也不选修了。因课程号是主属性，此操作将导致该学生信息的整个元组都要删除。

## (3) 数据冗余度大

如果一个学生选修了10门课程，那么他的Sdept和Sloc值就要重复存储了10次。

# 第一范式（续）

---

## (4) 修改复杂

例如学生转系，在修改此学生元组的Sdept值的同时，还可能也需要修改住处（Sloc）。如果这个学生选修了K门课，则必须无遗漏地修改K个元组中全部Sdept、Sloc信息。

因此SLC不是一个好的关系模式。

# 第一范式（续）

---

- 原因

**Sdept**、**Sloc**部分函数依赖于码。

- 解决方法

采用投影分解法，把SLC分解为两个关系模式，以消除这些部分函数依赖。

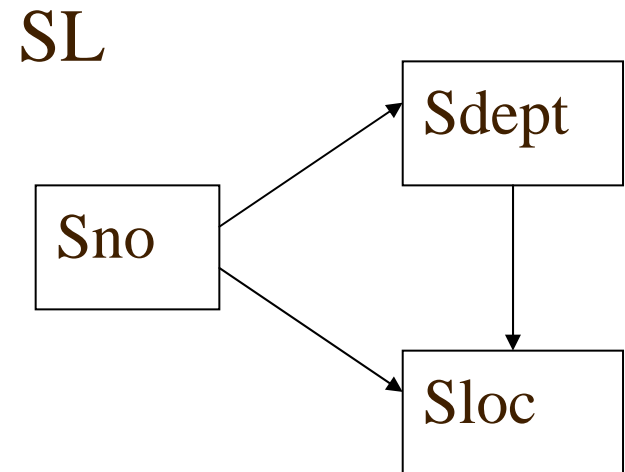
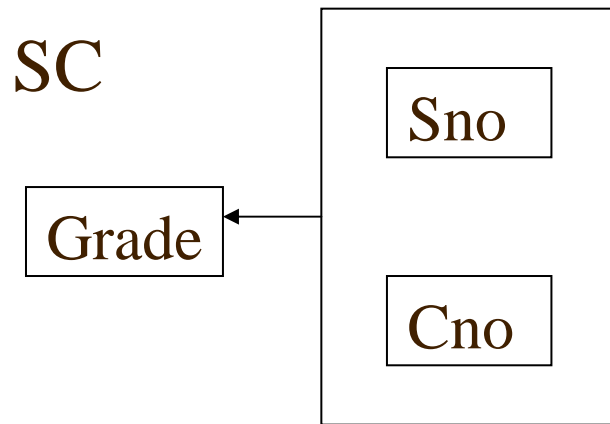
**SC (Sno, Cno, Grade)**

**SL (Sno, Sdept, Sloc)**

# 第一范式 (续)

---

函数依赖图:



# 第一范式（续）

---

在SC和SL中，非主属性都完全函数依赖于码了。从而使上述四个问题在一定程度上得到了一定的解决：

- (1) 由于学生选修课程的情况与学生的基本情况是分开存储在两个关系中的，在SL关系中可以插入尚未选课的学生。
- (2) 删除一个学生的所有选课记录，只是SC关系中没有关于该学生的记录了，SL关系中关于该学生的记录不受影响。

# 第一范式（续）

---

- (3) 不论一个学生选多少门课程，他的**Sdept**和**Sloc**值都只存储1次。这就大大降低了数据冗余
- (4) 学生转系只需修改SL关系中该学生元组的**Sdept**值和**Sloc**值，由于**Sdept**、**Sloc**并未重复存储，因此减化了修改操作。

## 5.2.2 第二范式 (2NF)

---

### ● 2NF的定义

**定义5.8** 若关系模式 $R \in 1NF$ ，并且每一个非主属性都完全函数依赖于 $R$ 的码，则 $R \in 2NF$ 。

例: **SLC(Sno, Sdept, Sloc, Cno, Grade)**

**SC (Sno, Cno, Grade)**

**SL (Sno, Sdept, Sloc)**

# 第二范式（续）

---

- 采用投影分解法将一个**1NF**的关系分解为多个**2NF**的关系，可以在一定程度上减轻原**1NF**关系中存在的插入异常、删除异常、数据冗余度大、修改复杂等问题。
- 将一个**1NF**关系分解为多个**2NF**的关系，并不能完全消除关系模式中的各种异常情况和数据冗余。

## 第二范式（续）

---

例：2NF关系模式SL(Sno, Sdept, Sloc)中

- 函数依赖：

$Sno \rightarrow Sdept$

$Sdept \rightarrow Sloc$

$Sno \rightarrow Sloc$

Sloc传递函数依赖于Sno，即SL中存在非主属性对码的传递函数依赖。

# 第二范式（续）

---

- SL关系存在的问题：

- (1) 插入异常

- 如果某个系因种种原因（例如刚刚成立），目前暂时没有在校学生，我们就无法把这个系的信息存入数据库。

- (2) 删除异常

- 如果某个系的学生全部毕业了，我们在删除该系学生信息的同时，把这个系的信息也丢掉了。

- 。

## 第二范式（续）

---

### (3) 数据冗余度大

每一个系的学生都住在同一个地方，关于系的住处的信息却重复出现，重复次数与该系学生人数相同。

### (4) 修改复杂

当学校调整学生住处时，由于关于每个系的住处信息是重复存储的，修改时必须同时更新该系所有学生的Sloc属性值。

所以SL仍不是一个好的关系模式。

# 第二范式（续）

---

- 原因

**Sloc**传递函数依赖于**Sno**

- 解决方法

采用投影分解法，把**SL**分解为两个关系模式，以消除传递函数依赖：

**SD (Sno, Sdept)**

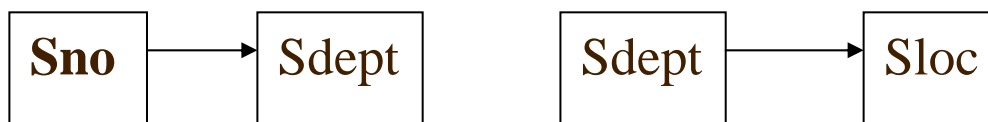
**DL (Sdept, Sloc)**

**SD**的码为**Sno**， **DL**的码为**Sdept**。

## 第二范式（续）

---

**SD**的码为**Sno**， **DL**的码为**Sdept**。



SD

DL

## 第二范式（续）

---

在分解后的关系模式中既没有非主属性对码的部分函数依赖也没有非主属性对码的传递函数依赖，在一定程度上解决了上述四个问题：

- (1) **DL**关系中可以插入无在校学生的系的信息。
- (2) 某个系的学生全部毕业了，只是删除**SD**关系中的相应元组，**DL**关系中关于该系的信息仍存在。
- (3) 关于系的住处的信息只在**DL**关系中存储一次。
- (4) 当学校调整某个系的学生住处时，只需修改**DL**关系中一个相应元组的**Sloc**属性值。

# 5.2 规范化

---

5.2.1 第一范式 (1NF)

5.2.2 第二范式 (2NF)

5.2.3 第三范式 (3NF)

5.2.4 BC范式 (BCNF)

5.2.5 多值依赖与第四范式 (4NF)

5.2.6 规范化

## 5.2.3 第三范式 (3NF)

---

- 3NF的定义

定义5.9 关系模式 $R\langle U, F \rangle$ 中若不存在这样的码 $X$ , 属性组 $Y$ 及非主属性 $Z$  ( $Z \notin Y$ ) 使得 $X \rightarrow Y$ ,  $Y \not\rightarrow X$ ,  $Y \rightarrow Z$ , 成立, 则称 $R\langle U, F \rangle \in 3NF$ 。

例,  $SL(Sno, Sdept, Sloc)$

$SD(Sno, Sdept)$

$DL(Sdept, Sloc)$

学生 (学号, 姓名, 宿舍楼, 宿舍号)

## 第三范式（续）

---

- 若 $R \in 3NF$ ，则 $R$ 的每一个非主属性既不部分函数依赖于候选码也不传递函数依赖于候选码。
- 如果 $R \in 3NF$ ，则 $R$ 也是 $2NF$ 。
- 采用投影分解法将一个 $2NF$ 的关系分解为多个 $3NF$ 的关系，可以在一定程度上解决原 $2NF$ 关系中存在的插入异常、删除异常、数据冗余度大、修改复杂等问题。
- 将一个 $2NF$ 关系分解为多个 $3NF$ 的关系后，并不能完全消除关系模式中的各种异常情况和数据冗余。

# 望岳

杜甫

岱宗夫如何，齐鲁青未了。  
造化钟神秀，阴阳割昏晓。  
荡胸生层云，决眦入归鸟。  
会当凌绝顶，一览众山小。



# Welcome questions

---

- Q & A

## 第三范式（续）

---

例：在关系模式STJ (S, T, J) 中，S表示学生，T表示教师，J表示课程。

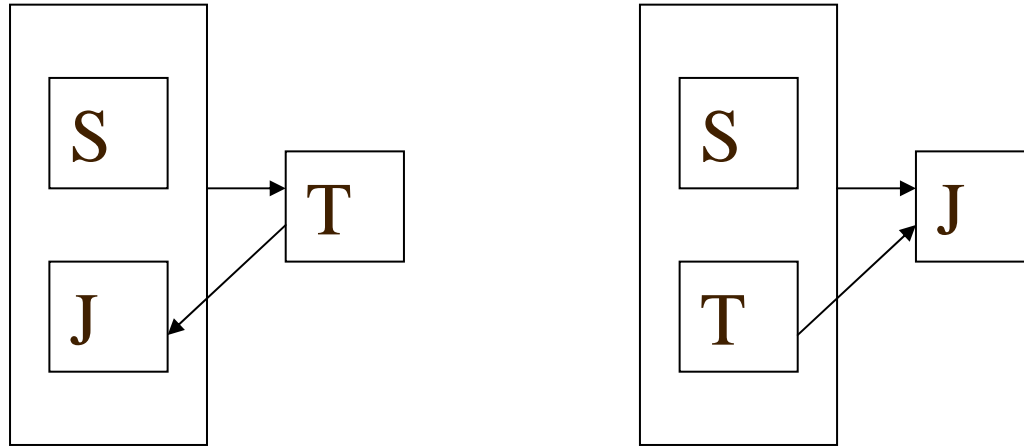
- 函数依赖：

假设每一教师只教一门课。每门课由若干教师教，某一学生选定某门课，就确定了一个固定的教师。于是有：

$$(S, J) \rightarrow T, (S, T) \rightarrow J, T \rightarrow J$$

# 第三范式 (续)

---



STJ

## 第三范式（续）

---

- $(S, J)$ 和 $(S, T)$ 都可以作为候选码。
- $STJ \in 3NF$
- $T \rightarrow J$ ，即 $T$ 是决定属性集，可是 $T$ 只是主属性，它既不是候选码，也不包含候选码。

## 第三范式（续）

---

### 数据冗余度大

虽然一个教师只教一门课，但每个选修该教师该门课程的学生元组都要记录这一信息。

### 修改复杂

某个教师开设的某门课程改名后，所有选修了该教师该门课程的学生元组都要进行相应修改。

因此虽然 $STJ \in 3NF$ ，但它仍不是一个理想的关系模式。

# 第三范式（续）

---

- 原因：

主属性**J**依赖于**T**，即主属性**J**部分依赖于码**(S, T)**。

- 解决方法：

采用投影分解法，将**STJ**分解为两个关系模式：

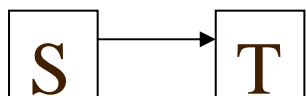
**ST (S, T)**

**TJ (T, J)**

# 第三范式（续）

---

ST的码为S， TJ的码为T。



ST



TJ

$(S, J) \rightarrow T$

$T \rightarrow J$

## 第三范式（续）

---

在分解后的关系模式中没有任何属性对码的部分函数依赖和传递函数依赖。它解决了上述问题：

关于每个教师开设课程的信息只在**TJ**关系中存储一次。某个教师开设的某门课程改名后，只需修改**TJ**关系中的一个相应元组即可。

但是，该分解不能保持函数依赖。

# 5.2 规范化

---

5.2.1 第一范式 (1NF)

5.2.2 第二范式 (2NF)

5.2.3 第三范式 (3NF)

5.2.4 BC范式 (BCNF)

5.2.5 多值依赖与第四范式 (4NF)

5.2.6 规范化

## 5.2.4 BCNF

---

- **BCNF (Boyce Codd Normal Form)** 是由 **Boyce**和**Codd**提出的，比**3NF**更进了一步。通常认为**BCNF**是修正的第三范式，所以有时也称为第三范式。
- **BCNF的定义**  
定义5.10 设关系模式 $R\langle U, F \rangle \in 1NF$ ，如果对于 $R$ 的每个函数依赖 $X \rightarrow Y$ ，若 $Y$ 不属于 $X$ ，则 $X$ 必含有候选码，那么 $R \in BCNF$ 。

# BCNF (续)

---

换句话说，在关系模式 $R\langle U, F \rangle$ 中，如果每一个决定属性集都包含候选码，则 $R \in BCNF$ 。

# BCNF (续)

---

- **BCNF**的关系，可以在进一步解决原**3NF**关系中存在的**数据冗余度大、修改复杂**等问题。
- **BCNF**的关系模式所具有的性质：
  1. 所有非主属性都完全函数依赖于每个候选码。
  2. 所有主属性都完全函数依赖于每个不包含它的候选码。
  3. 没有任何属性完全函数依赖于非码的任何一组属性。

# BCNF (续)

---

- **3NF与BCNF的关系**
  - 如果关系模式 $R \in \text{BCNF}$ ，必定有 $R \in \text{3NF}$ 。
  - 如果 $R \in \text{3NF}$ ，且 $R$ 只有一个候选码，则 $R$ 必属于 $\text{BCNF}$ 。
- 如果一个关系数据库中的所有关系模式都属于 $\text{BCNF}$ ，那么在函数依赖范畴内，它已实现了模式的彻底分解，达到了最高的规范化程度，消除了插入异常和删除异常。
- 我们无法得到保持函数依赖的 $\text{BCNF}$ 的分解
- 在 $\text{BCNF}$ 和 $\text{3NF}$ 之间要做权衡