

数据库系统原理

陈岭

浙江大学计算机学院

2

关系模型

- 关系和关系模式
- 关系代数
- 扩展的关系代数运算
- 数据库的修改

什么是关系模型？

- ❑ 关系数据库基于关系模型，是一个或多个关系组成的集合
- ❑ 关系通俗来讲就是表（由行和列构成）
- ❑ 关系模型的主要优点是其简单的数据表示，易于表示复杂的查询
- ❑ SQL语言是最广泛使用的语言，用于创建，操纵和查询关系数据库，而关系模型是其基础。

关系示例

Instructor:

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

联系：一些实体之间的关联

关系：是一种数学概念，指的是表
实体集和联系集 \leftrightarrow 真实的世界

关系 - 表，元组 - 行 \leftrightarrow 机器的世界

关系基本结构

- 一般地, 给出集合 $D_1, D_2, \dots, D_n, (D_i = a_{ij} \mid j=1 \dots k)$
 - 关系 r 是: $D_1 \times D_2 \times \dots \times D_n$ 的子集, 即一系列 D_i 域的笛卡尔积
- 因而关系是一组 n 元组 $(a_{1j}, a_{2j}, \dots, a_{nj})$ 的集合, 其中每个 $a_{ij} \in D_i$
- 例如:

{	张清玫教授, 计算机, 李勇
{	张清玫教授, 计算机, 刘晨
{	刘逸教授, 信息, 王名

关系: 导师-专业-学生

笛卡尔积示例

例如: D_1 = 导师集合 = {张清玫, 刘逸},
 D_2 = 专业集合 = {计算机, 信息},
 D_3 = 研究生集合 = {李勇, 刘晨, 王名}
则 $D_1 \times D_2 \times D_3 = \{$ (张清玫, 计算机, 李勇),
(张清玫, 计算机, 刘晨),
(张清玫, 计算机, 王名),
(张清玫, 信 息, 李勇),
(张清玫, 信 息, 刘晨),
(张清玫, 信 息, 王名),
(刘 逸, 计算机, 李勇),
(刘 逸, 计算机, 刘晨),
(刘 逸, 计算机, 王名),
(刘 逸, 信 息, 李勇),
(刘 逸, 信 息, 刘晨),
(刘 逸, 信 息, 王名) $\}$

共12个元组

D1	D2	D3
张清玫	计算机	李勇
张清玫	计算机	刘晨
张清玫	计算机	王名
张清玫	信 息	李勇
张清玫	信 息	刘晨
张清玫	信 息	王名
刘逸	计算机	李勇
刘逸	计算机	刘晨
刘逸	计算机	王名
刘逸	信 息	李勇
刘逸	信 息	刘晨
刘逸	信 息	王名

笛卡儿积可用一张二维表表示

导师-专业-学生

张清玫	计算机	李勇
张清玫	计算机	刘晨
刘逸	信息	王名



笛卡尔积示例

dept_name = {Biology, Finance, History, Music}

building = {Watson, Painter, Packard}

budget = {50000, 80000, 90000, 120000}

那么, *r* = { (Biology, Watson, 90000),
(Finance, Painter, 120000),
(History, Painter, 50000),
(Music, Packard, 80000)
}

是 *dept_name* *x* *building* *x* *budget* 三者间的关系。(共48个元组)

- ❑ 关系的每个属性都有一个名称
- ❑ 域：每个属性的取值集合称为属性的域
- ❑ 属性值必须是原子的，即不可分割(1NF, 第一范式)
 - 多值属性值不是原子的
 - 复合属性值不是原子的
- ❑ 特殊值null是每一个域的成员
- ❑ 空值给数据库访问和更新带来很多困难，因此应尽量避免使用空值
 - 我们先假设不存在空值，在后面的章节中，再讲解空值对不同操作的影响

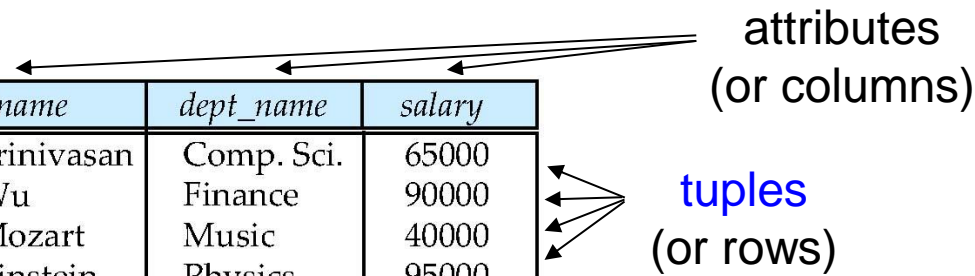
关系的概念

- ❑ 关系涉及两个概念：关系模式和关系实例
- ❑ 关系模式描述关系的结构：
 - 例, *Instructor-schema* = (*ID*: string, *name*: string, *dept_name*: string, *salary*: int)
 - 或 *Instructor-schema* = (*ID*, *name*, *dept_name*, *salary*)
- ❑ 关系实例表示一个关系的特定实例，也就是所包含的一组特定的行
- ❑ 关系、关系模式、关系实例区别：
 - 变量 \leftrightarrow 关系
 - 变量类型 \leftrightarrow 关系模式
 - 变量值 \leftrightarrow 关系实例

- A_1, A_2, \dots, A_n 是属性
- 一般地: $R = (A_1, A_2, \dots, A_n)$ 是一个关系模式
 - 例, *Instructor-schema* = (*ID*, *name*, *dept_name*, *salary*)
- $r(R)$ 是在关系模式 R 上的关系
 - 例, *instructor(Instructor-schema)* = *instructor(ID, name, dept_name, salary)*

关系实例

- ❑ 关系的当前值（关系实例）由表指定
- ❑ 一个元组 t 代表表中的一行
- ❑ 如果元组变量 t 代表一个元组，那么 $t[name]$ 表示属性 $name$ 的 t 的值



<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

关系的无序性

□ 元组的顺序性是无关紧要的（元组能够以任意顺序存储），但一个关系中不能有重复的元组

■ 例，*department*(*dep_name*, *building*, *budget*) 关系的无序显示

<i>dept_name</i>	<i>building</i>	<i>budget</i>
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

- 使 $K \subseteq R$
- 如果K值能够在一个关系中唯一地标志一个元组，则K是R的**超码**
 - 例， $\{instructor-ID, instructor-name\}$ 和 $\{instructor-ID\}$ 都是 *instructor* 的超键
- 如果K是最小超码，则K是**候选码**
 - 例， $\{instructor-ID\}$ 是 *instructor* 的候选码。因为它是一个超码，并且它的任意真子集都不能成为一个超码
- 如果k是一个候选码，并由用户明确定义，则K是一个**主键**。主键通常用下划线标记

外键

□ 假设存在关系r和s: $r(A, B, C)$, $s(B, D)$, 则在关系r上的属性B称作参照s的外码, r也称为外码依赖的参照关系, s叫做外码被参照关系

- 例, 学生(学号, 姓名, 性别, 专业号, 年龄) - 参照关系
专业(专业号, 专业名称) - 被参照关系 (目标关系)
其中属性 专业号 称为关系学生的外码

选修(学号, 课程号, 成绩)

课程(课程号, 课程名, 学分, 先修课号)

- $instructor(ID, name, dept_name, salary)$ - 参照关系
 $department(dept_name, building, budget)$ - 被参照关系

参照关系中外码的值必须在被参照关系中实际存在或为null

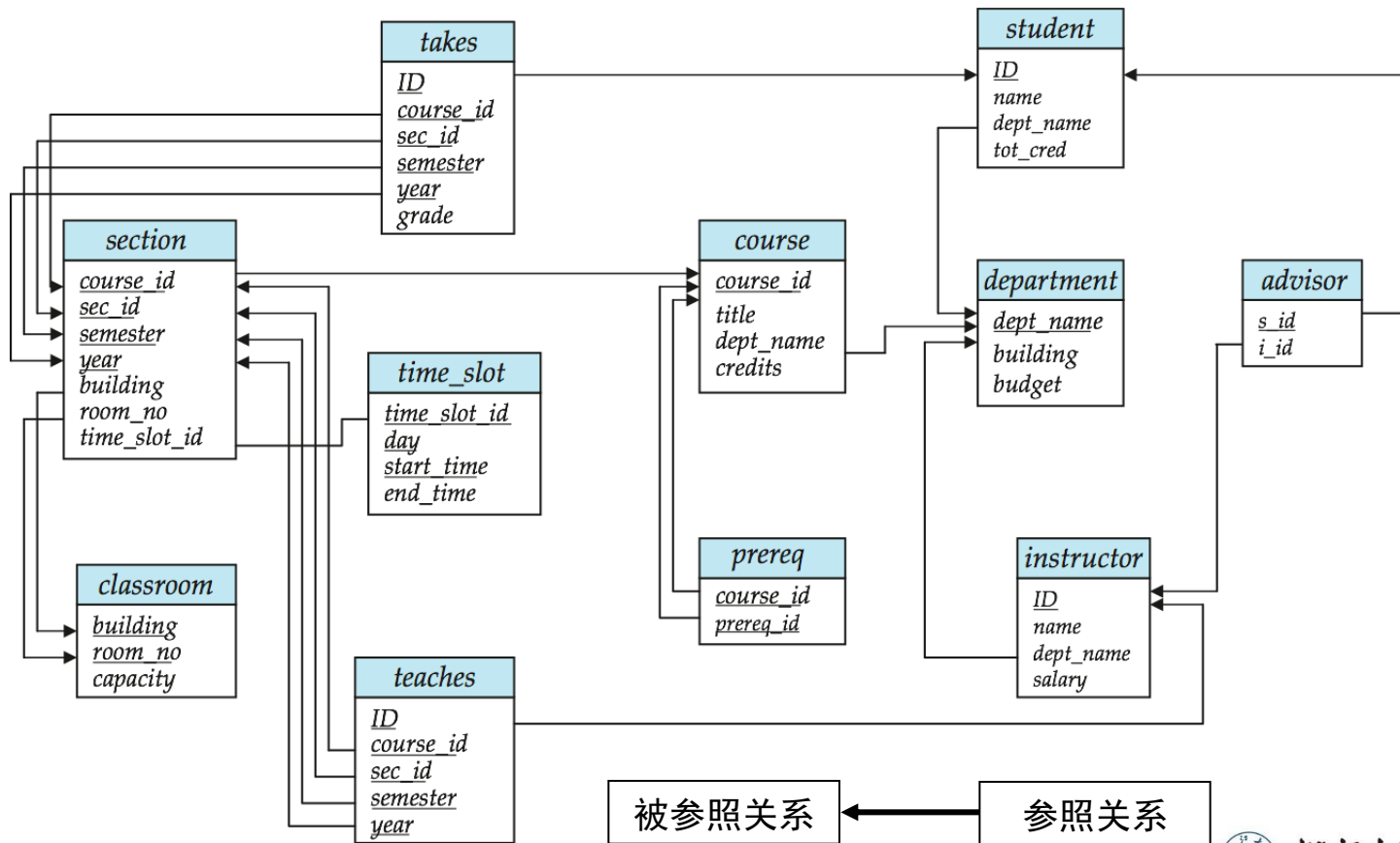


大学数据库模式

- ❑ *classroom*(building, room_number, capacity)
- ❑ *department*(dept_name, building, budget)
- ❑ *course*(course_id, title, dept_name, credits)
- ❑ *instructor*(ID, name, dept_name, salary)
- ❑ *section*(course_id, sec_id, semester, year, building, room_number, time_slot_id)
- ❑ *teaches*(ID, course_id, sec_id, semester, year)
- ❑ *student*(ID, name, dept_name, tot_cred)
- ❑ *takes*(ID, course_id, sec_id, semester, year, grade)
- ❑ *advisor*(s_ID, i_ID)
- ❑ *time_slot*(time_slot_id, day, start_time, end_time)
- ❑ *prereq*(course_id, prereq_id)



大学数据库模式图



- 查询语言：用户用来从数据库中请求获取信息的语言
- “纯” 查询语言：
 - 关系代数 - SQL的基础
 - 元组关系演算
 - 域关系演算
- “纯” 查询语言奠定了人们使用查询语言的基础，如SQL

- 在某种程度上是过程化语言
- 六个基本运算
 - Select 选择
 - Project 投影
 - Union 并
 - set difference 差（集合差）
 - Cartesian product 笛卡儿积
 - Rename 更名（重命名）
- 用户输入一个或两个关系，并得到新的关系

□ 附加运算

- Set intersection 交
- Natural join 自然连接
- Division 除
- Assignment 赋值

选择运算

□ 例：关系r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

注：执行选择时，选择条件必须是对同一元组中的相应属性值代入进行比较。

□ 选择元组，满足： $A = B$
and $D > 5$

■ $\sigma_{A=B \text{ and } D > 5}(r)$

A	B	C	D
α	α	1	7
β	β	23	10

选择运算

- ❑ 定义: $\sigma_p(r) = \{t \mid t \in r \text{ and } p(t)\}$, σ 读作sigma, 其中p为选择谓词
- ❑ 其中p是由逻辑连词 \wedge (与), \vee (或), \neg (非)连接起来的公式。逻辑连词的运算对象可以是包含比较运算符 $<$ 、 $<=$ 、 $>$ 、 $>=$ 、 $=$ 和 $><$ 的表达式
- ❑ 例如: $\sigma_{dept_name = 'Finance'}(department)$

<i>dept_name</i>	<i>building</i>	<i>budget</i>
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

投影运算

□ 例：关系r

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

□ 选择属性 A、C

■ 投影： $\Pi_{A,C}(r)$

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

□ 定义:

- $\Pi_{A_1, A_2, \dots, A_k}(r)$, Π 读作pi, A_1, \dots, A_k 是属性名, r 为关系名
- 其结果为保留此k列的值, 并删除重复的行

□ 例如: $\Pi_{building}(department)$

<i>dept_name</i>	<i>building</i>	<i>budget</i>
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000



<i>building</i>
Watson
Taylor
Painter
Packard

并运算

□ 例：关系 r

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

□ 并运算

■ $r \cup s$:

A	B
α	1
α	2
β	1
β	3

并运算

- 定义: $r \cup s = \{t \mid t \in r \text{ or } t \in s\}$
- $r \cup s$ 条件:
 - 等目, 同元, 即他们的属性数目必须相同
 - 对任意 i , r 的第 i 个属性域和 s 的第 i 个属性域相同
- 例如: $\Pi_{name}(\text{instructor}) \cup \Pi_{name}(\text{student})$

差运算

□ 例：关系 r , s

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

□ 差运算

■ $r - s$:

A	B
α	1
β	1

差运算

□ 定义: $r - s = \{t \mid t \in r \text{ and } t \notin s\}$

□ $r - s$ 条件:

- 等目, 同元, 即他们的属性数目必须相同
- 对任意 i , r 的第 i 个属性域和 s 的第 i 个属性域相同



广义笛卡尔积

□ 例：关系 r , s

A	B
-----	-----

α	1
β	2

r

C	D	E
-----	-----	-----

α	10	a
β	10	a
β	20	b
γ	10	b

s

□ 广义笛卡尔积

■ $r \times s$:

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

广义笛卡尔积

- 定义: $r \times s = \{ \{t \ q\} \mid t \in r \text{ and } q \in s \}$
- 假设 $r(R)$ 的属性和 $s(S)$ 的属性没有交集
- 如果 $r(R)$ 和 $s(S)$ 的属性有交集, 那么必须重命名这些有交集的属性

广义笛卡尔积

□ 例：关系 r , s

A	B
α	1
α	2
β	1

r

B	C
k	2
d	3

s

$r \times s =$

A	$r.B$	$s.B$	C
α	1	k	2
α	2	k	2
β	1	k	2
α	1	d	3
α	2	d	3
β	1	d	3

复合运算

□ 可以使用多种运算符构建表达式

□ 例: $\sigma_{A=C}(r \times s)$

A	B
---	---

α	1
β	2

r

C	D	E
---	---	---

α	10	a
β	10	a
β	20	b
γ	10	b

s

$r \times s =$

A	B	C	D	E
---	---	---	---	---

α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

$\sigma_{A=C}(r \times s) =$

A	B	C	D	E
---	---	---	---	---

α	1	α	10	a
β	2	β	20	a
β	2	β	20	b

更名运算

- 允许我们使用其他名字指代关系
- 例： $\rho_x(E)$ ， ρ 读作rho， 返回表达式E的结果， 并把名字X赋给了它。 假设关系代数表达式E是n元的， 则表达式

$\rho_x(A_1, A_2, \dots, A_n)(E)$ (对关系E及其属性都重命名)

返回表达式E的结果， 并赋给它名字X， 同时将属性重命名为A1, A2, ..., An.

银行示例

- ❑ *branch* (*branch-name*, *branch-city*, *assets*)
- ❑ *customer* (*customer-name*, *customer-street*, *customer-city*)
- ❑ *account* (*account-number*, *branch-name*, *balance*)
- ❑ *loan* (*loan-number*, *branch-name*, *amount*)
- ❑ *depositor* (*customer-name*, *account-number*)
- ❑ *borrower* (*customer-name*, *loan-number*)

查询示例

- 例1：找出贷款额大于\$1200的元组

$$\sigma_{amount > 1200} (loan)$$

- 例2：找出贷款大于\$1200的贷款号

$$\Pi_{loan-number} (\sigma_{amount > 1200} (loan))$$

loan (*loan-number*, *branch-name*, *amount*)

- 例3：找出有贷款或有帐户或两者兼有的所有客户姓名

$$\Pi_{customer-name} (borrower) \cup \Pi_{customer-name} (depositor)$$

- 例4：找出至少有一个贷款及一个账户的客户姓名

$$\Pi_{customer-name} (borrower) \cap \Pi_{customer-name} (depositor)$$

depositor (customer-name, account-number)

borrower (customer-name, loan-number)



□ 例5：找出在Perryridge 分支机构有贷款的顾客姓名

查询1: $\Pi_{customer-name} (\sigma_{branch-name = "Perryridge"} (\sigma_{borrower.loan-number = loan.loan-number} (borrower \times loan)))$

查询2: $\Pi_{customer-name} (\sigma_{borrower.loan-number = loan.loan-number} (borrower \times (\sigma_{branch-name = "Perryridge"} (loan))))$

查询2更好

$loan (\underline{loan-number}, branch-name, amount)$
 $borrower (\underline{customer-name}, \underline{loan-number})$

- 例6：找出在Perryridge分支机构有贷款，但在其他分支机构没有账号的顾客姓名

查询1: $\Pi_{customer-name} (\sigma_{branch-name = "Perryridge"} (\sigma_{borrower.loan-number = loan.loan-number} (borrower \times loan)))$
- $\Pi_{customer-name} (depositor)$

查询2: $\Pi_{customer-name} (\sigma_{borrower.loan-number = loan.loan-number} (borrower \times (\sigma_{branch-name = "Perryridge"} (loan))))$ - $\Pi_{customer-name} (depositor)$

□ 例7：找出银行中最大的账户余额

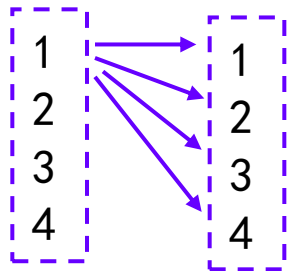
■ 将 $account$ 关系重命名为 d

■ 第一步：找出由非最大余额构成的临时关系

$$\Pi_{account.balance} (\sigma_{account.balance < d.balance} (account \times \rho_d (account)))$$

■ 第二步：找出最大余额

$$\Pi_{balance} (account) - \Pi_{account.balance} (\sigma_{account.balance < d.balance} (account \times \rho_d (account)))$$



d



- 定义一些附加运算，它们虽不能增加关系代数的表达能力，但却可以简化一些常用的查询
 - 集合交
 - 自然连接
 - 除
 - 赋值

交运算

□ 例：关系 r

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

□ 交运算

■ $r \cap s$:

A	B
α	2

□ 定义: $r \cap s = \{ t \mid t \in r \text{ and } t \in s \}$

□ 假设:

- r 和 s 同元

- r 和 s 的属性域是可兼容的

□ $r \cap s = r - (r - s)$

自然连接

□ $r \bowtie s$

□ 例: $R = (A, B, C, D)$ $S = (E, B, D)$

■ 关系 r 和 s 自然连接的结果模式为: (A, B, C, D, E)

■ $r \bowtie s = \Pi_{r.A, r.B, r.C, r.D, s.E} (\sigma_{r.B = s.B \wedge r.D = s.D} (r \times s))$

□ 设关系 r 和 s 分别代表模式 R 和 S , 那么 $r \bowtie s$ 是对模式 $R \cup S$ 运算后的关系表示:

■ 考虑 r 的每一个元组 t_r , s 的每一个元组 t_s

■ 如果 t_r 和 t_s 在 $R \cap S$ 的公共属性下有相同的值, 那么向结果集中插入元组 t :

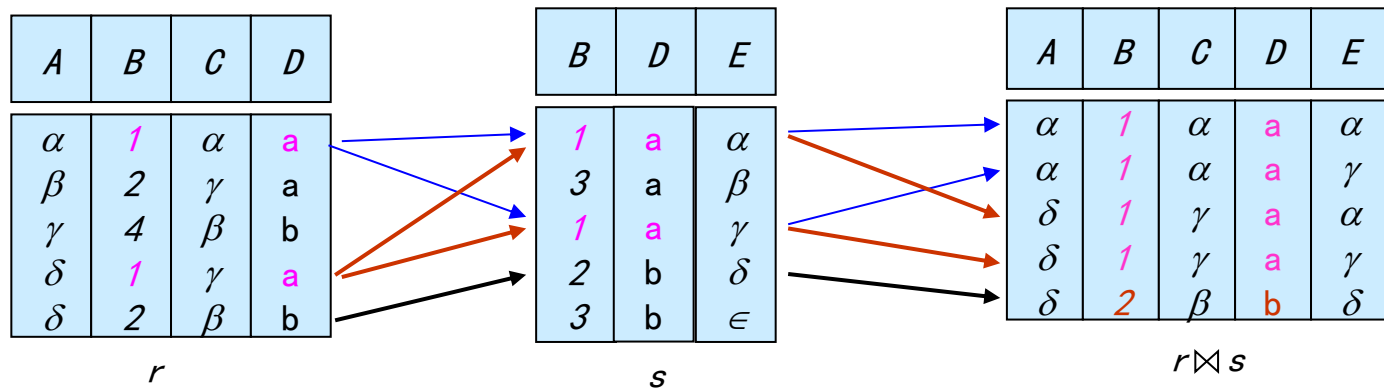
- 元组 t 与 t_r 有相同的值

- 元组 t 与 t_s 有相同的值



自然连接

□ 例：关系 r , s



注：

- (1) r, s 必须含有**共同属性**（名，域对应相同），
- (2) 连接二个关系中同名属性值相等的元组
- (3) 结果属性是二者属性集的并集，但消去重名属性。

theta连接

- theta连接: $r \bowtie_{\theta} s = \sigma_{\theta}(r \times s)$
 - θ 是模式 $R \cup S$ 属性上的谓词
- theta连接是自然连接的扩展

除运算

- $r \div s$ 适用于包含了“对所有的”此类短语的查询
- 例: 查询选修了所有课程的学生学号

<i>enrolled</i>	Sno	Cno	Grade
	95001	1	92
	95001	2	85
	95001	3	88
	95002	2	90
	95002	3	80

\div

<i>course</i>	
Cno	
1	
2	
3	

$=$

Sno
95001

$$\Pi_{Sno, Cno} (enrolled) \div \Pi_{Cno} (course)$$

□ 设 r 和 s 分别代表模式 R 和 S 的关系

■ $R = (A_1, \dots, A_m, B_1, \dots, B_n)$

■ $S = (B_1, \dots, B_n)$

■ $r \div s$ 的结果代表模式 $R - S$ 的关系:

- $R - S = (A_1, \dots, A_m)$

- $r \div s = \{t \mid t \in \Pi_{R-S}(r) \wedge \forall u \in s (tu \in r)\}$

注：商来自于 $\Pi_{R-S}(r)$ ，并且其元组 t 与 s 所有元组的拼接被 r 覆盖

除运算

□ 例：关系 r , s

A	B
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
ϵ	6
ϵ	1
β	2

r

B
1
2

s

$$r \div s =$$

A
α
β

$$(r \div s = \{t \mid t \in \Pi_{R-S}(r) \wedge [\forall u \in s (tu \in r)]\})$$



除运算

□ 例，关系 r , s

A	B	C	D	E
α	a	α	a	1
α	b	γ	a	1
α	b	γ	b	1
β	a	γ	a	1
γ	a	γ	c	2
β	a	γ	b	3
γ	a	γ	a	1
γ	a	γ	b	1
γ	c	β	b	1

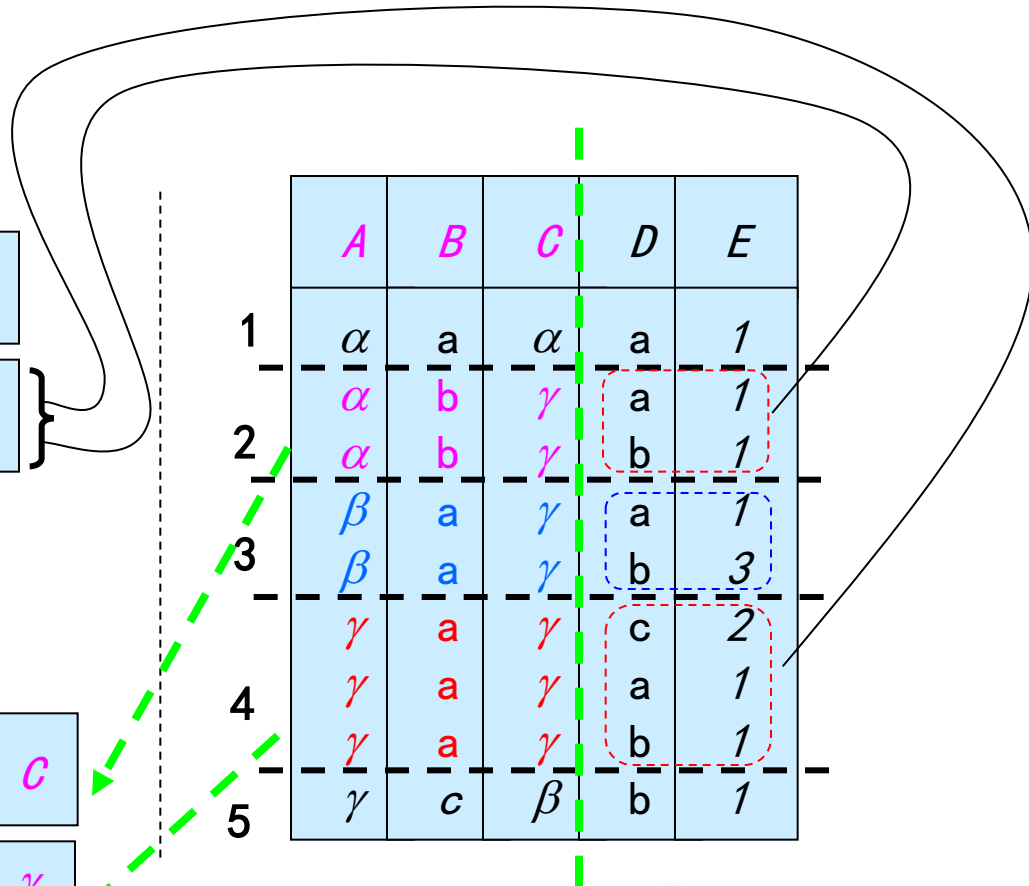
r

D	E
a	1
b	1

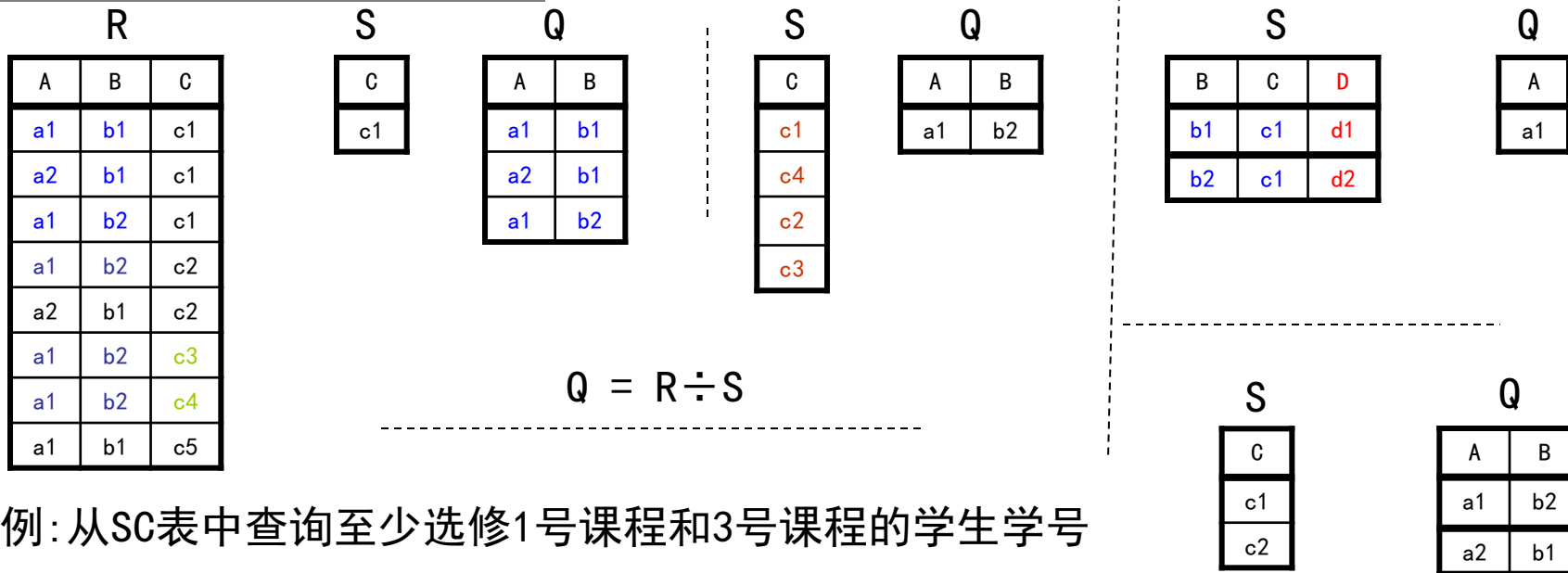
s

A	B	C
α	b	γ
γ	a	γ

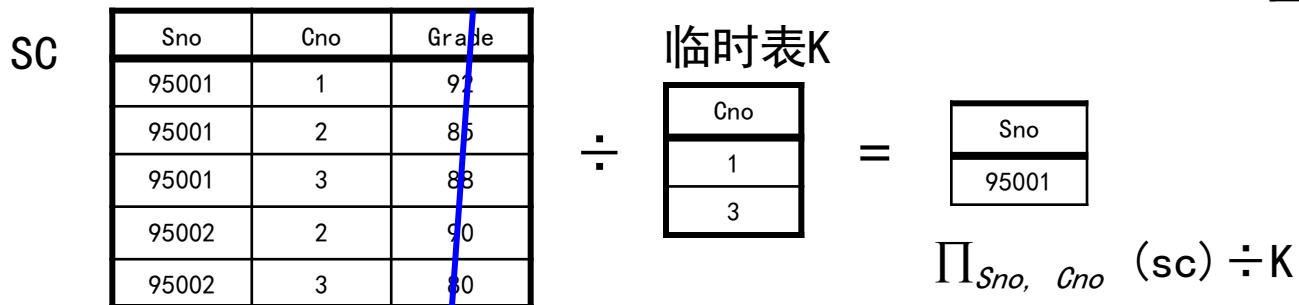
$$r \div s =$$



除运算



例: 从SC表中查询至少选修1号课程和3号课程的学生学号



□ 性质

- 若 $q = r \div s$, 则 q 是满足 $q \times s \subseteq r$ 的最大关系

□ 用基本代数运算来定义除运算

- 设 $r(R)$ 和 $s(S)$ 已知, 且 $S \subseteq R$:

$$r \div s = \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$$

- 为什么:

- $\Pi_{R-S,S}(r)$ 重新排列 r 的属性顺序
- $\Pi_{R-S}(\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r)$ 找出 $\Pi_{R-S}(r)$ 中的元组 t , 得到某些元组 $u \in s$, $tu \notin r$

□ 赋值运算 (\leftarrow) 可以使复杂的查询表达变得简单

- 使用赋值运算，可以把查询表达为一个顺序程序，该程序包括：
 - 一系列赋值
 - 一个其值被作为查询结果显示的表达式
- 对关系代数查询而言，赋值必须是赋给一个临时关系变量

□ 例：可以把 $r \div s$ 写作，

$$temp1 \leftarrow \Pi_{R-S} (r)$$

$$temp2 \leftarrow \Pi_{R-S} ((temp1 \times s) - \Pi_{R-S, S} (r))$$

$$result = temp1 - temp2$$

- 将 \leftarrow 右侧的表达式的结果赋给 \leftarrow 左侧的关系变量，该关系变量可以在后续的表达式中使用

- 例1：找出至少拥有一个“市区”和“住宅区”分支机构的账户的客户姓名

查询1： $\Pi_{CN}(\sigma_{BN = \text{“Downtown”}}(depositor \bowtie account)) \cap$
 $\Pi_{CN}(\sigma_{BN = \text{“Uptown”}}(depositor \bowtie account))$

其中， CN 表示“customer-name”， BN 表示“branch-name”

查询2： $\Pi_{customer-name, branch-name}(depositor \bowtie account)$
 $\div \rho_{temp(branch-name)}(\{(\text{“Downtown”}), (\text{“Uptown”})\})$

- 例2：找出拥有布鲁克林市所有分支机构的帐户的客户姓名

$$\Pi_{customer-name, branch-name} (depositor \bowtie account) \\ \div \Pi_{branch-name} (\sigma_{branch-city = "Brooklyn"} (branch))$$

- 例3：查询选修了全部课程的学生学号和姓名

- 涉及表：课程信息 $Course(cno, cname, pre-cno, score)$ ，选课信息 $SC(sno, cno, grade)$ ，学生信息 $Student(sno, sname, sex, age)$

- 当涉及到求“全部”之类的查询，常用“除法”

- 找出全部课程号： $\Pi_{cno}(Course)$

- 找出选修了全部课程的学生的学号：

- $\Pi_{sno, cno}(SC) \div \Pi_{cno}(Course)$

- 与 $Student$ 表自然连接（连接条件 Sno ）获得学号、姓名

$$(\Pi_{sno, cno}(SC) \div \Pi_{cno}(Course)) \bowtie \Pi_{sno, sname}(Student)$$



- 并、差、交为双目、等元运算
- 笛卡尔积，自然连接，除为双目运算
- 投影、选择为单运算对象
- 关系运算的优先级：
 - 投影
 - 选择
 - 笛卡尔积
 - 连接、除
 - 交
 - 并、差

- 广义投影
- 聚集函数
- 外连接

- 允许在投影列表中使用算术函数来对投影操作进行扩展，广义投影运算形式为：

$$\Pi_{F_1, F_2, \dots, F_n}(E)$$

- E是任意关系代数表达式
- F_1, F_2, \dots, F_n 是涉及E模式中常量和属性的算术表达式
- 给出关系 *credit-info(customer-name, limit, credit-balance)*，找出每个客户还能花费多少，可以表述为：

$$\Pi_{customer-name, \textcolor{brown}{limit} - \textcolor{brown}{credit-balance}}(\textcolor{brown}{credit-info})$$

聚集函数

□ 聚合函数输入一个值集合，然后返回单一值作为结果

- avg: 平均值
- min: 最小值
- max: 最大值
- sum: 值的总和
- count: 值的数量

$g_{avg(balance)}(account)$
(求平均存款余额)

□ 聚集函数的关系代数表示:

$$G_1, G_2, \dots, G_n \quad g_{F_1(A_1), F_2(A_2), \dots, F_n(A_n)}(E)$$

- E是任意关系表达式
- G_1, G_2, \dots, G_n 是用于分组的一系列属性
- F_i 是聚集函数
- A_i 是属性名

聚集函数

□ 例，关系r

A	B	C
α	α	7
α	β	7
β	β	8
β	α	14

$$g_{avg(c)}(r)$$

avg-c
9

$$Ag_{sum(c)}(r)$$

A	sum-c
α	14
β	22

$$Bg_{avg(c)}(r)$$

B	avg-c
α	10.5
β	7.5

按branch-name将关系account分组

branch-name	account-number	balance
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

branch-name g $sum(balance)$ ($account$)

sum-balance

branch-name	
Perryridge	1300
Brighton	1500
Redwood	700

□ 聚集运算的结果是没有名称的

- 可以使用更名运算为其命名

- 可以把重命名作为聚集运算的一部分，如：

branch-name *g* *sum(balance) as sum-balance* (*account*)

- 外连接运算是连接运算的扩展，可以处理缺失信息
- 保留一侧关系中所有与另一侧关系的任意元组都不匹配的元组，再把产生的元组加到自然连接的结构上
- 使用空值：
 - 空值表示值不知道或不存在

外连接

loan_number	branch_name	amount
L_170	Downtown	3000
L_230	Redwood	4000
L_260	Perryridge	1700

customer_name	loan_number
Jones	L_170
Smith	L_230
Hayes	L_155

假设由于
某种原因
造成帐目
不符



内连接
loan ⋈ *borrower*

loan_number	branch_name	amount	customer_name
L_170	Downtown	3000	Jones
L_230	Redwood	4000	Smith

左外连接
loan ⋈_l *borrower*

loan_number	branch_name	amount	customer_name
L_170	Downtown	3000	Jones
L_230	Redwood	4000	Smith
L_260	Perryridge	1700	null

外连接

右外连接

loan ⋈_r *borrower*

loan_number	branch_name	amount	customer_name
L_170	Downtown	3000	Jones
L_230	Redwood	4000	Smith
L_155	null	null	Hayes

全外连接

loan ⋈_{all} *borrower*

loan_number	branch_name	amount	customer_name
L_170	Downtown	3000	Jones
L_230	Redwood	4000	Smith
L_260	Perryridge	1700	null
L_155	null	null	Hayes



- ❑ 元组的某些属性值是可以为空的
- ❑ null表示未知值或值不存在
- ❑ 涉及空的任何算术表达式的结果为空
- ❑ 聚集函数会忽略空值
 - 可以返回空值作为结果
 - 我们遵循SQL对空值的处理语义
- ❑ 为了消除重复和分组，空值和其他值同等对待
 - 一种方法是两个空值被认为是相同的
 - 另一种方法是假设每个空值都是不同的
 - 这两种方法都可行，但我们更愿意遵循SQL对空值的处理语义

- ❑ 与空值的比较将返回一个特殊值：unknown
- ❑ 如果用false代替unknown，那么not ($A < 5$) 与 $A \geq 5$ 的结果就会不相等
- ❑ 使用特殊值unknown的三值逻辑：
 - OR: $(\text{unknown or true}) = \text{true}$
 $(\text{unknown or false}) = \text{unknown}$
 $(\text{unknown or unknown}) = \text{unknown}$
 - AND: $(\text{true and unknown}) = \text{unknown}$
 $(\text{false and unknown}) = \text{false}$
 $(\text{unknown and unknown}) = \text{unknown}$
 - NOT: $(\text{not unknown}) = \text{unknown}$

- ❑ 在SQL中，如果谓词P的值为unknown，那么“ P is unknown”的值为真
- ❑ 如果选择谓词的值未知，那么选择谓词的结果被认为false

数据库的修改

- 数据库的内容可以使用下面的操作来修改：
 - 删除
 - 插入
 - 更新
- 所有这些操作都使用赋值操作表示

删除

- ❑ 删除请求的表达与查询的表达非常相似，不同的是，前者不是要将找出的元组显示给用户，而是要将它们从数据库中去除
- ❑ 这样只能将元组整个地删除，而不能仅删除某些属性上的值
- ❑ 使用关系代数，删除可表达为：

$$r \leftarrow r - E$$

其中， r 是关系， E 是关系代数查询

删除示例

- 删除Perryridge分支机构的所有账户

$account \leftarrow account - \sigma_{branch-name = "Perryridge"} (account)$

- 删除贷款额在0到50之间的所有贷款

$loan \leftarrow loan - \sigma_{amount \geq 0 \text{ and } amount \leq 50} (loan)$

- 删除位于Needham的分支机构的所有账户

$r_1 \leftarrow \sigma_{branch-city = "Needham"} (account \bowtie branch)$

$r_2 \leftarrow \Pi_{branch-name, account-number, balance} (r_1)$

$r_3 \leftarrow \Pi_{customer-name, account-number} (r_2 \bowtie depositor)$

$account \leftarrow account - r_2$

$depositor \leftarrow depositor - r_3$

插入

- 为了将数据插入关系中：
 - 要么指明一个要插入的元组
 - 要么写出一个查询，其结果是要插入的元组集合
- 使用关系代数，插入可表达为：

$$r \leftarrow r \cup E$$

r 是关系， E 是关系代数表达式

- 如果让 E 是一个只包含元组的常量关系，就可以表达为向关系中插入单一元组

插入示例

- 向数据库中插入这样的信息：Smith在Perryridge分支机构的账户A-973上有\$1200

$$account \leftarrow account \cup \{(\text{"Perryridge"}, A-973, 1200)\}$$
$$depositor \leftarrow depositor \cup \{(\text{"Smith"}, A-973)\}$$

- 假设想对Perryridge分支机构的每一个贷款客户赠送一个新的\$200的存款账户，并将其贷款号码作为此账户的号码

$$r_1 \leftarrow (\sigma_{branch-name = \text{"Perryridge"}} (borrower \bowtie loan))$$
$$account \leftarrow account \cup \Pi_{branch-name, loan-number, 200} (r_1)$$
$$depositor \leftarrow depositor \cup \Pi_{customer-name, loan-number} (r_1)$$

- ❑ 某些情况下，可能只希望改变元组中的某个值，而不希望改变元组中的所有值
- ❑ 可以用广义投影运算来完成这个任务：

$$r \leftarrow \Pi_{F_1, F_2, \dots, F_l}(r)$$

- 其中，当第*i*个属性不被修改时， F_i 表示的是*r*的第*i*个属性
- 当第*i*个属性将被修改时， F_i 表示的是一个只涉及常量和*r*的属性的表达式，表达式给出了此属性的新值

更新示例

- 假设要付给所有账户5%的利息

$$account \leftarrow \Pi_{AN, BN, BAL * 1.05} (account)$$

- AN , BN 和 BAL 分别代表 $account-number$, $branch-name$ 和 $balance$

- 假设余额超过\$10 000以上的账户得到6%的利息，而其他账户得到5%的利息

$$account \leftarrow \Pi_{AN, BN, BAL * 1.06} (\sigma_{BAL > 10000} (account)) \\ \cup \Pi_{AN, BN, BAL * 1.05} (\sigma_{BAL \leq 10000} (account))$$

谢谢！