

第2章 关系数据库回顾



主要内容

- **数据库体系结构 (Database Architecture)**
- **关系数据模型 (Relational Data Model)**
- **SQL**

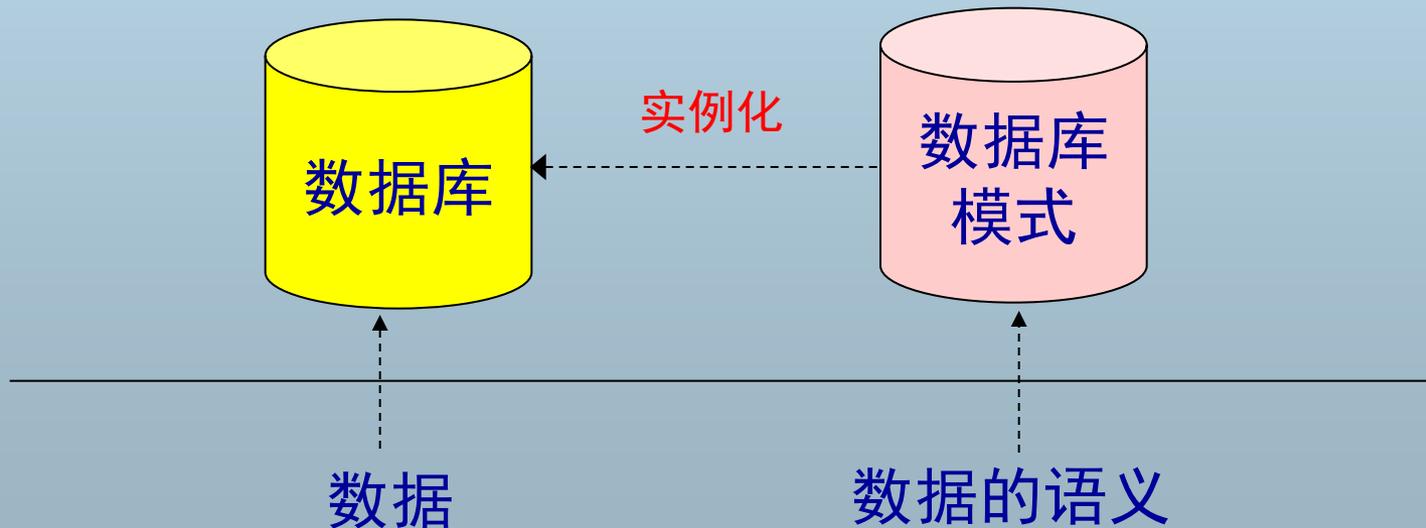
一、数据库体系结构

- 从模式的角度描述一般数据库系统的概念和结构
- 可以用于解释特定数据库系统的内部结构
- **ANSI/SPARC体系结构——三级模式结构+两级映象**
 - ◆ Oracle、Informix等SQL数据库系统的模式结构可通过ANSI/SPARC体系结构进行解释

1、数据库模式的概念

■ 模式（Schema）和实例（Instance）

- 模式是数据库中全体数据的逻辑结构和特征的描述，它仅仅涉及类型的描述，不涉及具体的值
- 模式的一个具体值称为模式的一个实例



2、模式和实例举例

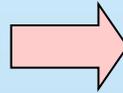
模式



- 学生表 (学号, 姓名, 年龄)
- 课程表 (课程号, 课程名, 学分)
- 选课表 (学号, 课程号, 成绩)

实际中的模式描述比本例要详细得多

两个实例



S001	张三	21
S002	李四	20

C001	数据库	4
C002	英语	6
C003	数学	6

S001	C001	90
S002	C001	80

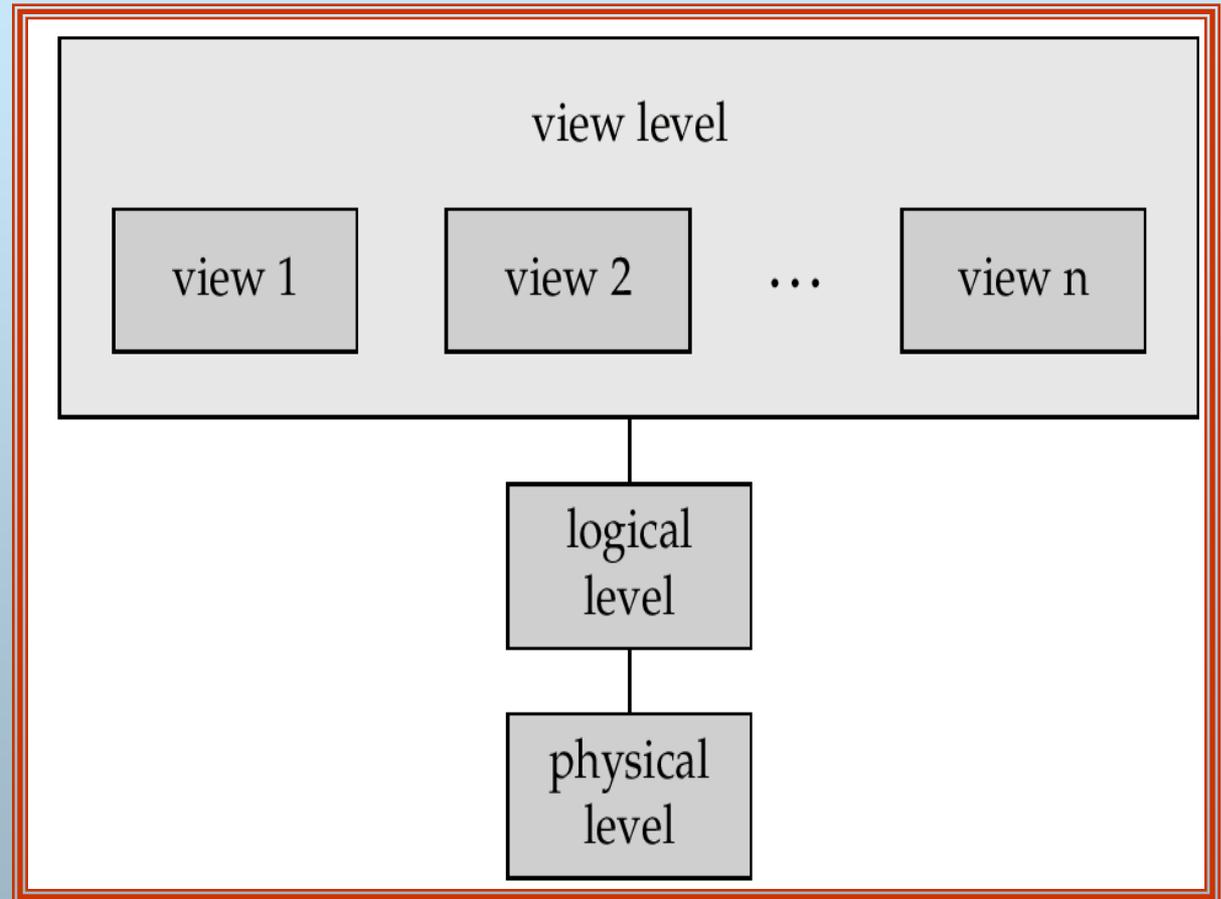
S001	张三	21
S002	李四	20
S003	王五	22

C001	数据库	4
C002	英语	6
C003	数学	6

S001	C001	90
S002	C001	80
S003	C001	90
S003	C002	96
S003	C003	98

3、数据库的三级模式结构

- 外模式
- 概念模式
- 内模式



(1) 概念模式（模式、逻辑模式）

- 数据库中全体数据的逻辑结构和特征的描述
 - 数据记录由哪些数据项构成
 - 数据项的名字、类型、取值范围
 - 数据之间的联系、数据的完整性等
- 不涉及数据物理存储的细节和硬件环境
- 一个数据库只有一个概念模式
- **概念视图**：概念模式的实例
- 通过模式**DDL**进行定义

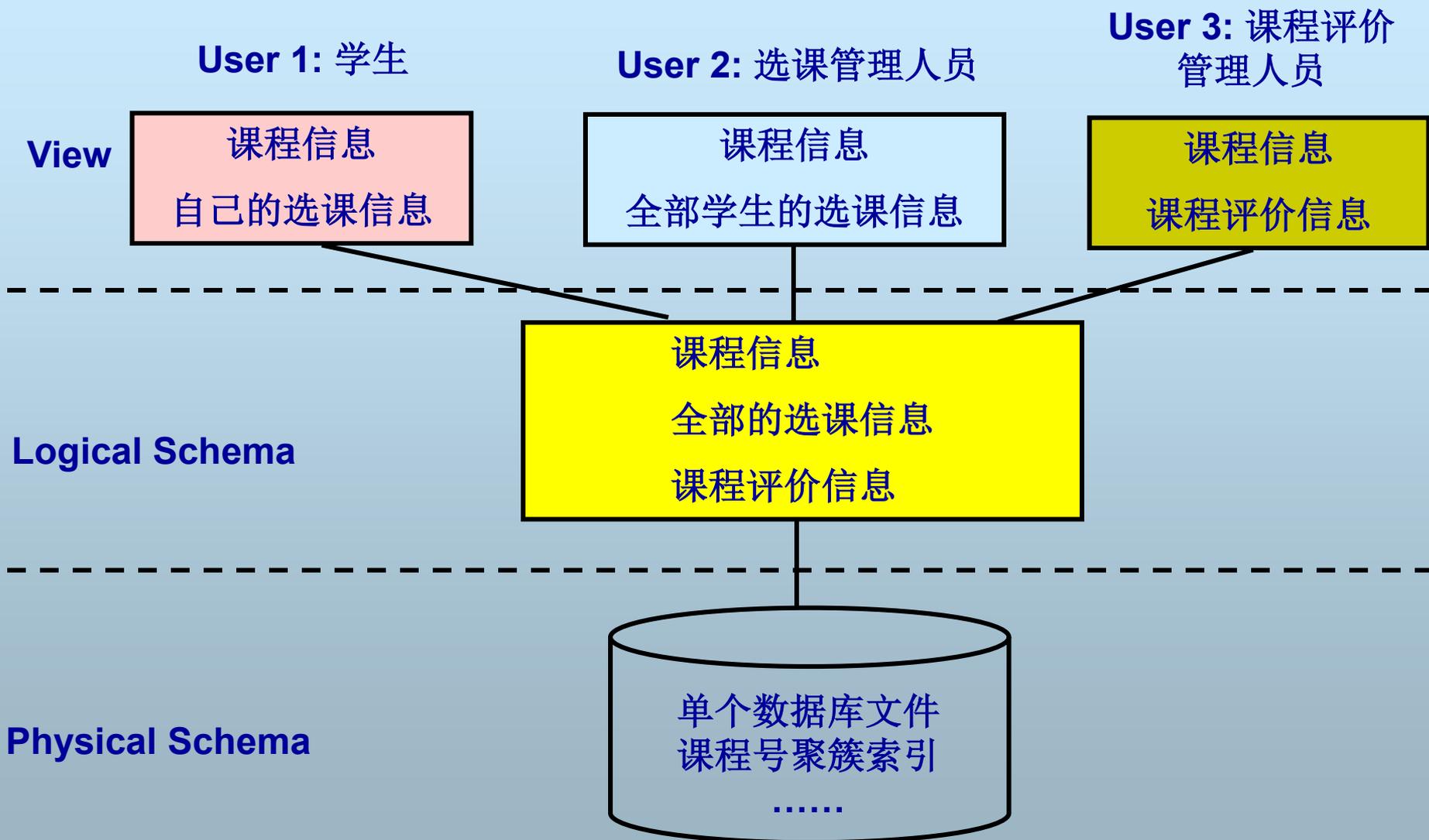
(2) 外模式（子模式、用户模式）

- 单个用户所看到的局部数据的逻辑结构和特征的描述
- 用户与数据库系统的数据接口，对于用户而言，外模式就是数据库
- 建立在概念模式之上，同一模式上可有多个不同的外模式
- **外部视图**：外模式的实例
- 通过子模式**DDL**进行定义

(3) 内模式（存储模式）

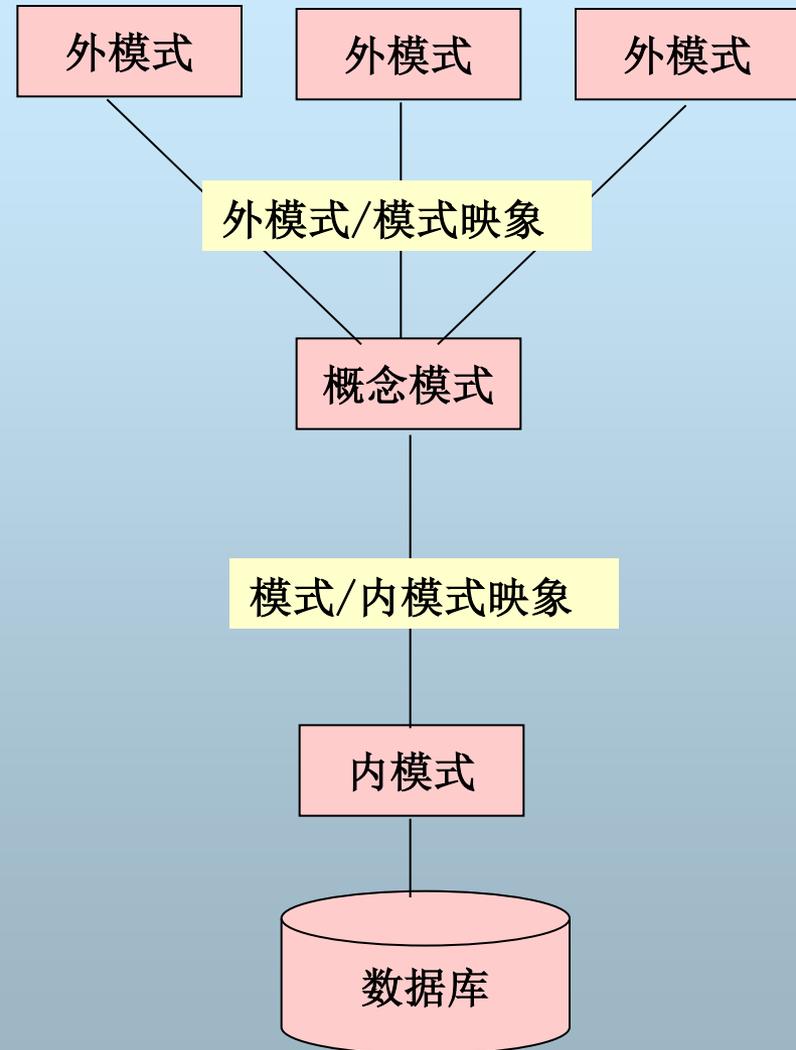
- 数据物理结构和存储方式的描述
 - 记录的存储方式：顺序存储、按B树组织还是散列存储？
 - 索引按什么方式组织：排序、散列？
 - 数据是否加密？是否压缩存储？
- 不涉及物理块（或页）的大小，也不考虑具体设备的柱面或磁道大小
- 一个数据库只有一个内模式
- **内部视图**：内模式的实例
- 通过内模式**DDL**定义

举例



4、二级映象和数据独立性

- 二级映象实现三级模式结构间的联系和转换，使用户可以逻辑地处理数据，不必关系数据的底层表示方式和存储方式



(1) 外模式/模式映象

- 定义了外模式与概念模式之间的对应关系
 - 属性名称可能不同
 - 外模式中的属性可能由模式中的多个属性运算而得
- 当概念模式发生改变时，只要修改外模式/模式映象，可保持外模式不变，从而保持用户应用程序不变，保证了数据与用户程序的逻辑独立性——数据的逻辑独立性

(2) 模式/内模式映象

- 定义了概念模式与内模式之间的对应关系
 - 概念模式中的逻辑记录和字段在内部如何表示
- 当数据库的内部存储结构发生改变时，只要修改模式/内模式映象，可保持概念模式不变，从而保持外模式以及用户程序的不变，保证了数据与程序的物理独立性——数据的物理独立性

举例

■ 外模式：EMP (EMP, DEPT, NAME)

```
Create View EMP(EMP,DEPT,NAME)  
As  
Select E# as EMP,D# as DEPT,name  
From Employee
```

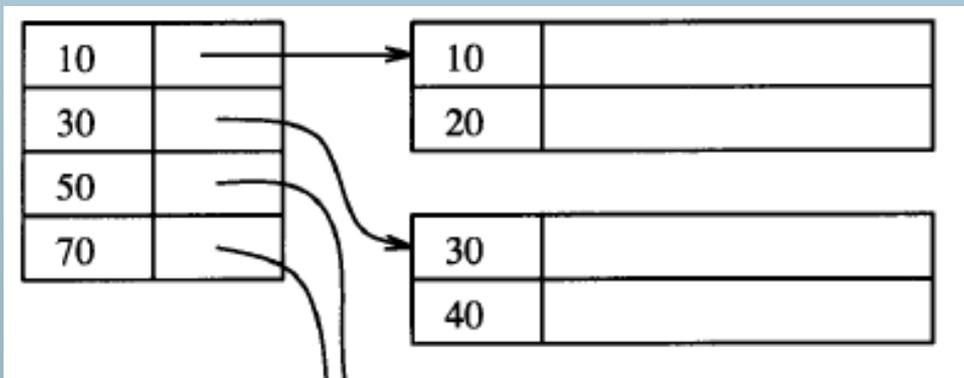
此语句定义了外模式，同时也定义了外/模映象关系

E# → EMP#

E# → EMP#

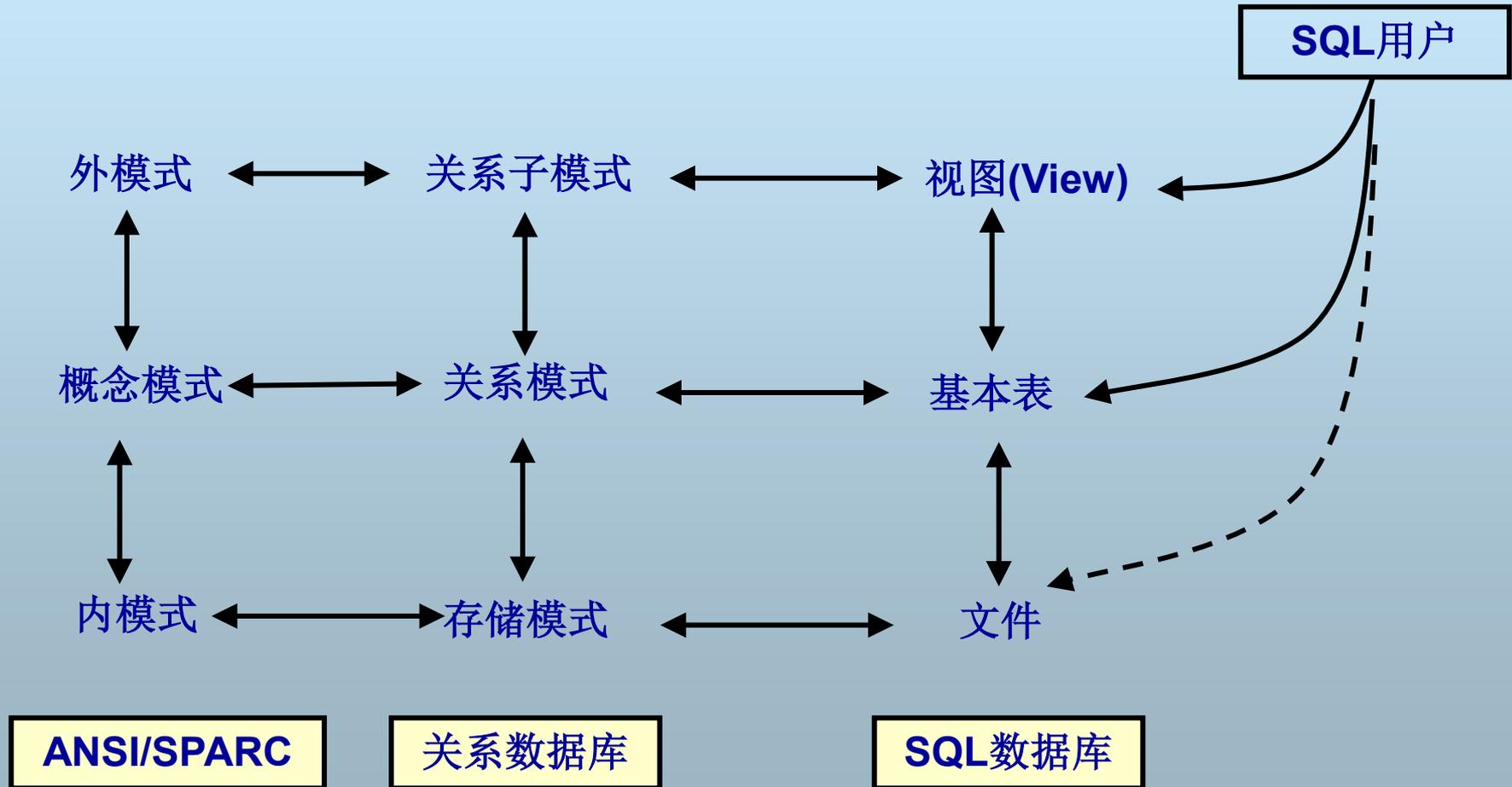
■ 模式：Employee(E#,D#,Name,Salary)

■ 内模式：顺序文件，索引文件，.....



5、SQL数据库体系结构

■ SQL数据库的三级体系结构



二、关系数据模型

- 使用数据库技术，首先必须把现实世界中的事物表示为计算机能够处理的数据
- 模型是对现实世界特征的抽象
- 数据模型是对现实世界**数据**特征的抽象
- 数据模型的定义
 - 描述现实世界实体、实体间联系以及数据语义和一致性约束的模型

1、数据模型的分类

■ 根据模型应用的不同目的

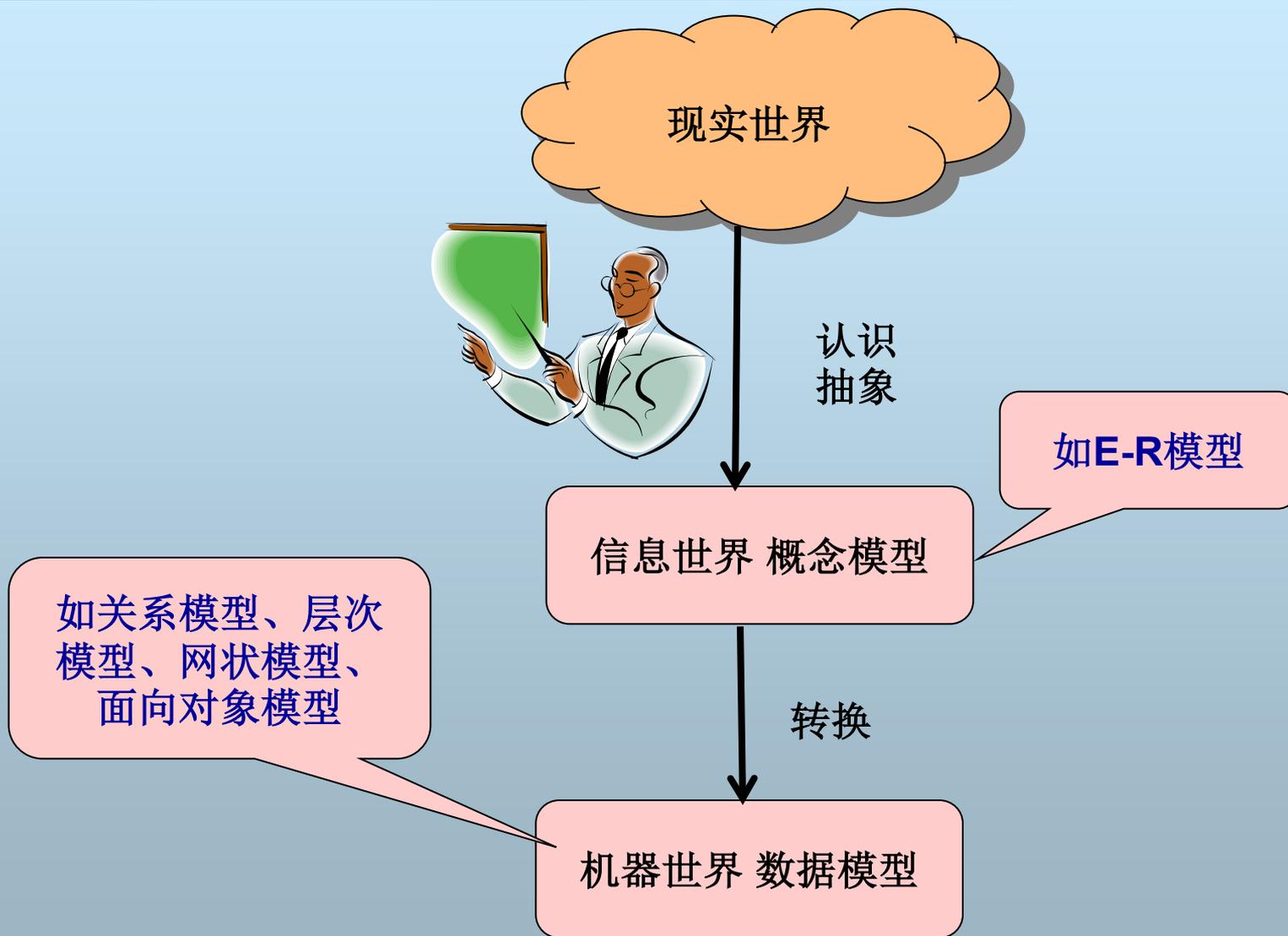
● 概念数据模型（概念模型）

- ◆ 按用户的观点对数据进行建模，强调语义表达功能
- ◆ 独立于计算机系统和DBMS
- ◆ 主要用于数据库的概念设计

● 结构数据模型（数据模型）

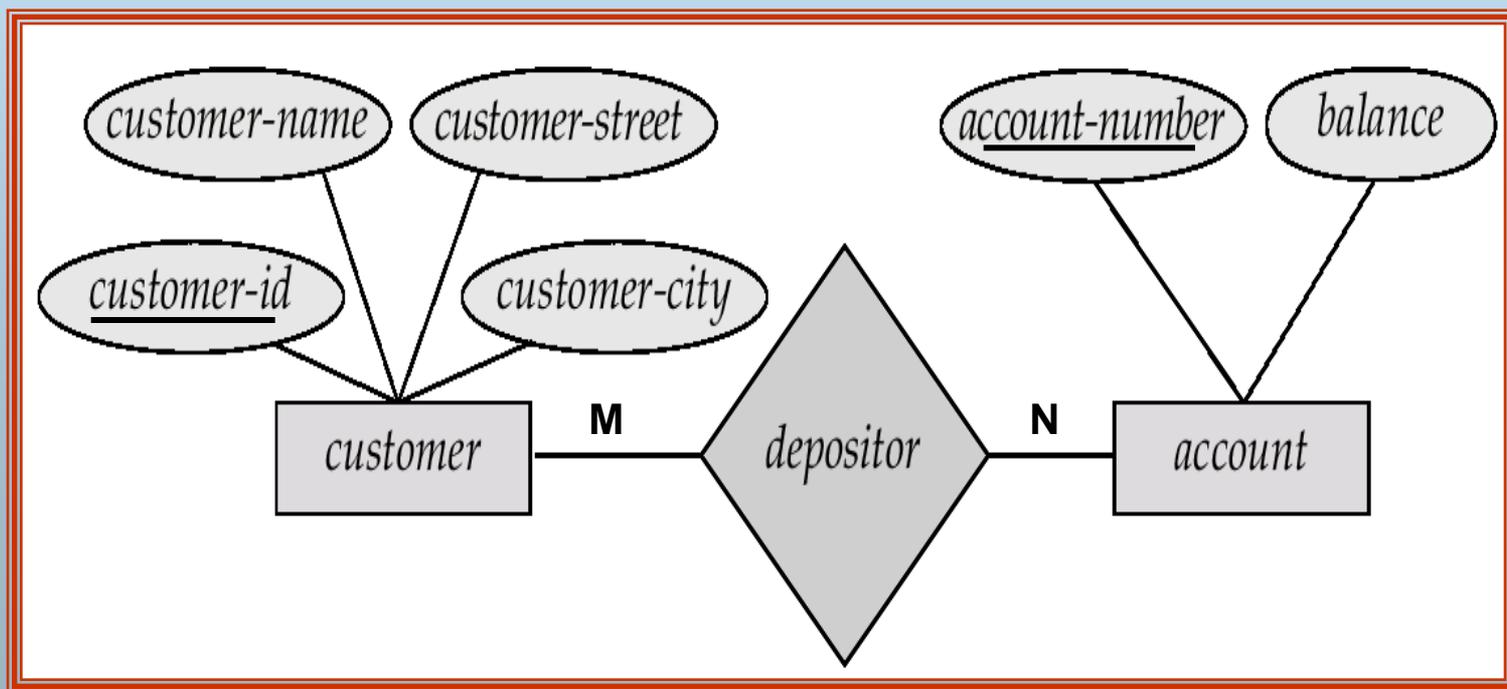
- ◆ 按计算机系统的观点对数据进行建模，直接面向数据库的逻辑结构
- ◆ 与计算机系统和DBMS相关（DBMS支持某种数据模型）
- ◆ 有严格的形式化定义，以便于在计算机系统中实现

2、数据抽象的层次



3、数据模型的例子

- 现实世界
 - 客户存款
- 信息世界
 - 概念模型（E-R模型）



3、数据模型的例子

- 机器世界
 - 数据模型 (关系模型)

<i>customer-id</i>	<i>customer-name</i>	<i>customer-street</i>	<i>customer-city</i>
192-83-7465	Johnson	12 Alma St.	Palo Alto
019-28-3746	Smith	4 North St.	Rye
677-89-9011	Hayes	3 Main St.	Harrison
182-73-6091	Turner	123 Putnam Ave.	Stamford
321-12-3123	Jones	100 Main St.	Harrison
336-66-9999	Lindsay	175 Park Ave.	Pittsfield
019-28-3746	Smith	72 North St.	Rye

(a) The *customer* table

<i>account-number</i>	<i>balance</i>
A-101	500
A-215	700
A-102	400
A-305	350
A-201	900
A-217	750
A-222	700

(b) The *account* table

<i>customer-id</i>	<i>account-number</i>
192-83-7465	A-101
192-83-7465	A-201
019-28-3746	A-215
677-89-9011	A-102
182-73-6091	A-305
321-12-3123	A-217
336-66-9999	A-222
019-28-3746	A-201

(c) The *depositor* table

4、数据模型的要素

■ 数据结构

- 现实世界实体及实体间联系的表示和实现

■ 数据操作

- 数据检索和更新的实现

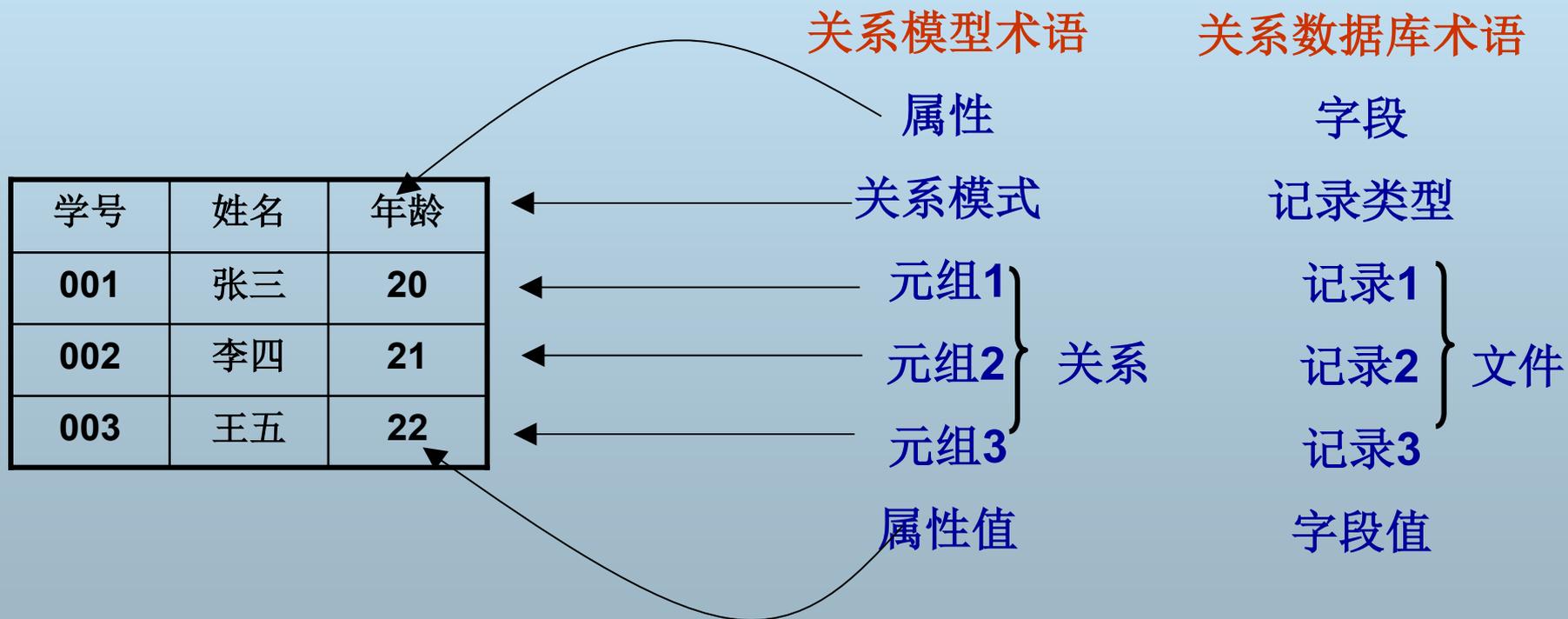
■ 数据的完整性约束

- 数据及数据间联系应具有有的制约和依赖规则

5、关系模型

■ 关系模型

- 用规范化的二维表格结构表示实体集，外码表示实体间联系，三类完整性表示数据约束的数据模型



6、几个术语

- **属性(Attribute)**: 二维表格的每一列称为关系的一个属性, 列的数目称为**度** (degree)
- **元组(Tuple)**: 每一行称为关系的一个元组, 元组的数目称为**势** (cardinality)
- **域(Domain)**: 一组具有相同数据类型的值的集合。每个属性有一个域
- **关系 (Relation)**: 元组的集合

6、几个术语

■ 超码 (Super Key)

- 在关系中能唯一标识一个元组的属性集称为关系模式的超码

■ 候选码 (Candidate Key)

- 不含多余属性的超码
- 包含在任何一个候选码中的属性称为主属性 (Primary Attribute)
- 不包含在任何一个候选码中的属性称为非主属性 (Nonprime Attribute)

■ 主码 (Primary Key)

- 用户选作元组标识的一个候选码称为主码，其余的候选码称为替换码 (Alternate Key)

6、几个术语

■ Student(Sno, Name, Age, LibraryID)

● 超码

- ◆ (sno,name)

- ◆ (libraryID, name) ...

● 候选码

- ◆ Sno

- ◆ LibraryID

● 主码

- ◆ 若选sno, 则sno为主码, libraryID为替换码

- ◆ 若选libraryID, 则libraryID为主码, sno为替换码

7、关系的性质

- 一个关系是一个规范化的二维表格
 - 属性值不可分解
 - ◆ 不允许表中有表
 - 元组不可重复
 - ◆ 因此一个关系模式至少存在一个候选码
 - 没有行序，即元组之间无序
 - ◆ 关系是元组的集合，集合的元素是无序的
 - 没有列序，即属性之间无序
 - ◆ 关系模式是属性的集合

8、关系模式与关系数据库

■ 关系模式 (Relation Schema)

- 关系的逻辑结构和特征的描述
- 对应于二维表格的表头
- 通常由属性集和各属性域表示，不关心域时可省略域
 - ◆ Student (Name, Age, Class)

■ 关系：关系模式的实例，即二维表（元组的集合）

■ 关系数据库模式 (Relational Database Schema)：关系模式的集合

■ 关系数据库：关系数据库模式的实例

9、关系模式的形式化定义

■ 关系模式可以形式化定义为：

● $R(U, D, \text{dom}, F)$

- ◆ R 为关系模式名， U 是一个属性集， D 是 U 中属性的值所来自的域， Dom 是属性向域的映射集合， F 是属性间的依赖关系

■ 例：Student关系模式的定义

● $\text{Student}(U, D, \text{dom}, F)$

- ◆ $U = \{\text{sno}, \text{name}, \text{age}\}$
- ◆ $D = \{\text{CHAR}, \text{INT}\}$
- ◆ $\text{Dom} = \{\text{dom}(\text{sno}) = \text{dom}(\text{name}) = \text{CHAR}, \text{dom}(\text{age}) = \text{INT}\}$
- ◆ $F = \{\text{sno} \rightarrow \text{name}, \text{sno} \rightarrow \text{age}\}$

■ 关系模式通常简写为 $R(U)$ ，或 $R(A_1, A_2, \dots, A_n)$

10、函数依赖

- $R(U, D, \text{dom}, F)$ 中的 F 在实际中一般只考虑函数依赖
- 函数依赖(Functional Dependency, FD)是指一个关系模式中一个属性集和另一个属性集间的多对一关系
 - 例如选课关系 $SC(S\#, C\#, \text{Score})$
 - 存在由属性集 $\{S\#, C\#\}$ 到属性集 $\{\text{Score}\}$ 的函数依赖
 - ◆ 对于任意给定的 $S\#$ 值和 $C\#$ 值, 只有一个 Score 值与其对应
 - ◆ 反过来, 可以存在多个 $S\#$ 值和 $C\#$ 值, 它们对应的 Score 值相等

10、函数依赖

- 函数依赖（FD, Functional Dependency）的形式化定义
 - 设关系模式 $R(A_1, A_2, \dots, A_n)$ 或简记为 $R(U)$, X 和 Y 是 U 的子集。 r 是 R 的任意一个实例（关系），若 r 的任意两个元组 t_1 、 t_2 , 由 $t_1[X]=t_2[X]$ 可导致 $t_1[Y]=t_2[Y]$, 即如果 X 相等则 Y 也相等, 则称 Y 函数依赖于 X 或称为 X 函数决定 Y , 记作 $X \rightarrow Y$
 - 即 R 的 X 属性集上的值可唯一决定 R 的 Y 属性集上的值
 - 也即对于 R 的任意两个元组, X 上的值相等, 则 Y 上的值也必相等
- **FD是相对于关系模式而言的, 因此关系模式 R 的所有实例都要满足FD**

10、函数依赖

■ 例如

- **Student**关系模式中， $\{S\# \} \rightarrow \{Sname\}$ （单个属性可去掉括号，简写成 $S\# \rightarrow Sname$ ）
- **SC**关系模式中， $\{S\#,C\# \} \rightarrow \{Score\}$

■ **FD**是否成立，唯一办法是仔细考察应用中属性的含义。**FD**实际上是对现实世界的断言。数据库设计者在设计时把应遵守的函数依赖通知**DBMS**，则**DBMS**会自动检查关系的合法性

- 对于关系模式 **R(Tname, Addr, C#, Cname)**
 - ◆ 若一门课只能有一个教师，则有 $\{C\# \} \rightarrow \{Tname\}$
 - ◆ 若一门课可有多个教师任教，则 $\{C\# \} \rightarrow \{Tname\}$ 不成立
 - ◆ 因此**FD**是与具体应用相关的

11、关系模型的形式化定义

■ 数据结构

- 关系：数据库中全部数据及数据间联系都以关系来表示

■ 数据操作

● 关系运算

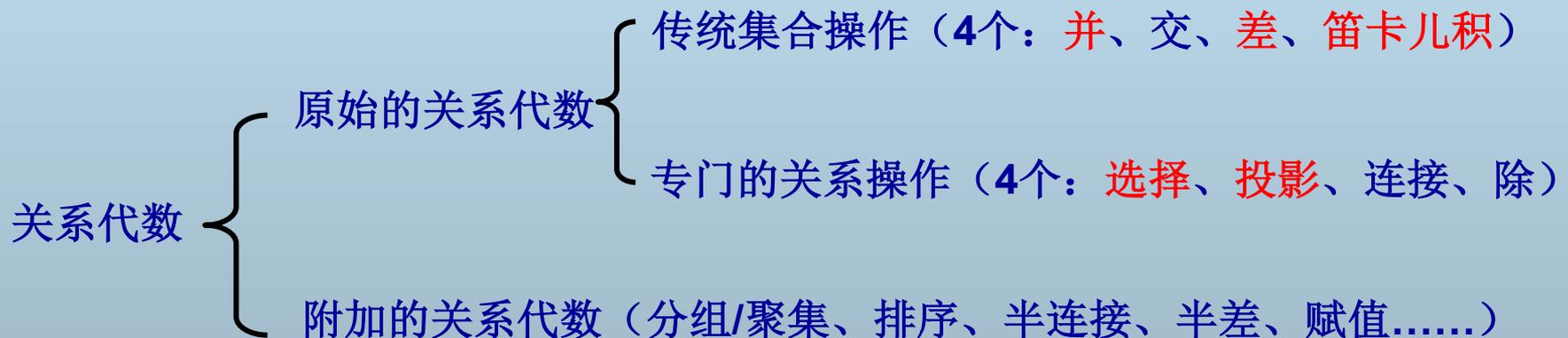
- ◆ 关系代数
- ◆ 关系演算（元组关系演算、域关系演算）

■ 数据的完整性约束

- 关系模型的三类完整性规则

11、关系代数(Relational Algebra)

- 以关系为运算对象的一组运算集合
- 运算结果仍是关系
- 以集合操作为基本运算



11、关系代数

■ 关系代数表达式的符号

● 数学符号表示

- ◆ 并 \cup 、交 \cap 、差 $-$ 、笛卡儿积 \times
- ◆ 选择 σ 、投影 π 、联接 \bowtie 、除 \div
- ◆ 重命名 $\rho_x(E)$
- ◆ 赋值 \leftarrow

● 英语关键字表示

- ◆ 并Union、交Intersect、差Minus、笛卡儿积Times
- ◆ 选择Where...、投影{All But...}、联接Join、除Devidedby

11、关系代数

- 关系代数表达式
 - 关系模型中数据操作都通过关系代数表达式来表示
- 关系代数中的基本表达式是关系代数表达式，基本表达式由如下之一构成：
 - 数据库中的一个关系
 - 一个常量关系
- 设E1和E2是关系代数表达式，则下面的都是关系代数表达式：
 - $E1 \cup E2$ 、 $E1 - E2$ 、 $E1 \times E2$
 - $\sigma_p(E1)$, 其中P是E1中属性上的谓词
 - $\pi_s(E1)$, 其中S是E1中某些属性的列表
 - $\rho_x(E1)$, 其中x是E1结果的新名字

11、关系代数

■ 关系代数操作的语义

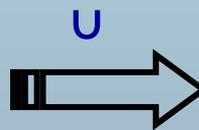
- ◆ 并 \cup 、交 \cap 、差 $-$ 、笛卡儿积 \times
- ◆ 选择 σ 、投影 π 、联接 \bowtie 、除 \div

(1) 并

■ $R \cup S = \{ t \mid t \in R \vee t \in S \}$

- t 是元组变量
- R 和 S 是关系代数表达式
- R 与 S 的degree必须相同
- R 与 S 的类型必须相同

R	学号	姓名	年龄
	001	John	20
	004	Mary	22
S	学号	姓名	年龄
	001	John	20
	002	Rose	22



学号	姓名	年龄
001	John	20
002	Rose	22
004	Mary	22

(2) 交

■ $R \cap S = \{ t \mid t \in R \wedge t \in S \}$

学号	姓名	年龄
001	John	20
004	Mary	22

R

学号	姓名	年龄
001	John	20
002	Rose	22

S

\cap

学号	姓名	年龄
001	John	20

(3) 差

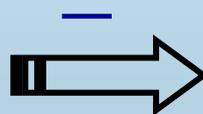
■ $R - S = \{ t \mid t \in R \wedge t \notin S \}$

R

学号	姓名	年龄
001	John	20
004	Mary	22

S

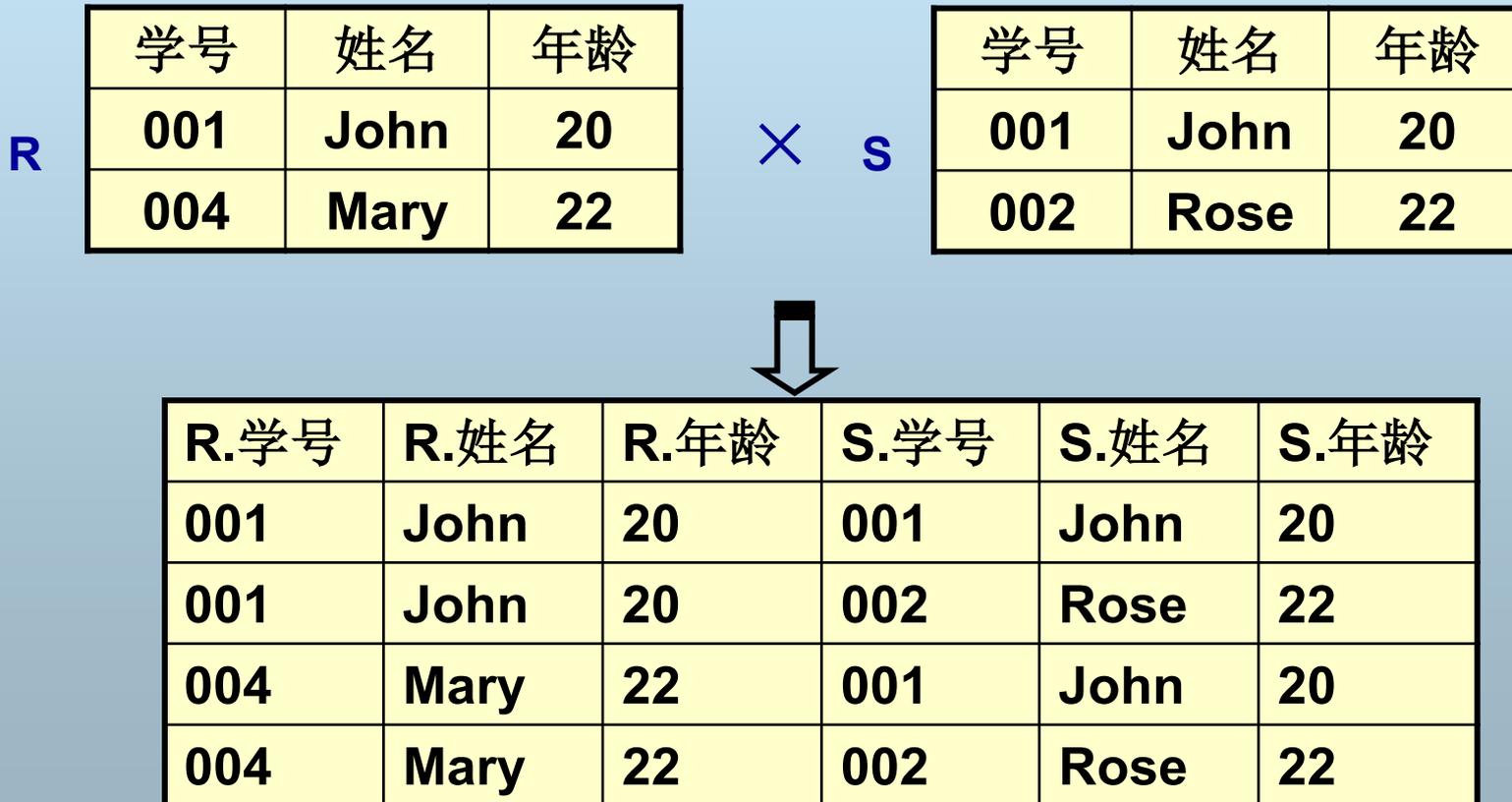
学号	姓名	年龄
001	John	20
002	Rose	22



学号	姓名	年龄
004	Mary	22

(4) 积

■ $R \times S = \{ t \mid t = \langle t^r, t^s \rangle \wedge t^r \in R \wedge t^s \in S \}$



(5) 选择

■ $\sigma_F(R) = \{ t | t \in R \wedge F(t) = \text{TRUE} \}$

- 水平划分关系

- **F**是一个逻辑表达式，表示所选的元组应满足的条件

- **F**由逻辑运算符 \neg (**NOT**)、 \wedge (**AND**)、 \vee (**OR**)连接算术表达式构成

- ◆ 算术表达式形为 $X \theta Y$ ， θ 可以是 $>$ 、 $<$ 、 $=$ 、 \leq 、 \geq 或 \neq ，**X**和**Y**可以是属性名、常量或简单函数

R

学号	姓名	年龄
001	John	20
002	Rose	22
004	Mary	22

$\sigma_{\text{年龄} > 20}(R)$

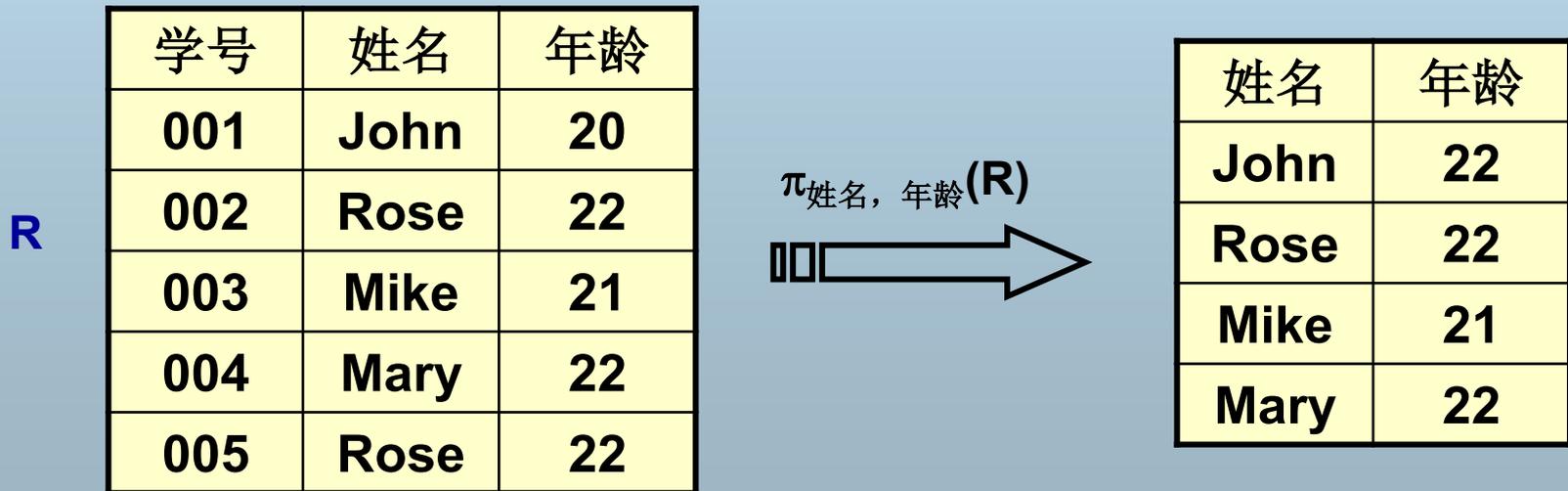


学号	姓名	年龄
002	Rose	22
004	Mary	22

(6) 投影

■ $\pi_A(R) = \{t[A] | t \in R\}$,其中A是R的属
性子集

- 垂直划分关系，选取若干列所构成的关系
- A中的属性不可重复



(7) 联接

- 自然联接 (Natural-Join)
- θ 联接 (Theta-Join)
- 等值联接 (Equi-Join)

(7) 联接：自然联接

- 设R的属性集为{X,Y}, S的属性集为{Y,Z}
- $R \bowtie S = \{t \mid t = \langle X, Y, Z \rangle \wedge t[X, Y] \in R \wedge t[Y, Z] \in S\}$
 - 相当于在 $R \times S$ 中选取R和S的所有公共属性值都相等的元组，并在结果中去掉重复属性

R

学号	姓名	年龄
001	John	20
002	Rose	22
004	Mary	22

S

学号	课程号	成绩
001	c001	80
002	c001	90

\bowtie

学号	姓名	年龄	课程号	成绩
001	John	20	c001	80
002	Rose	22	c001	90

(8) 联接: θ 联接

■ 设R的属性集为{X,Y}, S的属性集为{Y,Z}

■ $R \bowtie_{A \theta B} S = \{t \mid t = \langle t^r, t^s \rangle \wedge t^r \in R \wedge t^s \in S \wedge t^r[A] \theta t^s[B]\}$

- 相当于在 $R \times S$ 中选取R的属性A值与S的属性B值满足比较关系 θ 的元组。

R

学号	姓名	年龄
001	John	23
002	Rose	23

S

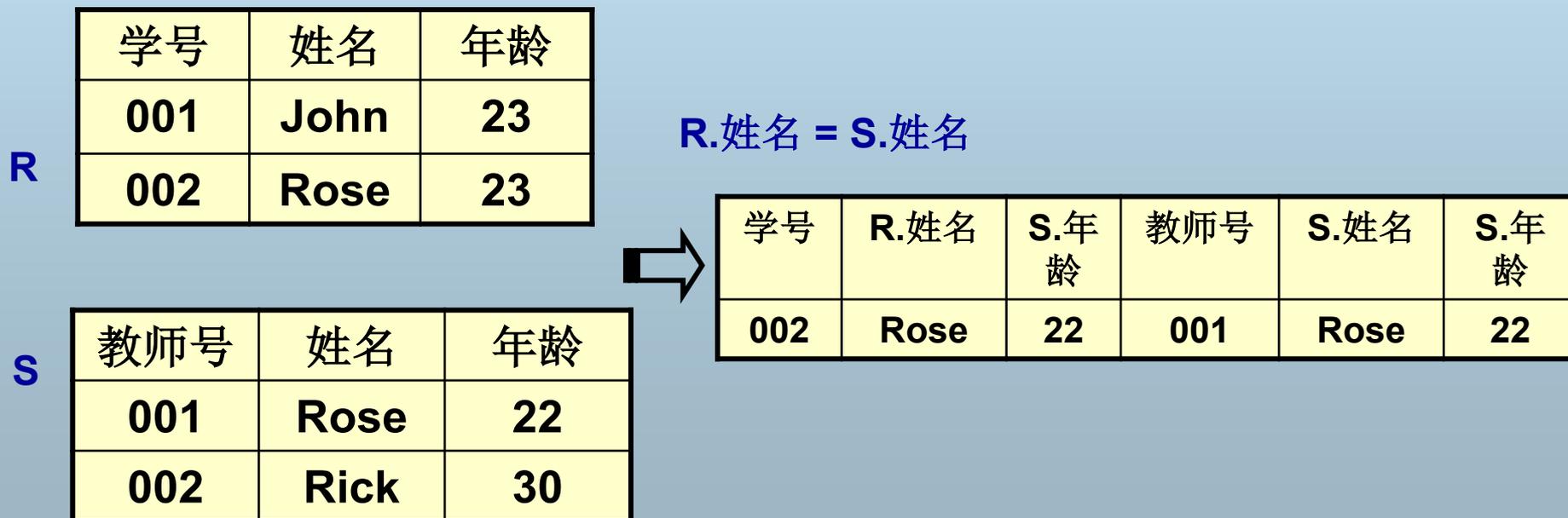
教师号	姓名	年龄
001	Bill	22
002	Rose	30

R.年龄 > S.年龄

学号	R.姓名	年龄	教师号	S.姓名	年龄
001	John	23	001	Bill	22
002	Rose	23	001	Bill	22

(9) 联接：等值联接 (equijoin)

- 在 θ 联接中 θ 为等号时，称为“等值联接”
- 等值联接是 θ 联接中比较常见的形式



11、关系代数

■ 关系代数的封闭性

- 任意关系代数操作的结果仍是一个关系

■ 关系代数的封闭性保证了关系代数操作的可嵌套性

- 例如: **(S Join P) Where City='Athens'**



举例：关系代数表达式

- 学生关系模式：S (s#, name, city)
- 求学生中的“老乡”

$$\pi_{s1\#,s2\#,S1.city} \left(\sigma_{S1.s1\# < S2.s2\#} \left(\rho_{S1(s1\#,city)}(\pi_{s\#,city}(S)) \bowtie \rho_{S2(s2\#,city)}(\pi_{s\#,city}(S)) \right) \right)$$

相应的SQL查询语句

```
SELECT s1#, s2#, S1.city
FROM (Select s# AS s1#,city From S) S1, (Select S# AS s2#,city From S) S2
WHERE S1.city = S2.city AND S1.s1# < s2.S2#
```

11、关系代数

■ 数据更新仍通过关系代数实现

■ 删除

- **$R \leftarrow R - E$** : **R**是关系, **E**是关系代数查询

- 例 “从数据库中删除姓名为 ‘Rose’的学生”

 - ◆ **$Student \leftarrow Student - \sigma_{name='Rose'}(Student)$**

11、关系代数

■ 插入

- $R \leftarrow R \cup E$: R 是关系, E 是关系代数表达式
- 如果 E 是常量关系, 则可以插入单个元组
- 例: 插入一个新的学生
 - ◆ $S1 \leftarrow S1 \cup \{ ('001', 'Rose', 19) \}$

11、关系代数

■ 修改

- $R \leftarrow \pi_{F_1, F_2, \dots, F_n}(R)$: 通过广义投影实现
- F_i : 当第*i*个属性没有被修改时是*R*的第*i*个属性；当被修改时是第*i*个属性和一个常量的表达式
- 例1: “将每个学生的学号前加上字母S”
 - ◆ $Student \leftarrow \pi_{S' || sno, name, sex, age}(Student)$
- 如果只想修改*R*中的部分元组，可以用下式
- $R \leftarrow \pi_{F_1, F_2, \dots, F_n}(\sigma_p(R)) \cup (R - \sigma_p(R))$
- 例2: “将所有男学生的学号前加上字母M”
 - ◆ $Student \leftarrow \pi_{M' || sno, name, sex, age}(\sigma_{sex='M'}(Student)) \cup (Student - \sigma_{sex='M'}(Student))$

11、关系模型的形式化定义

■ 数据结构

- 关系：数据库中全部数据及数据间联系都以关系来表示

■ 数据操作

● 关系运算

- ◆ 关系代数
- ◆ 关系演算（元组关系演算、域关系演算）

■ 数据的完整性约束

- 关系模型的三类完整性规则

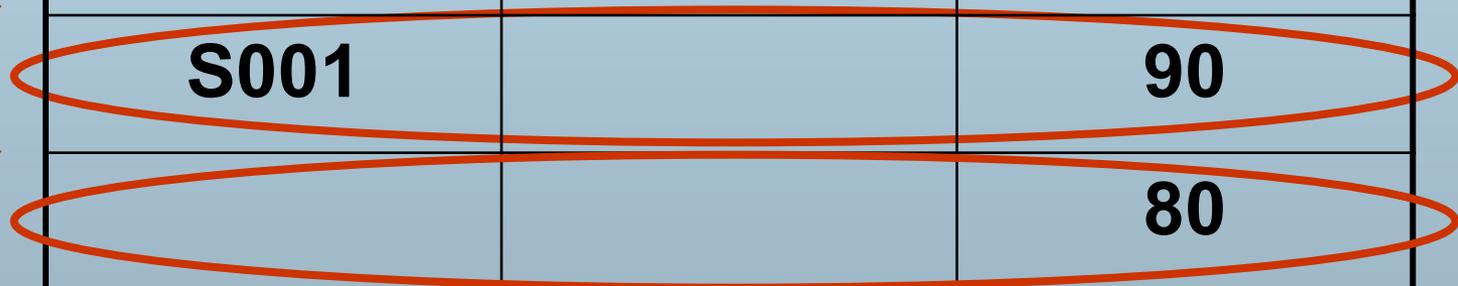
12、关系模型的三类完整性规则

- 关系数据库的数据和操作必须遵循的规则
 - 实体完整性（Entity Integrity）
 - 参照完整性（Referential Integrity）
 - 用户自定义完整性（User-Defined Integrity）

(1) 实体完整性

- 关系模式R的主码不可取空值

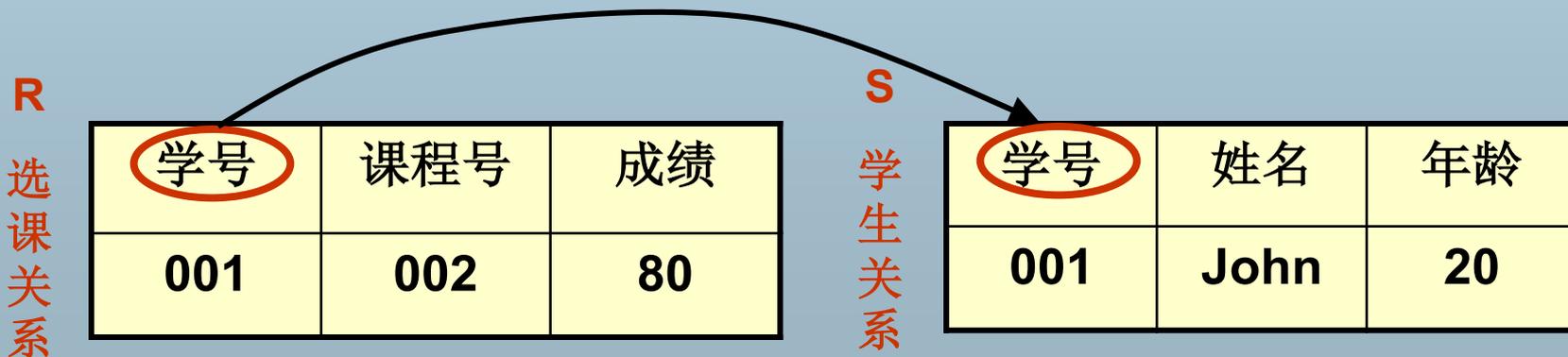
<u>学号</u>	<u>课程号</u>	成绩
S001	C001	80
S001		90
		80



(2) 参照完整性

■ 外码 (Foreign Key)

- 关系模式R的外码是它的一个属性集FK，满足：
 - ◆ 存在带有候选码CK的关系模式S，且
 - ◆ R的任一非空FK值都在S的CK中有一个相同的值
- S称为被参照关系 (Referenced Relation)，R称为参照关系 (Referential Relation)



(2) 参照完整性

- 参照关系R的任一个外码值必须
 - 等于被参照关系S中所参照的候选码的某个值
 - 或者为空

R
选课关系

学号	课程号	成绩
001	002	80

~~002~~

S
学生关系

学号	姓名	专业号
001	John	

专业号	专业名	学科类别
0020	PHY	1

(3) 用户自定义完整性

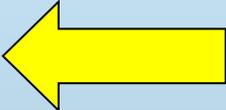
- 针对某一具体数据的约束条件，反映某一具体应用所涉及的数据必须满足的特殊语义
- 由应用环境决定

学号	课程号	成绩
001	002	80

成绩 ≥ 0 and 成绩 ≤ 100



Where are we?

- 数据库体系结构 (Database Architecture)
- 关系数据模型 (Relational Data Model)
- SQL 

三、SQL

■ 数据库语言

- 数据库语言是用户使用数据库的接口，**DBMS**支持用户通过数据库语言进行数据存取与管理

■ 有三类数据库语言

- 数据定义语言(Data Definition Language, **DDL**)
- 数据操纵语言(Data Manipulation Language, **DML**)
- 数据控制语言(Data Control Language, **DCL**)

1、SQL的基本组成



2、SQL的发展历程

- **1972: IBM开始研究System R系统, 配置了数据库语言SQUARE**
 - **SQUARE (Specifying Queries As Relational Expressions)**
 - 使用了大量的数学符号
- **1974: Boyce和Chamberlin将SQUARE修改为SEQUEL**
 - **SEQUEL (Structured English QUery Language)**
 - 去掉了数学符号, 以英语单词和结构式语法代替查询
 - 后简称为**SQL (Structured Query Language)**

2、SQL的发展历程

- **1970s末起**：主流的数据库厂商纷纷在其产品中支持SQL
 - **Oracle、DB2**
- **1986.10**：ANSI颁布了美国标准的SQL
- **1987.4**：ISO采纳美国标准为国际标准，后称“SQL86”
- **1989.4**：SQL89，增强了完整性特征
- **1992**：SQL92（“SQL2”）
- **1999**：SQL3

3、SQL与关系模型

■ 数据结构

- **SQL: Table**
- **关系模型: 关系**

■ 数据操作

- **SQL: DML**
- **关系模型: 关系代数**

■ 数据约束

- **SQL: Primary Key、Foreign Key、Unique、Check**
- **关系模型: 实体完整性、参照完整性、用户自定义完整性**

本章小结

- **数据库体系结构 (Database Architecture)**
 - 视图、基本表、文件
- **关系数据模型 (Relational Data Model)**
 - 数据结构：关系
 - 数据操作：关系代数/关系演算
 - 数据约束：三类完整性约束
- **SQL**