关系规范化


PART 01
模式设计问题


PART 02
函数依赖


PART 03
Amstrong公理


PART 04
模式分解


PART 05
关系范式


PART 06
规范化原则及过程

1. 模式设计问题

- 如何构造一个合适的数据库模式？
  应该构造**几个关系模式**？
  每个关系模式应该由**哪些属性**组成？
- 数据库逻辑设计的工具
  关系数据库的**规范化理论**
- 思考
  你认为教务管理的数据库需要包含哪些关系模式？

- 关系模式的五元组表示： R(U, D, DOM, F)
  R：关系名称
  U：组成该关系的属性名称集合
  D：属性组U中属性所来自的域
  DOM：属性向域的映象集合
  F：属性间数据的依赖关系集合

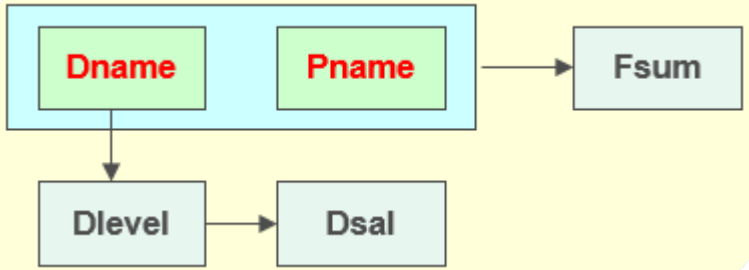
- 关系模式的简化三元组表示： **R（U, F）**

- 数据依赖举例
  医生姓名、医生职称、医生年龄与医生编号之间的关系？

Dname	Dlevel	Dsalary	Pname	Fsum
罗晓	主任医师	3200	张珍	¥30.00
杨勋	副主任医师	2800	张珍	¥50.00
杨勋	副主任医师	2800	刘景	¥55.00
杨勋	副主任医师	2800	张柳	¥58.00
邓英超	主治医师	2400	李秀	¥75.00
罗晓	主任医师	3200	傅伟相	¥35.00

其中包含以下函数依赖：

F = { {Dname, Pname}→Fsum,
Dname→Dlevel,
Dlevel→Dsal }



*假设医生和患者的姓名分别都是唯一的。

- 就诊模式R存在的问题（虽然只有5个属性）：

  数据冗余：浪费存储空间，引起异常。

  操作异常：

  更新异常（Update Anomalies）

  删除异常（Delete Anomalies）

  插入异常（Insert Anomalies）

- 因此，就诊关系模式R
  不是一个好的关系模式。

Dname	Dlevel	Dsalary	Pname	Fsum
罗晓	主任医师	3200	张珍	¥30.00
杨勋	副主任医师	2800	张珍	¥50.00
杨勋	副主任医师	2800	刘景	¥55.00
杨勋	副主任医师	2800	张柳	¥58.00
邓英超	主治医师	2400	李秀	¥75.00
罗晓	主任医师	3200	傅伟相	¥35.00

- 消除冗余和异常现象（**模式分解**）
  - R1（Dname，Dlevel）
  - R2（Dlevel，Dsal）
  - R3（Dname，Pname，Fsum）

Dname	Dlevel	Dsalary	Pname	Fsum
罗晓	主任医师	3200	张珍	¥30.00
杨勋	副主任医师	2800	张珍	¥50.00
杨勋	副主任医师	2800	刘景	¥55.00
杨勋	副主任医师	2800	张柳	¥58.00
邓英超	主治医师	2400	李秀	¥75.00
罗晓	主任医师	3200	傅伟相	¥35.00

（a）R_1的实例

Dname	Dlevel
罗晓	主任医师
杨勋	副主任医师
邓英超	主任医师

（b）R_2的实例

Dlevel	Dsalary
副主任医师	2800
主任医师	3200
主治医生	2400

（c）R_3的实例

Dname	Pname	Fsum
罗晓	张珍	¥30.00
杨勋	张珍	¥50.00
杨勋	刘景	¥55.00
杨勋	张柳	¥58.00
邓英超	李秀	¥75.00
罗晓	傅伟相	¥35.00

- 什么是数据依赖？

  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

- 数据依赖的类型

  函数依赖（Functional Dependency，简记为FD）

  多值依赖（Multivalued Dependency，简记为MVD）

- 函数依赖是指一个关系表中属性（列）之间的联系。
- 函数依赖是关系中属性之间在语义上的关联特性。
- 函数依赖关注一个属性或属性集与另外一个属性或属性集之间的依赖，亦即两个属性或属性集之间的约束。
- 数据库设计者根据对关系中的**属性的语义（业务需求）**理解确定函数依赖，确定约束的所有元组的函数依赖集，并获知属性间的语义关联。

- 符号说明：

  R表示一个关系的模式；

  U＝{A1，A2，…，An}是R的所有属性的集合；

  F是R中函数依赖的集合；

  r是R所取的值；

  t[X]表示元组t在属性X上的取值。例如 t[Dname] = ‘杨勋’

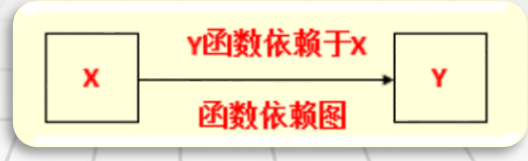
- 函数依赖定义

  设有一关系模式 $R(U)$，X 和 Y 为其属性 U 的子集，即 $X \subseteq U$，$Y \subseteq U$。设 t，s 是关系 R 中的任意两个元组，如果 $t[X]=s[X]$，则 $t[Y]=s[Y]$。那么称 Y 函数依赖于 X，或 X 函数决定 Y。也可称 FD $X \rightarrow Y$ 在关系模式 $R(U)$ 上成立。

- 函数依赖图

  左部称为决定因子

  右部称为依赖因子。



- 定义

如果 $Y \subseteq X$，显然 $X \rightarrow Y$ 成立，这称为平凡函数依赖（Trivial Functional Dependency）。

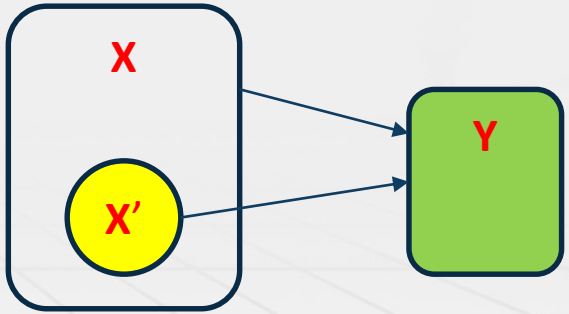
- 平凡函数依赖必然成立，它不反映新的语义。
  例如：{Dname, Pname} → {Pname}。

- 平常所指的函数依赖一般都指非平凡函数依赖。

- 定义

设 X、Y 是某关系的不同属性集，如 $X \to Y$，且不存在 $X' \subset X$，使 $X' \to Y$，则 Y 称完全函数依赖（Full Function Dependency）于 X，记为 $X \xrightarrow{f} Y$；否则称 Y 部分依赖（Partial Functional Dependency）于 X，记为 $X \xrightarrow{p} Y$。

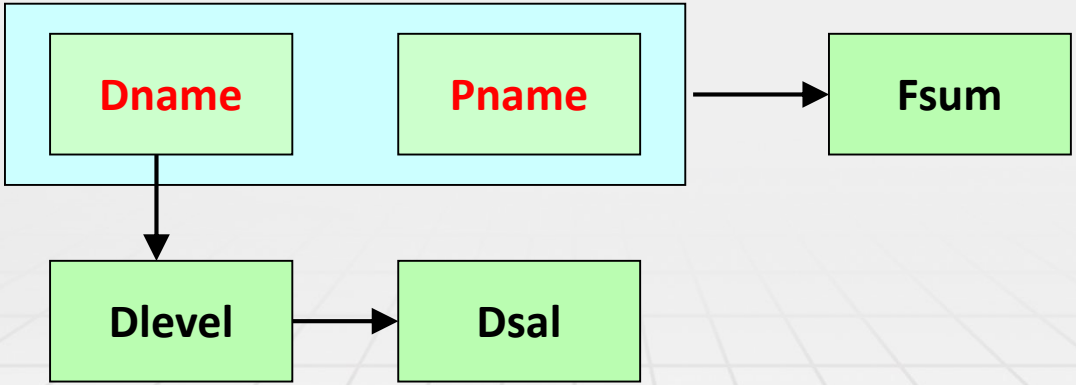
- 完全函数依赖用来表明函数依赖的**决定因子**中的**最小属性集**。



- 属性集Y完全函数依赖于属性集X，如果满足下列条件：
  - Y函数依赖于X。
  - Y不函数依赖于X的任何真子集。

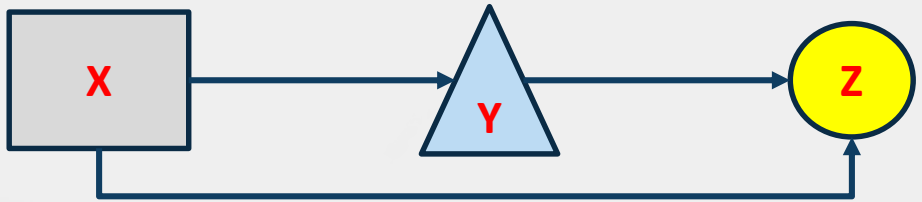
- 示例

在 R（Dname，Dlevel，Dsal，Pname，Fsum）中，{Dname,Pname}是主键，故{Dname,Pname} → Dlevel，但Dname → Dlevel，故{Dname,Pname} $\xrightarrow{p}$ Dlevel，而{Dname,Pname} $\xrightarrow{f}$ Fsum 。



- 定义

设 X，Y，Z 是某关系的不同的属性集，如果 $X \to Y$，$Y \not\to X$，$Y \to Z$，则称 Z 对 X 传递函数依赖（Transitive Functional Dependency）。

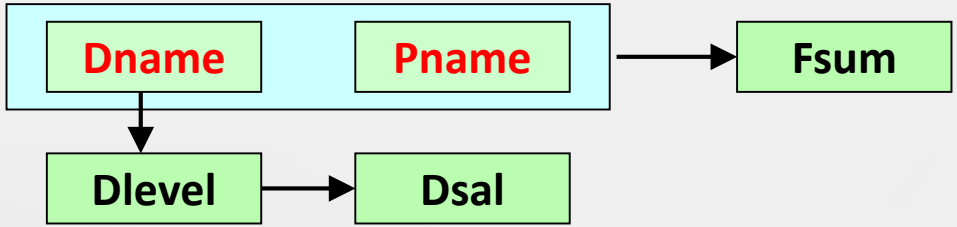


- 直接依赖

由于有了条件 $Y \not\to X$，说明 X 与 Y 不是一一对应的；
否则，$X \leftrightarrow Y$，Z 就直接函数依赖于 X，而不是传递函数依赖于 X 了。

- 举例：

在就诊关系R中，存在函数依赖Dname→Dlevel，Dlevel→Dsal，所以Dname→ Dsal。



- 函数依赖中需要解决的问题：**从一些已知的函数依赖去判断另外一些函数依赖是否成立？**

- 例如

如果A→B和B→C在某个关系中成立，记F={A→B，B→C}，那么A→C在该关系中是否成立的问题称为逻辑蕴涵问题，若A→C成立，则说F逻辑蕴涵A→C，记作：F ⊨ A→C。

- 定义

设 F 是 R 的函数依赖集，X，Y 是 R 的属性子集，$X \rightarrow Y$ 是 R 的一个函数依赖。如果一个关系模式 R 满足 F，则必然满足 $X \rightarrow Y$，这时称 F 逻辑蕴涵 $X \rightarrow Y$，或表示为 $F \models X \rightarrow Y$。

- 一个关系模式可能有多个函数依赖形成函数依赖集，现在有一个新的函数依赖不在函数依赖集里，但能从集合里根据一定的**规则**推导出来，就说那个集合**逻辑蕴涵**这个新的函数依赖。

- 举例

$X \to Z$ 并不是显式地表现出来，而是从 $X \to Y$ 和 $Y \to Z$ 推出的，这可表示为 $\{X \to Y, Y \to Z\} \models X \to Z$

- 如果一个函数依赖能够由集合中的其他函数推出，则该函数依赖是多余的。

- 闭包(*)

> 函数依赖集合 F 所逻辑蕴涵的函数依赖的全体称为 F 的闭包(Closure)，记为 F^+，即 $F^+=\{X\rightarrow Y \mid F \models X\rightarrow Y\}$。

- 函数依赖集的闭包（也成为**完备集**）定义了由给定函数依赖集所能推导出的所有函数依赖。
- 通过F得到F^+的算法可以由**Armstrong公理**推导出来。

- **无冗余**的函数依赖集和函数依赖的完备集（闭包）是好的关系设计。
- 根据已知函数依赖集，推导其它函数依赖时所依据的规则称为**推理规则**。
- 函数依赖的推理规则最早出现在Armstrong的论文里。这些规则常被称为“Armstrong公理”。

- 自反性（Reflexivity）

如果$Y \subseteq X \subseteq U$，则$X \rightarrow Y$成立。这是一个平凡函数依赖。

- 增广性（Augmentation）

如果$X \rightarrow Y$成立，且$Z \subseteq U$，则$XZ \rightarrow YZ$成立。

式中，XZ和YZ是$X \cup Z$和$Y \cup Z$的简写。

- 传递性（Transitivity）

如果$X \rightarrow Y$，$Y \rightarrow Z$成立，则$X \rightarrow Z$成立。

- 扩展规则（*）

合并性（Union）：$\{X \to Y, X \to Z\} \models X \to YZ$
伪传递性（Pseudo-transitivity）：$\{X \to Y, WY \to Z\} \models XW \to Z$
分解性（Decomposition）：$\{X \to Y, Z \subseteq Y\} \models X \to Z$
复合性（Composition）：$\{X \to Y, W \to Z\} \models XW \to YZ$
通用一致性（General Unification）：$\{X \to Y, W \to Z\} \models X \cup (W - Z) \to YZ$

证明与思考

- Armstrong公理是正确的，即如果X→Y是从F用推理规则导出，那么X→Y在F^+中。
- Armstrong公理（FD推理规则{A1，A2，A3}）是完备的（Complete）。
- 推理规则的正确性是指“从FD集F使用推理规则推出的FD必定在F^+中”，而推理规则的完备性是指“F^+中的FD都能从F集使用推理规则推出”。
- 即正确性保证了推出的所有FD都是正确的，完备性保证了可以推出所有被蕴涵的FD。这些就保证了推导的有效性和可靠性。

- 用函数依赖定义**候选码与主码**：

> 设 K 为 $R(U, F)$ 中的属性或属性组合，若 $K \xrightarrow{f} U$，则 K 为 R 的候选码（Candidate Key）。若候选码多于一个，则选定其中的一个为主码（Primary Key）。

- 包含在任何候选码中的属性称**为主属性（Prime Attribute）**。不包含在任何候选码中的属性称为**非主属性（Non-Key Attribute）**。

- 最简单的情况，单个属性是码。最极端的情况，整个属性组是码，称为**全码（All-key）**。

- 候选码与主码的举例

在关系模式R1（Dno，Dlevel，Dsal）中Dno是码

在关系模式R2（Dno，Pno，Fsum）中的属性组合（Dno，Pno）是码。

关系模式R3（Dno，Pno，Mno）表示医生给患者开具的药品，假设一个医生可以给多个患者看病，一位患者可以选择不同的医生就诊，不同的医生可以给患者开具不同的药品，因此（Dno，Pno，Mno）为R3的码，即全码。

由 FD 推理规则的分解性可知，如果 $X \to \{A_1, A_2, ..., A_k\}$，则 $X \to A_i (i = 1, 2, ..k)$。
由合并性可知，如果 $X \to A_i (i = 1, 2, ..k)$，则 $X \to \{A_1, A_2, ..., A_k\}$，
故 $X \to \{A_1, A_2, ..., A_k\}$ 和 $X \to A_i (i = 1, 2, ..k)$ 是等价的。
由此可知，候选码唯一地决定一个元组；也可以说，候选码决定每个属性。

- 用函数依赖定义**外码**

> 关系模式 R_1 中属性或属性组合 X 并非 R_1 的码，但 X 是另一个关系模式 R_2 的码，则称 X 是 R_1 的外码（Foreign Key）。

- 举例：

在关系模式R2（Dno，Pno，Fsum）中，Dno不是码，
但Dno是关系模式R1（Dno，Dlevel，Dsal）的码，
则Dno是关系模式R2的外码。

- **主码与外码提供了一个表示关系之间联系**的手段。如上述关系模式R1和R2的联系就是通过Dno来体现的。

- 由函数依赖可以引起的更新异常问题，同样，多值依赖和连接依赖也会引起类似的问题。解决这些问题的途径就是按照**“一事一地”**的原则，对关系模式进行分解，使之表达的**语义概念单纯化**。
- 模式分解就是将模式分解到最小，将数据转换为规范格式以避免冗余。
- 关系模式R的分解就是用两个或两个以上关系来替换R，分解后的关系模式的属性集都是R中属性的子集，其并集与R的属性集相同。
- 模式分解帮助**消除不良设计中的一些问题，如冗余、不一致或异常。**

- 模式分解的定义

设有关系模式 $R(U)$，属性集为 U，若用一关系模式的集合 $\{R_1(U_1), R_2(U_2), \cdots, R_k(U_k)\}$ 来取代，其中 $U = \bigcup_{i=1}^{k} U_i$，则称此关系模式的集合为 R 的一个分解，以 $\rho = \{R_1, \cdots, R_k\}$ 表示。

- 模式分解的问题

R 称为泛关系模式，R 对应的当前值 r 称为泛关系。数据库模式 ρ 对应的当前值 σ 称为数据库实例，它是由数据库模式中的每一个关系模式的当前值组成，用 $\sigma = < r_1, \cdots, r_2 >$ 表示。但这里就有两个问题：

- σ 和 r 是否等价，即是否表示同样的数据。这个问题用“**无损分解**”特性表示。
- 在模式 R 上有一个 FD 集 F，在 ρ 的每一个模式 R_i 上有一个 FD 集 F_i，那么 $\{F_1, \cdots, F_k\}$ 与 F 是否等价。这个问题用“**保持依赖**”特性表示。

- 定义：一个关系表被分解成两个或两个以上的小表，通过**连接**被分解后的小表可以获得原始表的内容，则称为**无损连接分解**。

设 R 是一个关系模式，F 是 R 上的一个 FD 集。R 分解成数据库模式 $\rho=\{R_1,\cdots,R_k\}$。如果对 R 中满足 F 的每一个关系 r，有：

$$r=\pi_{R_1}(r)\bowtie\pi_{R_2}(r)\bowtie\cdots\bowtie\pi_{R_k}(r)$$

那么称分解 ρ 相对于 F 是“无损连接分解”（Lossless Join Decomposition），简称为“无损分解”；否则，称为“损失分解”（Lossy Decomposition）。

- 例如：将R（X，Y，Z）分解成R1（X，Y）和R2（X，Z），如果**X**是R1和R2的**共同属性或属性集，且存在函数依赖集F={X→Y，X→Z}**，则该分解是无损的。

- 举例

r	A	B	C
	1	1	1
	1	2	1

(a)

r1	A	B
	1	1
	1	2

(b)

r2	A	C
	1	1

(c)

未丢失信息的分解

无损中的损是指信息丢失。如果一个分解不是无损分解，则所得结果的元组数总比原来的多（增加了噪声，但把原来的信息丢失了）。所谓“有损”就损在出现多余的元组上。

r	A	B	C
	1	1	4
	1	2	3

(a)

r1	A	B
	1	1
	1	2

(b)

r2	A	C
	1	4
	1	3

(c)

r1 ⋈ r2	A	B	C
	1	1	4
	1	1	3
	1	2	4
	1	2	3

(d)

丢失信息的分解

- 无损分解测试算法——追逐法（Chase）*

输入：关系模式 $R(A_1,\cdots A_n)$；

R 上成立的函数依赖集 F；

R 上的分解 $\rho=\{R_1,\cdots,R_k\}$。

输出：ρ 相对于 F 是否无损分解。

方法：

① 构造一张 k 行 n 列的矩阵表格，每列对应一个属性 A_j，每行对应一个模式 R_i。如果 A_j 在 R_i 中，那么在表格的第 i 行第 j 列处填上符号 a_j；否则填上 b_{ij}。

② 把表格看成模式 R 的一个关系，反复检查 F 中每个 FD 在表格中是否成立，若不成立，则修改表格中的值。修改方法如下。

对于 F 中一个 FD $X\rightarrow Y$，如果表格中有两行在 X 值上相等，在 Y 值上不相等，那么把这两行在 Y 值上也改成相等的值。如果 Y 值中有一个是 a_j，那么另一个也改成 a_j；如果没有 a_j，那么用其中一个 b_{ij} 替换另一个值（尽量把下标 ij 改成较小的数）。一直到表格不能修改为止（这个过程称为 Chase 过程）。

③ 若修改的最后一张表格中有一行是全 a，即 $a_1a_2\cdots a_n$，那么称 ρ 相对于 F 是无损分解，否则称损失分解。

- Chase是一个普遍的算法，无论一个关系模式分解为多少个关系模式，都可以用此算法进行检验。
- 如果一个关系模式**一分为二**，可以简化检验。

设 $\rho=\{R_1(U_1),R_2(U_2)\}$ 是关系模式 R（U）的一个分解，则 ρ 是无损分解的充分必要条件是：

$(U_1\cap U_2)\rightarrow(U_1-U_2)$ 或 $(U_1\cap U_2)\rightarrow(U_2-U_1)$。

- 例如

医生关系模式 R（Dno，Dlevel，Dsal）上函数依赖集 F={Dno→Dlevel，Dlevel→Dsal}。

若 $R_1(Dno,Dlevel)$ 和 $R_2(Dlevel,Dsal)$ 为 R 的一个模式分解，

因为 $Dlevel\rightarrow Dsal$，令 $U_1=\{Dno,Dlevel\}$，$U_2=\{Dlevel,Dsal\}$。

则 $U_1\cap U_2=\{Dlevel\}$，$U_2-U_1=\{Dsal\}$。

故 $(U_1\cap U_2)\rightarrow(U_2-U_1)$。

故 ρ 是无损分解。

- 所有的模式分解必须是**无损的**。
- 无损连接分解总是**关于特定函数依赖集F定义的**。
- 模式分解能**消除数据冗余和操作异常现象**。
- 但是分解以后，检索操作需要做笛卡尔积或连接操作，这将**付出时间代价**。
- 一般认为，为了消除冗余和异常现象，对模式进行分解是值得的。

- 模式分解的另一个特性是分解的过程中**能否保持函数依赖集**，如果不能保持函数依赖，那么数据的语义就会出现混乱。
- 模式分解要保持函数依赖，因为函数依赖集F中的**每一个函数依赖都代表数据库的一个约束（业务语义）**。
- 如果某个分解能保持函数依赖集，那么在数据输入或更新时，只要每个关系模式本身的函数依赖约束被满足，就可以**确保整个数据库中的语义完整性不被破坏**。显然这是一种良好的特性。

- 定义

设 $\rho=\{R_1,\cdots,R_k\}$ 是 R 的一个分解，F 是 R 上的 FD 集，如果有 $\bigcup_{i=1}^{k}\prod R_i(F)\vdash F$，那么称分解 ρ 保持函数依赖集 F。

- 举例：

设医生关系模式R（Dname，Dlevel，Dsal）。假设每个医生只有一个职称级别，每个职称级别只有一个工资数目。那么R上函数依赖集F={Dname→Dlevel，Dlevel→Dsal}。

如果将R分解成 ρ={R1(Dname,Dlevel)，R2(Dname,Dsal)}，可以验证这个分解是无损分解。

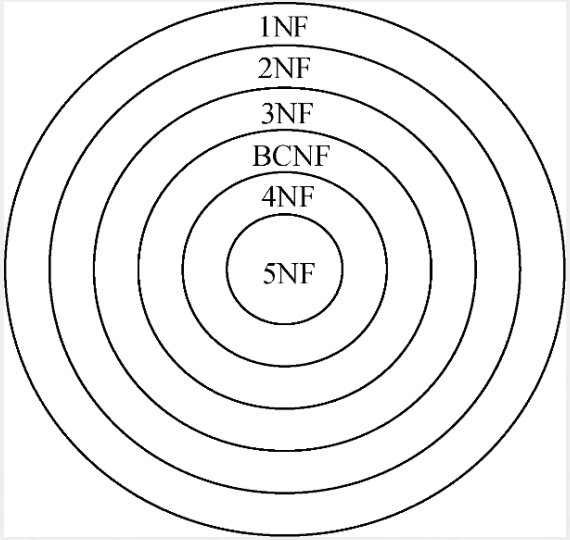
但是，R1上的函数依赖是F1={Dname→Dlevel}，R2上的函数依赖是F2={Dname→Dsal}。从这两个函数依赖**推导不出在R上成立的函数依赖Dlevel→Dsal**。因此分解 ρ 把函数依赖Dlevel→Dsal丢失了，即 ρ 不保持函数依赖。

- 关系模式分解的**两个特性**实际上涉及两个数据库模式的等价问题，这种等价包括**数据等价**和**依赖等价**两个方面。
  - 数据等价是指两个数据库实例应表示**同样的信息内容，用“无损分解”衡量**。如果是无损分解，那么对关系反复的投影和连接都不会丢失信息。
  - 依赖等价是指两个数据库模式应有**相同的依赖集闭包**。在依赖集闭包相等情况下，**数据的语义是不会出错的**。
- 违反数据等价或依赖等价的分解很难说是一个很好的设计模式。
- 但是要同时达到无损分解和保持函数依赖的分解也不是一件容易的事情，需要认真权衡。

- 范式（Normal Forma，NF）是一种关系的状态，也是**衡量关系模式好坏的标准**。

- 范式的种类（1NF，2NF，3NF，BCNF, 4NF, 5NF）与数据依赖有着直接的联系。**在关系模式中存在函数依赖时就有可能存在数据冗余，引出数据操作异常现象。**

例如：就诊关系模式R（Dname，Dlevel，Dsal，Pname，Fsum）不是一个好的设计，因为存在冗余信息（重复存储的职称和工资）。数据冗余不仅浪费存储空间，而且会使数据库难以保持数据的一致性。



- 范式可以用于确保数据库模式中没有各种类型的异常和不一致性。为了确定一个特定关系是否符合范式要求，需要**检查关系中属性间的函数依赖，而不是检查关系中的当前实例**。

- 一个**规范化的模式**有**最小的数据冗余**，它要求除了与元组进行连接的**外键**（在另一个关系中是主键的属性组）之外，数据库实例中的其他属性的值都**不能被复制（冗余）**。

- **规范化**主要作为验证和改进逻辑数据库设计的工具，使得逻辑设计能够：
  - **满足特定约束**
  - **避免不必要的数据重复。**

- 定义：在关系模式R的每个关系r中，如果每个属性值都是不可再分的**原子值**，那么称R是第一范式（1NF）的模式。

- 1NF不允许每个元组的每个属性对应一组值、一行值或两个值的组合。简单地说，**1NF中不允许出现“表中有表”**的现象。
  - 1NF是关系模式应具有的最起码的条件。**满足1NF的关系称为规范化的关系**；否则称为非规范化的关系。关系数据库研究的关系都是规范化的关系。

职工号	姓名	职称	工资			扣除		实发
			基本工资	工龄工资	职务工资	房租费	水电费	
86051	陈 平	讲 师	105	9.5	15	6	12	115.5
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-

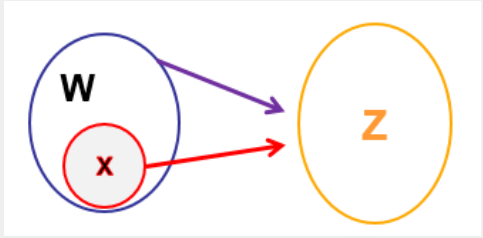
- 如果关系模式中存在**部分函数依赖**，那么它就不是一个好的关系模式，因为它很可能出现**数据冗余和操作异常现象**。因此，需要对这样的关系模式进行分解，以排除局部函数依赖，使模式达到2NF的标准。

- 2NF定义：如果关系模式R∈1NF，且每个**非主属性**（不是组成候选码的属性）**完全函数依赖**于候选码，那么称R属于2NF的模式。

- 2NF是基于完全函数依赖的，只有在主键是**复合属性**下才可能不符合2NF。

- 2NF是通往更高范式的中间步骤，它消除了1NF存在的部分问题。

- 2NF分解算法：将关系模式R分解成2NF模式子集
  - 设有关系模式R(U)，主键是W，R上还存在函数依赖 X→Z，其中Z是非主属性和 X ⊂ W，则 W→Z 就是一个局部依赖。此时应该把R分解成两个模式：
  - ① R_1（XZ），主键是X；
  - ② R_2（U-Z），主键仍为W，外键是X（参考R_1）。
  - 利用外键和主键的连接可以从R_1和R_2重新得到 R。
  - 如果R_1和R_2还不是2NF，则重复上述过程，一直到数据库模式中每一个关系模式都是2NF为止。

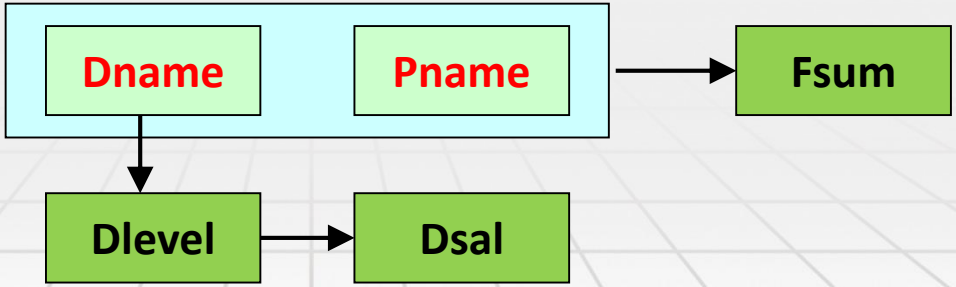


2NF举例

设有关系模式R（Dname，Pname，Dlevel，Dsal，Fsum)的属性分别表示医生姓名、患者姓名、医生职称级别、医生工资和诊疗费用。（Dname，Pname）是R的候选码。

如果R上有两个FD：（Dname，Pname）→（Dlevel，Dsal）和Dname →（Dlevel，Dsal），因此前面一个FD是局部依赖，所以R不是2NF。此时R会出现冗余和异常。例如，某个医生为N个病人看病，则在关系中会出现N个元组，而医生的职称级别和工资就会重复N次。

如果将R分解为R1（Dname，Dlevel，Dsal）和R2（Dname，Pname，Fsum）后，局部依赖（Dname ，Pname ）→（Dlevel，Dsal）就消失了，R1和R2都是2NF了。



- 定义：如果关系模式R∈1NF，且每个**非主属性**都**不传递依赖**于R的候选码，那么称R属于3NF的模式。

- 在3NF中，关系模式是由主键和一组相互独立的非主属性组成的，它满足两个条件：
  （1）R中的非主属性相互独立；
  （2）R中的非主属性函数依赖于主键。

- 3NF分解算法： 将关系模式R分解为3NF模式集。
  - 设关系模式R（U），主键是W，R上还存在FD X→Z，其中Z是非主属性，$Z \not\subseteq X$ 且X不是候选键，这样**W→Z就是一个传递依赖**。此时应把R分解成两个模式：
  - ① R1（XZ），主键是X；
  - ② R2（U-Z），主键仍是W，外键是X（参考R1）。
  - 利用外键和主键相匹配机制，R1和R2通过连接可以重新得到R。
  - 如果R1和R2还不是3NF，则重复上述过程，一直到数据库模式中每一个关系模式都是3NF为止。

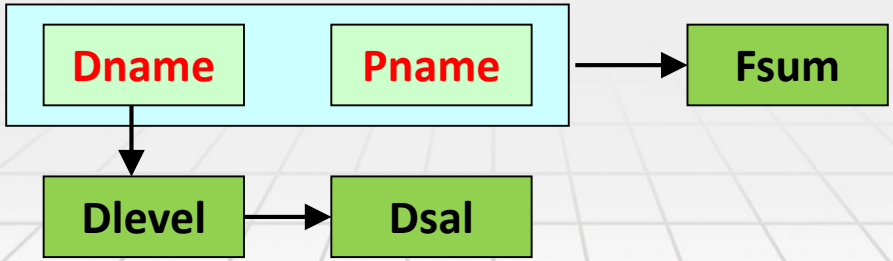
- 3NF举例：

R2（Dname，Pname，Fsum）是2NF模式，而且也是3NF模式。

但是R1（Dname，Dlevel，Dsal）是2NF模式，但不一定是3NF。因为如果R1中存在函数依赖Dname → Dlevel和Dlevel → Dsal，那么Dname → Dsal就是一个传递依赖，即R1不是3NF模式。

此时R1的关系也会出现冗余和异常。例如，R2中存在M个职称同为主任级别的医生，则R1中需要重复存储M个相同的工资数目。

如果将R1分解为R11（ Dname ，Dlevel）和R12（Dlevel，Dsal）后，Dname→Dsal就不会出现在R11和R12中，因此R11和R12都是3NF的模式。



- 定理：如果R是3NF模式，那么R也是2NF模式。
- **局部依赖和传递依赖**是关系模式产生**数据冗余和异常**的两个重要原因。
- 由于3NF模式中不存在非主属性对候选键的局部依赖和传递依赖，因此3NF消除了很大一部分存储异常，具有较好的性能。
- 而对于非1NF、1NF和2NF的关系模式，由于它们的性能较差，通常不宜作为数据库模式，需要将这些关系模式变换为3NF或更高级的范式。这种变换过程称为“关系的规范化处理”。

- 在3NF模式中，并未排除**主属性**对候选键的传递依赖，因此有必要提出更高一级的范式。
- BC范式（Boyce-Codd Normal Form，BCNF），由Boyce与Codd提出的，比上述的3NF又进了一步，通常认为BCNF是修正的第三范式，有时也称为扩充的第三范式。
- BCNF定义：如果关系模式R∈1NF，且（包含主属性和非主属性）都**不传递依赖**于R的候选码，那么**每个属性**称R是BCNF的模式。
- 由BCNF的定义可以得到以下结论，一个满足BCNF的关系模式有：
  - 所有非主属性对每一个码都是完全函数依赖。
  - 所有的主属性对每一个不包含它的码，也是完全函数依赖。
  - 没有任何属性完全函数依赖于非码的任何一组属性。

● 举例

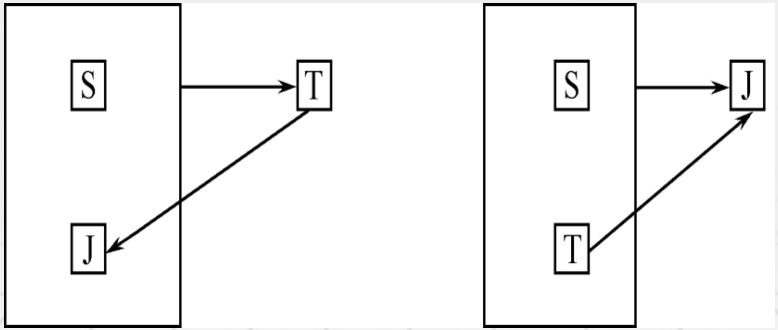
关系模式SJP(S, J, P)中，S是学生，J表示课程，P表示名次。每一个学生选修每门课程的成绩有一定的名次，每门课程中每一名次只有一个学生（即没有并列名次）。

- 由语义可得到函数依赖：(S, J)→P；(J, P)→S
- (S, J)与(J, P)都可以作为候选码。
- 关系模式中没有属性对码传递依赖或部分依赖，所以 SJP∈3NF。
- 除(S, J)与(J, P)以外没有其他决定因素，所以SJP∈BCNF。

● 举例

关系模式STJ(S, T, J)中，S表示学生，T表示教师，J表示课程。每一教师只教一门课。每门课有若干教师，某一学生选定某门课，就对应一个固定的教师。

- 由语义可得到函数依赖：(S, J)→T；(S, T)→J；T→J
- 因为没有任何非主属性对码传递依赖或部分依赖，STJ ∈ 3NF。
- 因为T是决定因素，而T不包含码，所以STJ ∈ BCNF。



- 举例

关系模式R（Bno，Bname，Author）的属性分别表示书号、书名和作者名。假如每个书号只有一个书名，但不同的书号可以有相同的书名；每本书可以有多个作者合写，但每个作者参与编著的书名应该互补相同。

- R上的FD如下：Bno→Bname和（Bname，Author）→Bno
- 因此R的关键码是（Bno，Author）或（Bname，Author），因而模式R的属性都是主属性，R是3NF模式。
- 但根据两个FD可知，属性Bname传递依赖于关键码（Bname，Author），因此R不是BCNF。
- 例如，一本书由多个作者编写时，其书名与书号之间的联系在关系中将多次出现，会导致数据冗余和操作异常。
- 如果将R分解为R1（Bno，Bname）和R2（Bno，Author），则能够解决上述问题，且R1和R2都是BCNF。
- 但这样分解可能会导致新的问题，例如，这个分解把（Bname，Author）→Bno丢失了，数据语义将会引起新的矛盾。

- BCNF分解算法：将R无损分解且保持依赖地分解成3NF模式集。

  ① 对于关系模式R和R上成立的FD集F，先求出F的**最小依赖集**，然后再把最小依赖集中那些左部相同的FD用合并性合并起来。

  ② 对最小依赖集中每个FD X→Y去构成一个模式(XY)。

  ③ 在构成的模式集中，如果每个模式都不包含R的候选码，那么把候选码作为一个模式放入模式集中。

- 举例：

设关系模式R（ABCDE），R的最小依赖集为{A→B，C→D}。从依赖集可知R的候选码为ACE。

先根据最小依赖集，可知ρ={AB, CD}。然后再加入由候选码组成的模式ACE。因此最后结果ρ={AB, CD, ACE}是一个3NF模式集，R相对于该依赖集是无损分解且保持函数依赖。

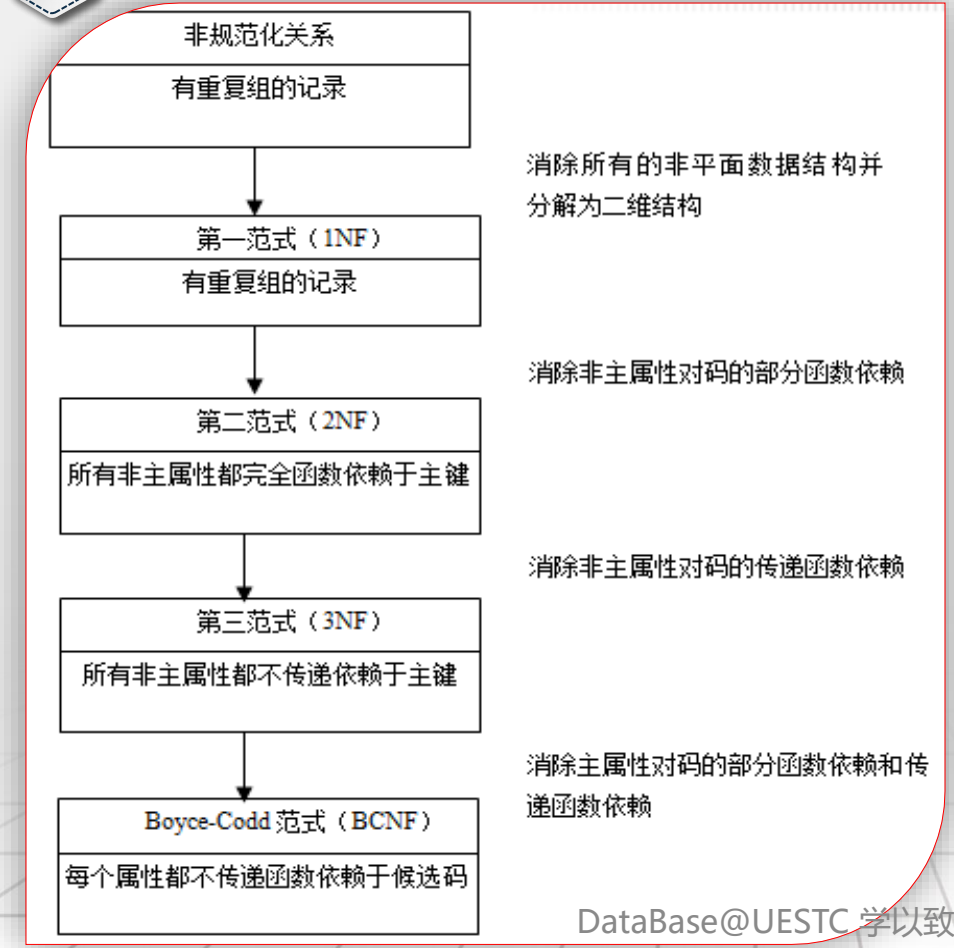
- 定理：如果R是BCNF模式，那么R也是3NF模式。
- 但是，若R∈3NF，则R未必属于BCNF。
- 3NF和BCNF是在函数依赖的条件下对模式分解所能达到的分离程度的测度。

  一个数据库模式中的关系模式如果都属于BCNF，那么在函数依赖范畴内，它已实现了彻底的分离，已消除了插入和删除的异常。
- 3NF的“不彻底性”表现在可能存在**主属性对码的部分依赖和传递依赖**。

- 满足范式要求的数据库设计是结构清晰的，同时可避免数据冗余和操作异常。这**不意味着不符合范式要求的设计一定是错误的**。
- 关系模式分解一般应具有3个特性：
  - 达到BCNF，或3NF；
  - 无损分解；
  - 保持函数依赖。
- 数据库设计者在设计和关系数据库时，应做权衡，尽可能使数据库模式保持最好的特性。
  - 一般尽可能设计成BCNF模式集。
  - 如果设计成BCNF模式时达不到保持函数依赖的特点，那么只能降低要求，设计成3NF模式集，以求达到保持函数依赖和无损分解的特点。

- 一个好的模式设计方法应符合3条原则：**表达性、分离性和最小冗余性**。
  - **表达性**涉及两个数据库模式的等价问题，即**数据等价和语义等价**，分别用无损分解和保持依赖集来衡量。
  - **分离性**是指在关系中只存储有直接联系的属性值，把有间接联系的属性值放在不同的表中。
    - 实际上“分离”就是清除冗余和异常现象。
    - 分离的基准是一系列范式。
    - 在分解成BCNF模式集时，分离与依赖等价有时是不兼容的。
  - **最小冗余性要求分解后的模式个数和模式中属性总数应最少。**
    - 目的是节省存储空间，提高操作效率，消除不必要的冗余。
    - 但要注意，实际使用时并不一定要达到最小冗余。因为有时带点冗余对提高查询速度是有好处的。尤其对于那些更新频度不高，查询频度极高的数据库系统更是如此。

6. 模式设计原则



- 关于模式分解的几个重要事实
  - 若要求分解保持函数依赖，那么模式分解总可以达到3NF，但不一定能达到BCNF；
  - 若要求分解既保持函数依赖，又具有无损连接性，可以达到3NF，但不一定能达到BCNF；
  - 若要求分解具有无损连接性，那一定可以达到4NF。