《数据库原理及应用教程（MySQL）》

第2章 关系数据库

01 OPTION 关系的形式化定义及性质

02 OPTION 关系模式与关系数据库模式

03 OPTION 关系的码和关系的完整性

1. 关系的形式化定义

（1）域（Domain）

定义2.1 域是一组具有相同数据类型的值的集合，又称为值域（用D表示）。

例如，整数、实数和字符串的集合都是域。

✓ 域中所包含的值的个数称为域的基数（用m表示），例如，以1.4.3节中的表1-1所示的教师关系 t 为例

D_1={刘杨，石丽，顾伟，赵礼，赵希希，张刚}，m_1=6；

D_2={男，女}，m_2=2；

D_3={26，30，32，36，40，50}，m_3=6。

其中，D_1、D_2、D_3 分别表示教师关系中的姓名域、性别域和年龄域的集合。

1. 关系的形式化定义

(2) 笛卡尔积（Cartesian Product）

定义2.2 给定一组域D_1，D_2，…，D_n（它们包含的元素可以完全不同，也可以部分或全部相同），其笛卡尔积为：

$$D_1 \times D_2 \times \ldots \times D_n = \{ (d_1, d_2, \ldots, d_n) \mid d_i \in D_i, i=1, 2, \ldots, n \}$$

- ✓ 每一个元素（d_1，d_2，…，d_n）中的每一个值d_i叫做一个分量（Component），分量来自相应的域（$d_i \in D_i$）
- ✓ 每一个元素（d_1，d_2，…，d_n）叫做一个n元组（n-Tuple），简称元组（Tuple）。但元组是有序的，相同分量d_i的不同排序所构成的元组不同。如，以下三个元组是不同的，（1，2，3）≠（2，3，1）≠（1，3，2）

1. 关系的形式化定义

(2) 笛卡尔积 (Cartesian Product)

✓ 若D_i（i=1，2，…，n）为有限集，D_i中的集合元素个数称为Di的基数，用m_i（i=1，2，…，n）表示，则笛卡尔积$D_1 \times D_2 \times \cdots \times D_n$的基数M（即元组（$d_1$，$d_2$，…，$d_n$）的个数）为所有域的基数的累乘之积

例如，教师关系中的姓名域D_1和性别域D_2的笛卡尔积为：

$D_1 \times D2$={(刘杨,男),(刘杨,女),(石丽,男),(石丽,女),(顾伟,男),(顾伟,女),(赵礼,男),(赵礼,女),(赵希希,男),(赵希希,女),(张刚,男),(张刚,女)}

其中，刘杨、石丽、顾伟、赵礼、赵希希、张刚、男、女都是分量，(刘杨,男),(刘杨,女)等是元组，$D_1 \times D_2$的基数$M=m_1 \times m_2=6 \times 2=12$，即集合中元组的个数为12。

- 笛卡尔积（所有可能的搭配）

食堂点餐有三列可选：

- 主食：{米饭，面条}
- 配菜：{鸡肉，素菜，鱼}
- 饮料：{可乐，茶}

不管味道合不合，你只要把**每一列各取一个**，就得到一份套餐。所有能点出的套餐一股脑儿罗列出来，就是这三个集合的**笛卡尔积**。一共有 2×3×2=12 种。

> 关键词：**每列各取一个 + 全部列出 + 数量是个数相乘**

给定一组“域”（就是取值范围）$D_1, D_2, \ldots, D_n$，笛卡尔积记为

$$D_1 \times D_2 \times \cdots \times D_n$$

它是由形如$(d_1, d_2, \ldots, d_n)$的**有序**组合构成的集合，其中每个d_i都来自自己的域D_i。

- **有序**：位置有意义，(刘杨, 男) 与 (男, 刘杨) 不是一回事，因为第一位规定放“姓名”，第二位规定放“性别”。
- **分量（component）**：$(d_1, d_2, \ldots)$里每一位就是一个分量，对应各自的域。
- **基数（个数）**：如果每个域的元素个数分别是 $m_1, m_2, \ldots, m_n$，那整个笛卡尔积里共有 $m_1 \times m_2 \times \cdots \times m_n$ 个 n 元组。

- 域1（姓名）：{刘杨，石丽，顾伟，赵礼，赵希希，张刚}（6个）
- 域2（性别）：{男，女}（2个）

它们的笛卡尔积 $D_1 \times D_2$ 就是把这 6 个名字与男女两种性别**逐一配对**得到的所有 12 个二元组，比如：(刘杨, 男)、(刘杨, 女)、(石丽, 男).....共 12 项。

1. 关系的形式化定义

(2) 笛卡尔积（Cartesian Product）

✓ 笛卡尔积可用二维表的形式表示，例如，笛卡尔积$D_1 \times D_2$的二维表形式为

tn 姓名	sex 性别
刘杨	男
刘杨	女
石丽	男
石丽	女
顾伟	男
顾伟	女
赵礼	男
赵礼	女
赵希希	男
赵希希	女
张刚	男
张刚	女

4）和数据库有什么关系？

（a）关系/表的本质

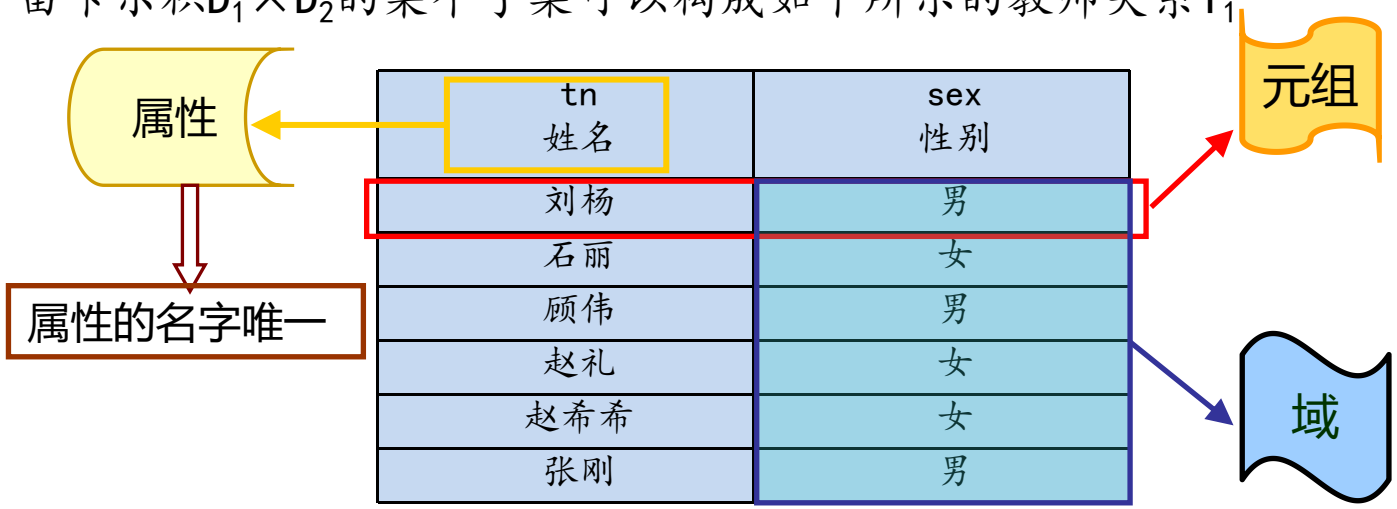
- 在关系模型里，一个关系（表）可以看作是**若干域的笛卡尔积中的一个子集**。
- 解释：理论上“姓名×性别”会有所有可能的搭配（这就是笛卡尔积），但真实的“人员表”只会出现**真实存在**的那几行（即从“所有可能”里挑出**满足约束**的那部分）。
  所以：**笛卡尔积 = 全部可能；表 = 满足约束的那部分可能。**

1. 关系的形式化定义

（3）关系（Relation）

定义2.3　笛卡尔积$D_1 \times D_2 \times \ldots \times D_n$的任一子集称为定义在域$D_1$，$D_2$，…，$D_n$上的n元关系（Relation），可用R（$D_1$，$D_2$，…，$D_n$）表示。其中，R表示关系的名字，n是关系的目或度（Degree）。

✓ 例如，笛卡尔积$D_1 \times D_2$的某个子集可以构成如下所示的教师关系T_1



① 什么是“关系”（Relation）——把“所有可能”变成“有意义的表”

一句话：

关系 = **若干域的笛卡尔积中的一个“有意义的子集”**，并给它起个名字 R。

- **域（Domain）**：某一列允许出现的取值范围。比如“性别 ∈ {男, 女}”“年龄 ∈ 整数[0,150]”。
- **属性（Attribute）**：表头的一列，就是“这个位置放什么”的**名字**，如 `name、sex、age`。
  同一个关系里**属性名必须唯一**（不然查询会分不清）。
- **元组（Tuple）**：一行数据，写成有序小括号 `(刘杨，男)`。
- **度（Degree）**：关系里**有多少个属性**。1 个属性叫一**元关系**，2 个属性叫二**元关系**，以此类推。

> 类比：
> “笛卡尔积”像把所有食材随便搭到一起；
> “关系”是餐厅**真正卖的菜**——从“所有可能搭配”里筛掉没意义的，只保留**符合约束**的那些组合，再给这道菜起个名字（R）。

用图里的教师关系示例

- 域：`name` 来自“姓名集合”，`sex` 来自 `{男，女}`
- 关系 `T1(name, sex)`：把“姓名×性别”的**可能搭配**里，保留实际存在的那几条，排成两列的表。
- `(刘杨，男)` 就是一条**元组**；`name`、`sex` 是两列的**属性**；这个关系的**度**是 2（= 二元关系）。

1. 关系的形式化定义

(3) 关系（Relation）

✓ **关系的几点说明：**

（1）在关系R中，当n=1时，称为单元关系。当n=2时，称为二元关系，以此类推。

（2）关系中的元组通常用t表示，关系中元组个数是关系的基数。

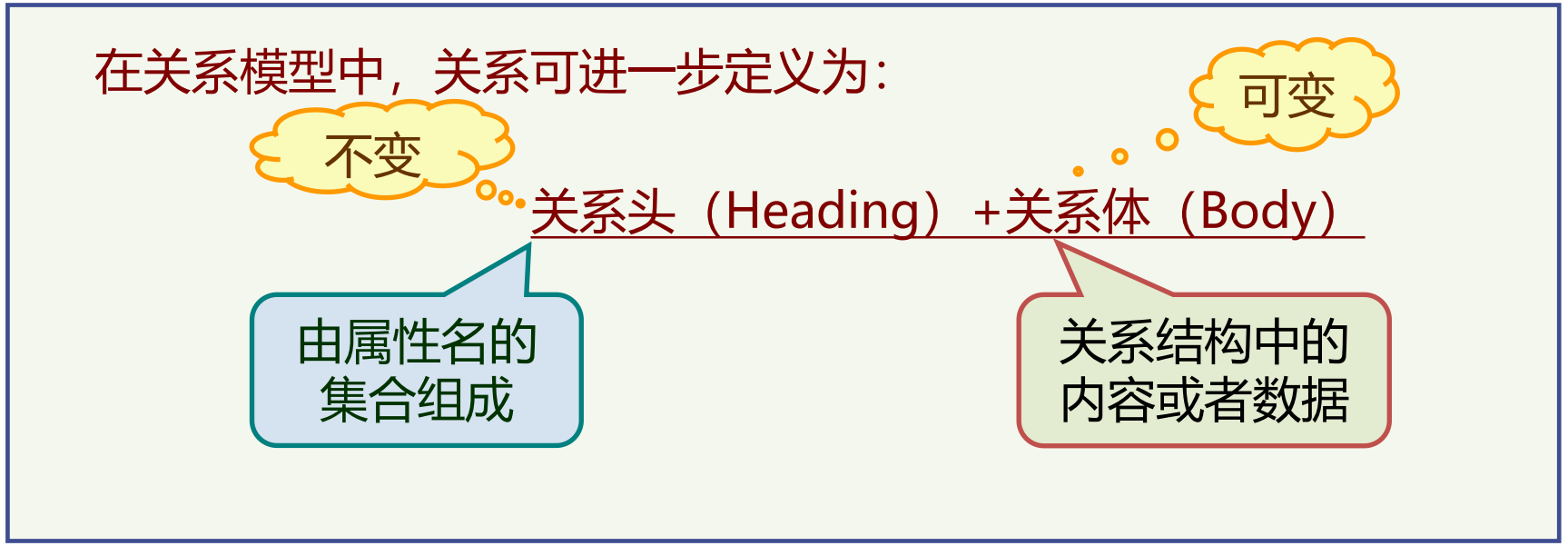
（3）关系中的不同域（列）的取值可以相同，为了加以区别，必须对每个域（列）起一个名字，称为属性（Attribute），n元关系必有n个属性，属性的名字唯一；属性的取值范围称为值域，等价于对应域D_i（i=1，2，…，n）的取值范围。具有相同关系框架的关系称为同类关系。

（4）在数学上，关系是笛卡尔积的任意子集，但在实际应用中，关系是笛卡尔积中所取的有意义的子集。

1. 关系的形式化定义

(3) 关系（Relation）

定义2.4 定义在域D_1，D_2，…，D_n（不要求完全相异）上的关系由关系头（Heading）和关系体（Body）组成。



③ 关系 = 关系头（Heading） + 关系体（Body）——“表模板”和“表内容”

这页是**最实用的工程化视角**：

- **关系头（Heading）**：**不变的**结构部分
  - 就是**属性名 + 各属性的域**（也可理解为“列名 + 列的类型/取值范围”）。
  - 在数据库里，对应**表结构/模式（Schema）**：列名、类型、约束。
- **关系体（Body）**：**可变的**内容部分
  - 由**若干元组**（行）组成。
  - 在数据库里，就是**数据本身**，增删改查都发生在这里。

- “**Heading** = 表单的**空白模板**（写明“第 1 格填姓名、只能中文；第 2 格填性别、只能男/女；第 3 格填年龄、0–150”）。”
- “**Body** = 学生**实际递交的表单**叠起来的一摞。”
- “你今天加一条数据，是往 **Body** 多放了一张表单；”
- “你变更“年龄允许区间”，是在改 **Heading**（结构变化，影响全局）。”

2. 关系的性质

- ✓ 列是同质的，即每一列中的分量必须来自同一个域，必须是同一类型的数据。
- ✓ 不同的属性可来自同一个域，但不同的属性必须有不同的名字。例如，假设某关系中的两个属性“职业”和“兼职”，它们可以来自同一个域{教师，工人，辅导员}。
- ✓ 列的顺序可以任意交换。但交换时，应连同属性名一起交换，否则将得到不同的关系。
- ✓ 关系中元组的顺序（即行序）可任意，在一个关系中可以任意交换两行的次序。因为关系是以元组为元素的集合，而集合中的元素是无序的，所以作为集合元素的元组也是无序的。
- ✓ 关系中不允许出现相同的元组。因为数学上集合中没有相同的元素，而关系是元组的集合，所以作为集合元素的元组应该是唯一的。

2. 关系的性质

✓ 关系中每一分量必须是不可分的数据项，也就是说，不能出现“表中有表”的现象。满足此条件的关系称为规范化关系，否则称为非规范化关系。

- 例如，以下左表是非规范化关系，可以把其中的属性“籍贯” 分成两个新的属性，即“省（区市）” 、“市/县” ，将其规范化，如右表所示。

姓名	籍　贯	
	省	市 / 县
张强	吉林	长春
王丽	山西	大同

非规范化的关系

姓　名	省	市 / 县
张强	吉林	长春
王丽	山西	大同

规范化的关系

1. 设计一个关系 `CourseSelect(studentId, courseId, term)`
   - 说出它的 **度**；每个属性的**域**；再给出 3 条**元组**示例。
2. 把上面的 **Heading** 单独写出来（列名 + 域/类型），再单独写一个 **Body**（若干行数据）。
3. 思考：如果误把 `term` 写成 `VARCHAR` 而不是 `ENUM{'2025春','2025秋'}`，将来会出现什么数据质量问题？

关系设计：`CourseSelect(studentId, courseId, term)`

1) 度（Degree）

- 度 = 3（三个属性/列）

2) 每个属性的“域”（Domain）

- **studentId**：学号集合 S（例如：10 位数字或学校定义的学号编码；等价于 *Student(id)* 的取值范围）
- **courseId**：课程号集合 C（例如：字母数字课程编码，如 *DB201*、*AI101*；等价于 *Course(id)* 的取值范围）
- **term**：固定枚举集合 {`2025春`, `2025秋`}

3) 三条元组（Tuple）示例

- `(10001, 'DB201', '2025春')`
- `(10002, 'DB201', '2025春')`
- `(10001, 'AI101', '2025秋')`

关系头（Heading）与关系体（Body）

MySQL 习惯写法（ENUM 表达域）

sql　　Copy code

```sql
CREATE TABLE CourseSelect (
  studentId  VARCHAR(10) NOT NULL,
  courseId   VARCHAR(16) NOT NULL,
  term       ENUM('2025春','2025秋') NOT NULL,
  PRIMARY KEY (studentId, courseId, term),
  FOREIGN KEY (studentId) REFERENCES Student(id),
  FOREIGN KEY (courseId)  REFERENCES Course(id)
);
```

注：上面把 **(studentId, courseId, term)** 设成主键，体现“同一学生同一课程同一学期只选一次”的业务约束；课堂上可顺带点出“主键=在关系体中唯一标识元组的属性组”。

Body（若干行数据）

等价于下面这个集合（或表格）：

- {
  (10001, 'DB201', '2025春'),
  (10002, 'DB201', '2025春'),
  (10001, 'AI101', '2025秋')
  }

3. 思考：如果误把 `term` 写成 `VARCHAR` 而不是 `ENUM{'2025春','2025秋'}`，将来会出现什么数据质量问题？

思考题答案：若把 `term` 误写成 `VARCHAR`（而不是受限的枚举/检查约束），会带来哪些数据质量问题？

1. **非法/不一致取值泛滥**
   - 例如：`'2025 春'`（多了空格）、`'2025春季'`、`'春2025'`、`'2025-春'`、`'2025秋 '`（尾随空格）、`'2026春'`（年份写错）、`'Fall2025'`（中英混用）等都可能被插入。
2. **统计与分组失真**
   - `GROUP BY term` 时会把“2025春” “2025 春” “2025春季”分成三组，**同一学期被拆散**，报表口径混乱。
3. **唯一性/主键被绕开**
   - 即使你把 `(studentId, courseId, term)` 设为主键，错别字/变体（如 `'2025 春'`）会被系统当成“不同的 term”，从而**绕开唯一约束**，出现同一人同一课同一学期“重复选课”的脏数据。
4. **连接/查询漏数**
   - 查询条件写 `WHERE term='2025春'`，含空格或变体的行**匹配不上**，导致**漏查**、漏统计。
5. **数据清洗成本增加**
   - 需要额外写触发器/应用层校验/定期清洗脚本去合并同义取值，**维护成本高**且容易遗漏。
6. **排序与语义顺序失真**
   - 纯文本排序是按字典序，混入空格/后缀后顺序会很诡异；枚举/受限域可以定义清晰的**语义顺序**（如先春后秋）。

“**域就是规则**。不把规则写进类型/约束里，迟早要把时间花在补锅上。”

01 OPTION
关系的形式化定义及性质

02 OPTION
关系模式与关系数据库模式

03 OPTION
关系的码和关系的完整性

1. 关系模式

表是怎么被‘规定’出来的？答案就是关系模式。

定义2.5 关系的描述称为关系模式（Relation Schema）。它可以形式化地表示为：

R（U，D，DOM，F）

R--关系名

U--属性名集合

D--属性所来自的域

DOM--属性向域的映像集合

F--属性间数据的依赖关系集合

属性名

✓ 关系模式通常简记为：R（U）或R（A_1，A_2，…，A_n）

符号	英文	中文含义
R	Relation	关系名
U	Universe of Attributes 或 Attributes Set	属性集合
D	Domain	域（属性取值范围）
DOM	Domain Mapping	属性到域的映射
F	Functional Dependencies	函数依赖集合

更完整的关系模式通常写作：

$$R(U, D, DOM, F)$$

R--关系名

U--属性名集合

D--属性所来自的域

DOM--属性向域的映像集合

F--属性间数据的依赖关系集合

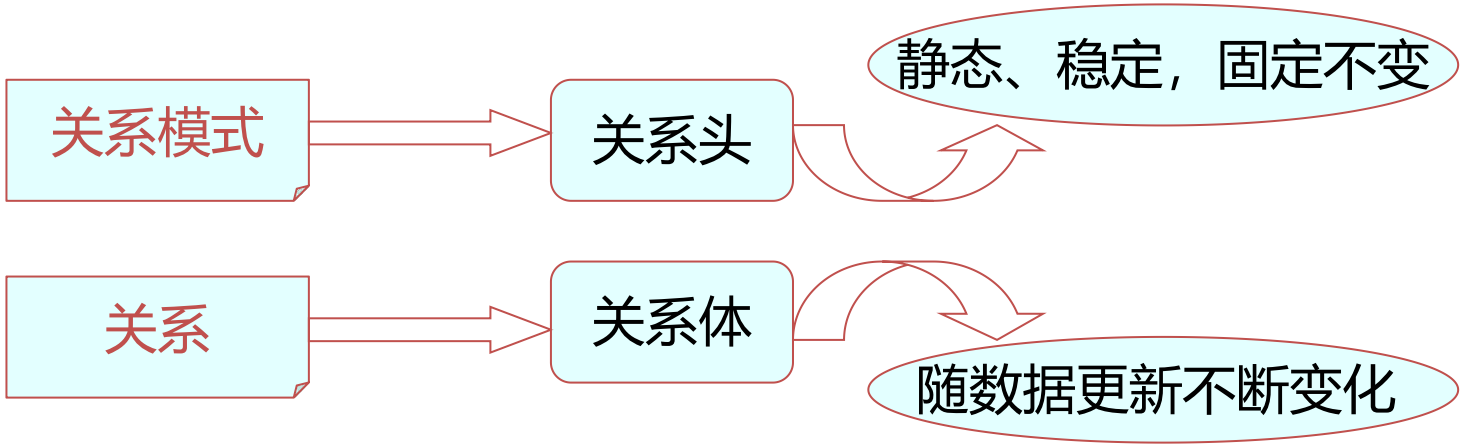
- **R —— 关系名**（表名）。
  像给孩子起名：`Student`、`CourseSelect`。
- **U —— 属性名集合**（一张表有哪些列）。
  比如 `U = {studentId, courseId, term}`。注意：**列名在一个表里必须唯一**。
- **D —— 域的集合**（会用到哪些“类型/取值范围”）。
  不是“data”，而是“domain”！例如：学号是“10 位数字”、性别是 `{男, 女}`、学期是 `{2025春, 2025秋}`，这些**域**都属于 D。
- **DOM —— 从属性到域的映射**（给每一列“穿尺码”）。
  写作 `DOM(studentId)=学号域`，`DOM(sex)={男, 女}`，`DOM(term)={2025春, 2025秋}`……
  一句话：**哪一列用哪种取值范围**，在 DOM 里一对一说清楚。
- **F —— 依赖/约束的集合**（列与列的“因果关系”/规则）。
  典型是**函数依赖**和**键约束**：
  例如 `studentId → name, major`（学号能唯一确定姓名与专业），`(studentId, courseId, term)` 是主键等。
  在 SQL 里，F 常由 `PRIMARY KEY / UNIQUE / FOREIGN KEY / CHECK` 这些约束来“落地”。

- **关系模式**：

$$\mathbf{CourseSelect}(U, D, DOM, F)$$

- $U = \{\text{studentId}, \text{courseId}, \text{term}\}$
- $D = \{\text{学号域}, \text{课程号域}, \{2025\text{春}, 2025\text{秋}\}\}$
- $DOM(\text{studentId}) = \text{学号域}$
  $DOM(\text{courseId}) = \text{课程号域}$
  $DOM(\text{term}) = \{2025\text{春}, 2025\text{秋}\}$
- $F = \{\ (\text{studentId}, \text{courseId}, \text{term}) \text{ 是主键 }\}$（可再加外键约束等）

1. 关系模式



新增 1000 条选课记录，**模式**变了吗？

把 `term` 从 `{2025春, 2025秋}` 扩展到 `{2025春, 2025秋, 2026春}`，这是改哪里？

1. 关系模式

例如，在第1章的表1-1~表1~5所示的教学数据库teaching中，共有五个关系，其关系模式可分别表示为：

- 教师（教师号，姓名，性别，年龄，职称，工资，专业，院系）
- 学生（学号，姓名，性别，年龄，专业，院系）
- 课程（课程号，课程名，课时）
- 选课（学号，课程号，成绩）
- 授课（教师号，课程号）

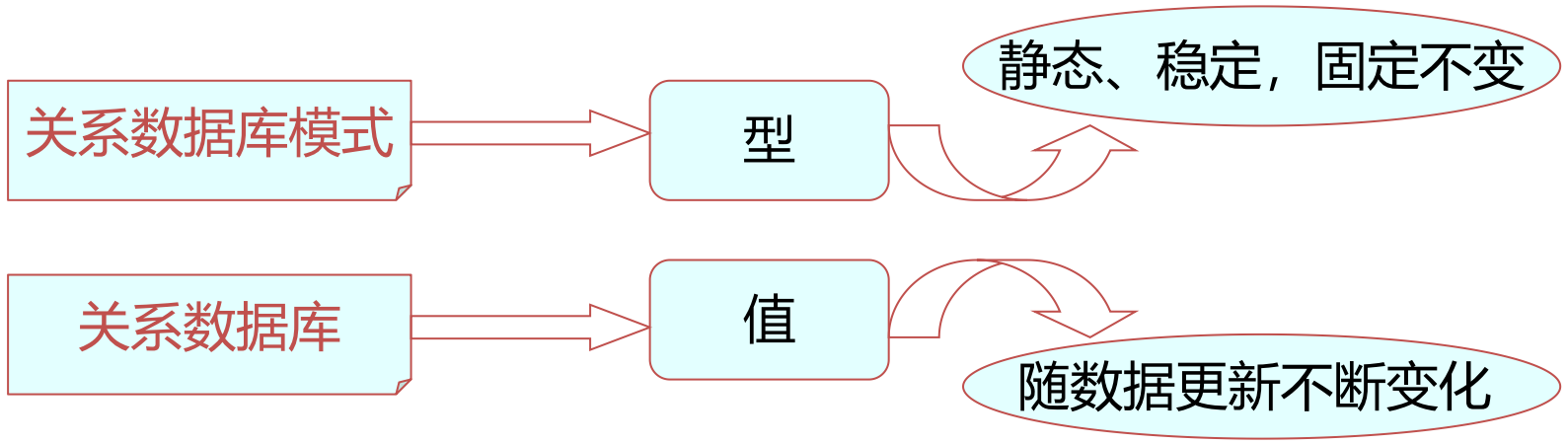
1. 关系模式

对于上述每个关系模式，又有其相应的实例

- 例如，在第1章的表1-1中，与教师关系模式对应的数据库中的实例如下表所示。

t1	刘杨	男	40	教授	3610.5	计算机	信息学院
t2	石丽	女	26	讲师	2923.3	信息	信息学院
t3	顾伟	男	32	副教授	3145	计算机	信息学院
t4	赵礼	女	50	教授	4267.9	自动化	工学院
t5	赵希希	女	36	副教授	3332.67	数学	理学院
t6	张刚	男	30	讲师	3012	自动化	工学院

2. 关系数据库模式



01 OPTION 关系的形式化定义及性质

02 OPTION 关系模式与关系数据库模式

03 OPTION 关系的码和关系的完整性

1. 候选码和主码

（1）候选码

能**唯**一锁定一行记录的属性组合，而且**再也裁不掉**任何一列，否则就不唯一了。

定义2.6 设关系R有属性A_1，A_2，…，A_n，其属性集K=（A_i，A_j，…，A_k），当且仅当满足下列条件时，K被称为候选码。

- 唯一性（Uniqueness），关系R的任意两个不同元组，其属性集K的值是不同的。
- 最小性（Minimum），组成关系键的属性集（A_i，A_j，…，A_k）中，任一属性都不能从属性集K中删掉，否则将破坏唯一性的性质。

✓ “课程关系”中的课程号能唯一标识每一门课程，则属性“课程号”是课程关系的候选码

✓ “授课关系”中，只有属性的组合“教师号+课程号”才能唯一地区分每一条授课记录，则属性集“教师号+课程号”是授课关系的候选码

✓ “选课关系”中“学号+课程号”的组合是唯一的，同时，“学号+课程号”满足最小性，从中去掉任一属性，都无法唯一标识选课记录

例子 A：Student(studentId, idCard, name, phone)

- 在多数学校里，`studentId`（学号）**唯一且最小** ⇒ 候选码①：`{studentId}`
- 如果 `idCard`（身份证号）也唯一 ⇒ 候选码②：`{idCard}`
- `{studentId, name}` 虽唯一，但**不最小**（去掉 name 仍唯一）⇒ 这是**超码**（Superkey），不是候选码

例子 B：CourseSelect(studentId, courseId, term, grade)

- 一名学生，同一门课，在同一学期只能选一次 ⇒ 候选码：`{studentId, courseId, term}`
  任何一列删掉都会出现重复，所以满足“最小性”。

1. 候选码和主码

（2）主码

当一个表有多个候选码时，从中挑一把作为日常使用的主键，其它候选码可用 UNIQUE 保留。

- ✓ 如果一个关系中有多个候选码，可以从中选择一个作为查询、插入或删除元组的操作变量，被选用的候选码称为主码
- ✓ 主码也称为主关系键、主键、关系键、关键字等，后续章节中，统一称为主码

当一个表有**多个候选码**时，**从中挑一把**作为日常使用的主键，其它候选码可用 `UNIQUE` 保留。

怎么挑？（工程常识）

- **稳定**（不随业务变化）：学号比手机号稳。
- **短小**（存储、索引友好）：单列优先于多列。
- **不含含糊值**：不得为 `NULL`。
- **对外引用方便**：常被外键引用的列要易用。

延续例子 A：Student

- 候选码有 `{studentId}`、`{idCard}`，通常选 `studentId` 当主码；`idCard` 加 `UNIQUE` 即可。

延续例子 B：CourseSelect

- 只有一个候选码 `{studentId, courseId, term}` ⇒ 它也就是主码。

> 规则点题：**每个关系有且只有一个主码**（但可以有多个候选码）。

1. 候选码和主码

（3）主属性和非主属性

- ✓ 主属性（Prime Attribute）是指包含在主码中的各个属性
- ✓ 非主属性（Non-Prime Attribute）是指不包含在任何候选码中的属性，也称为非码属性

- **主属性（Prime）**：出现在**任一候选码**中的属性。
- **非主属性（Non-Prime）**：不出现在任何候选码中的属性。

例子 A：Student

- 候选码：`{studentId}`、`{idCard}`
- **主属性**：`studentId, idCard`
- **非主属性**：`name, phone`

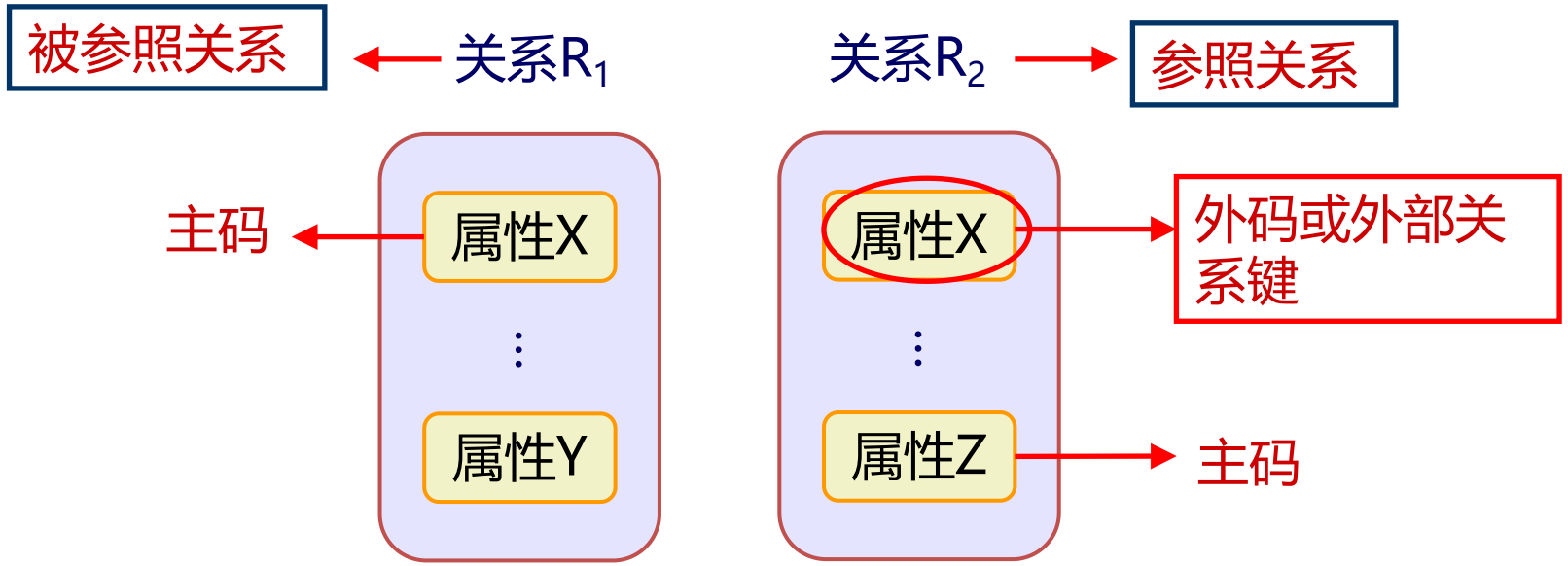
例子 B：CourseSelect

- 候选码（=主码）：`{studentId, courseId, term}`
- **主属性**：`studentId, courseId, term`
- **非主属性**：`grade`

> 这个区分会在范式/分解时派上大用场（比如判断部分函数依赖、传递依赖）。

2. 外码

定义2.7 如果关系R_2的一个或一组属性X不是R_2的主码，而是另一关系R_1的主码，则该属性或属性组X称为关系R_2的外码（Foreign key）或外部关系键（在后续章节中统一称为外码），并称关系R_2为参照关系（Referencing Relation），关系R_1为被参照关系（Referenced Relation）。



被参照关系的主码和参照关系的外码必须定义在同一个域上

假设有两个系统：

表1：公民表（Citizen）

身份证号	姓名
110101	张三
110102	李四
110103	王五

这里：

- **身份证号 = 主码（Primary Key）**
- 因为身份证号是 **唯一的**

表2：学生表（Student）

学号	姓名	身份证号
S001	张三	110101
S002	李四	110102

这里：

- **学生表里的“身份证号”就是外码**

因为：

- 它 **引用了公民表的身份证号**
- 但它 **不是学生表的主码**

二、为什么要外码？

假设没有外码约束：

你可能插入：

学号	姓名	身份证号
S003	赵六	999999

但 **公民表根本没有 999999 这个身份证。**

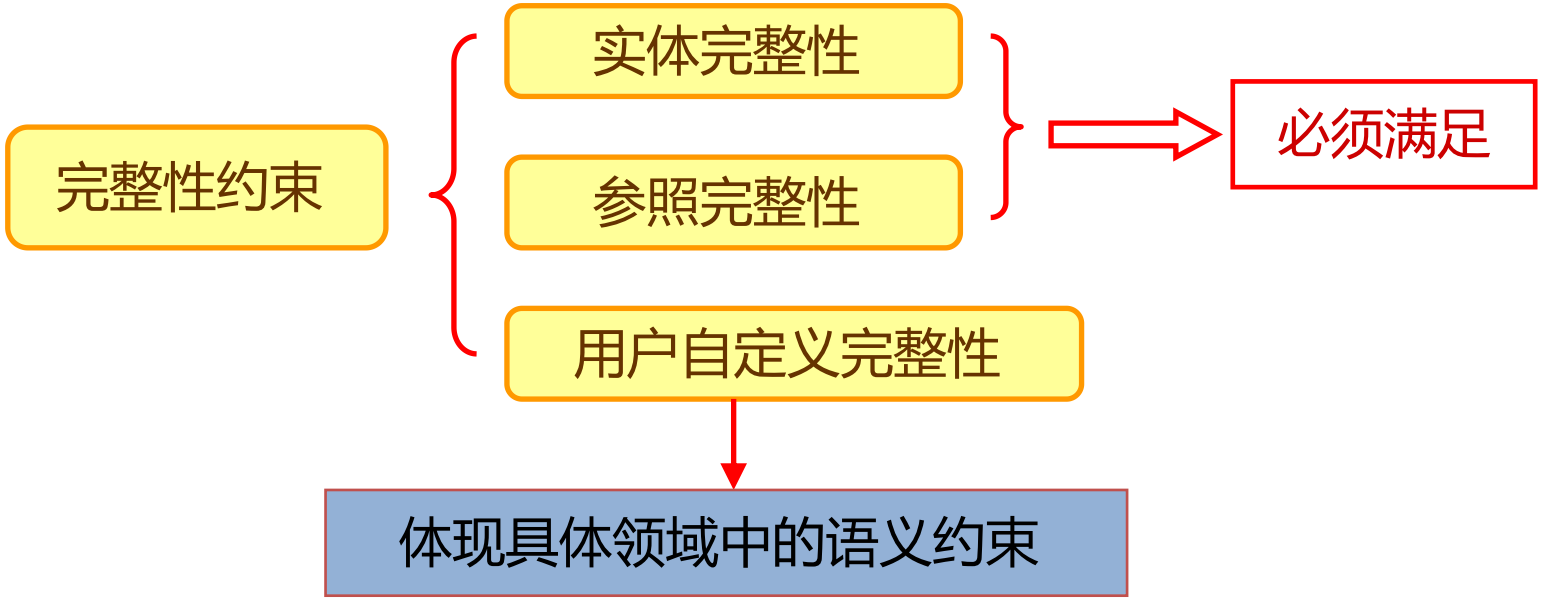
数据就乱了。

所以数据库规定：

> Student.身份证号 必须来自 Citizen.身份证号

这就是 **外码约束。**

3. 关系的完整性



3. 关系的完整性

(1) 实体完整性

✓ 实体完整性是指主码的值不能为空或部分为空

- 课程关系中的主码“课程号”不能为空
- 授课关系中的主码“教师号+课程号”不能部分为空，即“教师号”和“课程号”两个字段的取值都不能为空。

一、实体完整性（Primary Key / 唯一身份）

像“每个人一本护照、每件行李一个条形码”。**主键不能重复、不能为 NULL。**

有趣例子：

- 考场座位表：每个准考证号必须唯一；没有准考证（NULL）就进不了场。
- 图书馆副本：同一本书可以有多册，用 **(ISBN, 副本号)** 组成复合主键，才不会两册书“同体穿越”。
- 选课表：同一学期，同一学生对同一门课最多一条记录，用 **(student_id, course_id, term)** 做主键，杜绝“一人两报同一课”。

课堂梗：**“幽灵同学”**——插入一条没有学号的学生记录（NULL）会被数据库当场拦下。

3. 关系的完整性

（2）参照完整性

外卖平台有两张表：

1 商家表（Restaurant）

商家ID	商家名
R01	兰州拉面
R02	重庆小面
R03	沙县小吃

这里：

- **商家ID = 主码**

2 订单表（Order）

订单号	用户	商家ID
O001	张三	R01
O002	李四	R03
O003	王五	R02

这里：

- **商家ID = 外码**
- 指向 Restaurant.商家ID

数据库要求：

> 订单表里的商家ID，必须在商家表中存在。

例如：

下面这个订单就 **不允许存在**：

订单号	用户	商家ID
O004	赵六	R99 ❌

因为：

- **R99 在商家表不存在**

这就是 **参照完整性约束。**

假设商家表删除：

```
R03　沙县小吃
```

但订单表还有：

订单号	用户	商家ID
O002	李四	R03

数据库会有三种处理方式：

1 禁止删除（最常见）

```
ERROR：该商家还有订单
```

必须先删订单。

SQL叫：

```
RESTRICT
```

2 删除时订单也删除

删除商家 → 订单也删

```
O002 自动删除
```

SQL叫：

```
CASCADE
```

3 订单变成 NULL

删除商家后：

订单号	用户	商家ID
O002	李四	NULL

表示：

订单存在，但商家没了

SQL叫：

```
SET NULL
```

3. 关系的完整性

(3) 用户自定义完整性

✓ 用户自定义完整性是针对某一具体关系数据库的约束条件，它反映某一具体应用所涉及的数据必须满足的语义要求

- 属性值根据实际需要，要具备一些约束条件。
- 如规定选课关系中成绩属性的取值范围在0和100之间；某些数据的输入格式要有一些限制等。
- 关系模型应该提供定义和检验这类完整性的机制，以便用统一的、系统的方法处理它们，而不要由应用程序承担这一功能。

业务规则的“自由发挥”，比如：

- 成绩只能在 **0–100**。
- 年龄必须 ≥ **16** 才能选“酒类品鉴”课。
- 一个班**最多一个“班长（男）”和一个“班长（女）”**。
- 每门课**容量上限**（比如 100 人）。