形式化关系查询语言

从第2章~第5章，我们介绍了关系模型，并详细讲述了SQL。在本章中我们将介绍SQL所基于的形式化模型，同时它也是其他关系查询语言的基础。

本章内容包括三种形式化语言。我们首先介绍关系代数，它为广泛应用的SQL查询语言打下了基础。然后我们学习元组关系演算和域关系演算，它们都是基于数学逻辑的声明式查询语言。

6.1 关系代数

关系代数是一种过程化查询语言。它包括一个运算的集合，这些运算以一个或两个关系为输入，产生一个新的关系作为结果。关系代数基本运算有：选择、投影、并、集合差、笛卡儿积和更名。在基本运算以外，还有一些其他运算，即集合交、自然连接和赋值。我们将用基本运算来定义这些运算。

6.1.1 基本运算

选择、投影和更名运算称为一元运算，因为它们对一个关系进行运算。另外三个运算对两个关系进行运算，因而称为二元运算。

6.1.1.1 选择运算

选择(selelct)运算选出满足给定谓词的元组。我们用小写希腊字母sigma(σ)来表示选择，而将谓词写作σ的下标。参数关系在σ后的括号中。因此，为了选择关系*instructor*中属于“物理(Physics)”系的那些元组，我们应写作：

$$\sigma_{dept\_name = \text{"Physics"}}(instructor)$$

如果*instructor*关系如图6-1所示，则从上述查询产生的关系如图6-2所示。

通过书写：

$$\sigma_{salary > 90\,000}(instructor)$$

我们可以找到工资额大于90 000美元的所有元组。

ID	name	dept_name	salary
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

图6-1 关系*instructor*

ID	name	dept_name	salary
22222	Einstein	Physics	95000
33456	Gold	Physics	87000

图6-2 $\sigma_{dept_name = \text{"Physics"}}(instructor)$的结果

通常，我们允许在选择谓词中进行比较，使用的是=，≠，<，≤，>和≥。另外，我们可以用连词*and*(∧)、*or*(∨)和*not*(¬)将多个谓词合并为一个较大的谓词。因此，为了找到物理系中工资额大于90 000美元的教师，我们需要书写：

$$\sigma_{dept\_name = \text{"Physics"} \wedge salary > 90\,000}(instructor)$$

选择谓词中可以包括两个属性的比较。我们以关系*department*为例说明。为了找出那些系名与楼名相同的系，我们可以这样写：

$$\sigma_{dept\_name = building}(department)$$

SQL 与关系代数

此种代数中的选择选项(select)与我们在 SQL 中所使用的关键词 select 具有不一样的含义，这是历史原因造成的。在关系代数中，术语 select 对应于我们在 SQL 中使用的 where。我们在这里强调二者含义的区别，以减少可能出现的混淆。

6.1.1.2 投影运算

假设我们想要列出所有教师的 *ID*、*name* 和 *salary*，而不关心 *dept_name*。**投影**(project)运算使得我们可以产生这样的关系。投影运算是一元运算，它返回作为参数的关系，但把某些属性排除在外。由于关系是一个集合，所以所有重复行均被去除。投影用大写希腊字母 pi(∏)表示，列举所有我们希望在结果中出现的属性作为∏的下标，作为参数的关系在跟随∏后的括号中。因此，我们可以写如下的查询来得到上述的教师列表：

$$\Pi_{ID, name, salary}(instructor)$$

图 6-3 显示了此查询产生的关系。

6.1.1.3 关系运算的组合

关系运算的结果自身也是一个关系，这一事实是非常重要的。考虑一个更复杂的查询"找出物理系的所有教师的名字"，我们写作：

$$\Pi_{name}(\sigma_{dept\_name = "Physics"}(instructor))$$

请注意，对投影运算而言，我们没有给出一个关系的名字来作为其参数，而是用一个对关系进行求值的表达式来作为参数。

一般地说，由于关系代数运算的结果同其输入的类型一样，仍为关系，所以我们可以把多个关系代数运算组合成一个**关系代数表达式**(relational-algebra expression)。将关系代数运算组合成关系代数表达式如同将算术运算(如 +、−和 ÷)组合成算术表达式一样。我们将在 6.1.2 节给出关系代数表达式的形式化定义。

ID	name	salary
10101	Srinivasan	65000
12121	Wu	90000
15151	Mozart	40000
22222	Einstein	95000
32343	El Said	60000
33456	Gold	87000
45565	Katz	75000
58583	Califieri	62000
76543	Singh	80000
76766	Crick	72000
83821	Brandt	92000
98345	Kim	80000

图 6-3 $\Pi_{ID, name, salary}(instructor)$ 的结果

6.1.1.4 并运算

假设有一个查询，要找出开设在 2009 年秋季学期或者 2010 年春季学期或者这二者皆开的所有课程的集合。关系 *section*(图 6-4)中包含这些信息。为了找到 2009 年秋季学期开设的所有课程的集合，我们可以这样写：

$$\Pi_{course\_id}(\sigma_{semester = "Fall" \wedge year = 2009}(section))$$

为了找出在 2010 年春季学期开设的课程，我们可以这样写：

$$\Pi_{course\_id}(\sigma_{semester = "Spring" \wedge year = 2010}(section))$$

为了实现该查询，我们需要将这两个集合**并**(union)起来；即我们需要出现在这两个集合之一的或同时出现在这两个集合中的所有课程段的 ID。我们通过二元运算"并"来获得这些数据，"并"运算像集合论中一样用∪表示。于是，所需表达式为：

$$\Pi_{course\_id}(\sigma_{semester = "Fall" \wedge year = 2009}(section)) \cup \Pi_{course\_id}(\sigma_{semester = "Spring" \wedge year = 2010}(section))$$

此查询产生的关系如图 6-5 所示。需要注意的是，尽管有 3 门课在 2009 年秋季开设，6 门课在 2010 年春季开设，但是在结果中共有 8 个元组。由于关系是集合，所以像 CS-101 这样在两个学期都开设的重复值只留下了单个元组。

可以看到，在我们的例子中，做并运算的两个集合都由 *course_id* 值构成。概括地说，我们必须保证做并运算的关系是相容的。例如，将 *instructor* 关系和 *student* 关系做并运算就没有意义。尽管两个关系都包含 4 个属性，但是二者在 *salary* 和 *tot_cred* 的域上是不同的。在大多数情况下这两个属性的并集

course_id	sec_id	semester	year	building	room_number	time_slot_id
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

图 6-4　关系 *section*

是没有意义的。因此，要使并运算 $r \cup s$ 有意义，我们要求以下两个条件同时成立：

1. 关系 r 和 s 必须是同元的，即它们的属性数目必须相同。

2. 对所有的 i，r 的第 i 个属性的域必须和 s 的第 i 个属性的域相同。

请注意 r 和 s 可以是数据库关系或者作为关系代数表达式结果的临时关系。

6.1.1.5　集合差运算

用 $-$ 表示的**集合差**（set-difference）运算使得我们可以找出在一个关系中而不在另一个关系中的那些元组。表达式 $r - s$ 的结果即一个包含所有在 r 中而不在 s 中的元组的关系。

course_id
CS-101
CS-315
CS-319
CS-347
FIN-201
HIS-351
MU-199
PHY-101

图 6-5　2009 年秋季学期或 2010 年春季学期或者这两个学期都开设的课程

我们可以通过书写如下表达式来找出所有开设在 2009 年秋季学期但是在 2010 年春季学期不开的课程：

$$\Pi_{course\_id}(\sigma_{semester = \text{"Fall"} \wedge year = 2009}(section)) - \Pi_{course\_id}(\sigma_{semester = \text{"Spring"} \wedge year = 2010}(section))$$

该查询产生的关系如图 6-6 所示。

course_id
CS-347
PHY-101

图 6-6　在 2009 年秋季学期开设但在 2010 年春季学期不开的课程

就像并运算一样，我们必须保证集合差运算在相容的关系间进行。因此，为使集合差运算 $r - s$ 有意义，我们要求关系 r 和 s 是同元的，且对于所有的 i，r 的第 i 个属性的域与 s 的第 i 个属性的域都相同。

6.1.1.6　笛卡儿积运算

用 $\times$ 表示的**笛卡儿积**（Cartesian-product）运算使得我们可以将任意两个关系的信息组合在一起。我们将关系 r_1 和 r_2 的笛卡儿积写作 $r_1 \times r_2$。

回忆一下，关系定义为一组域上的笛卡儿积的子集。从这个定义，我们应该已经对笛卡儿积运算的定义有了直观的认识。但是，由于相同的属性名可能同时出现在 r_1 和 r_2 中，我们需要提出一个命名机制来区别这些属性。我们这里采用的方式是找到属性所来自的关系，把关系名称附加到该属性上。例如，$r = instructor \times teaches$ 的关系模式为：

(*instructor. ID*, *instructor. name*, *instructor. dept_name*, *instructor. salary*
teaches. ID, *teaches. course_id*, *teaches. sec_id*, *teaches. semester*, *teaches. year*)

用这样的模式，我们可以区别 *instructor. ID* 和 *teaches. ID*。对那些只在两个关系模式之一中出现的属性，我们通常省略其关系名前缀。这样的简化不会导致任何歧义。这样，我们就可以将 r 的关系模式写作：

(*instructor. ID*, *name*, *dept_name*, *salary*
teaches. ID, *course_id*, *sec_id*, *semester*, *year*)

这个命名规则规定作为笛卡儿积运算参数的关系名字必须不同。这一规定有时会带来一些问题，例如当某个关系需要与自身做笛卡儿积时。当我们在笛卡儿积中使用关系代数表达式的结果时也会产生类似问题，因为我们必须给关系一个名字以引用其属性。在 6.1.1.7 节中，我们将学习如何通过更名运算来避免这些问题。

我们已经知道 $r = instructor \times teaches$ 的关系模式，那么到底哪些元组会出现在 r 中呢？或许你已经想到了，如下所述每一个可能的元组对都形成 r 的一个元组，元组对中一个来自关系 *instructor*（图 6-1）而另一个来自关系 *teaches*（图 6-7）。因此，正如你可以从图 6-8 看到的那样，r 是一个很大的关系，而该图只显示了构成 r 的部分元组。[⊖]

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

图 6-7 关系 *teaches*

inst.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp.Sci.	65000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Comp.Sci.	65000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Comp.Sci.	65000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Comp.Sci.	65000	12121	FIN-201	1	Spring	2010
10101	Srinivasan	Comp.Sci.	65000	15151	MU-199	1	Spring	2010
10101	Srinivasan	Comp.Sci.	65000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
15151	Mozart	Music	40000	10101	CS-101	1	Fall	2009
15151	Mozart	Music	40000	10101	CS-315	1	Spring	2010
15151	Mozart	Music	40000	10101	CS-347	1	Fall	2009
15151	Mozart	Music	40000	12121	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2010
15151	Mozart	Music	40000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
22222	Einstein	Physics	95000	10101	CS-101	1	Fall	2009
22222	Einstein	Physics	95000	10101	CS-315	1	Spring	2010
22222	Einstein	Physics	95000	10101	CS-347	1	Fall	2009
22222	Einstein	Physics	95000	10101	FIN-201	1	Spring	2010
22222	Einstein	Physics	95000	15151	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

图 6-8 *instructor* × *teaches* 的结果

假设 *instructor* 中有 n_1 个元组，*teaches* 中有 n_2 个元组。那么我们可以有 $n_1 \times n_2$ 种方式来选择元组对——每个关系中选出一个元组；因此 r 中有 $n_1 \times n_2$ 个元组。特别地，请注意对 r 中的某些元组 t 来说，可能 $t[instructor.ID] \neq t[teaches.ID]$。

一般地说，如果有关系 $r_1(R_1)$ 和 $r_2(R_2)$，则关系 $r_1 \times r_2$ 的模式是 R_1 和 R_2 串接而成的。关系 R 中包含所有满足以下条件的元组 t：r_1 中存在元组 t_1 且 r_2 中存在元组 t_2，使得 $t[R_1] = t_1[R_1]$ 且 $t[R_2] = t_2[R_2]$。

⊖ 注意，在图 6-8 和图 6-9 中，为了减小表格的宽度，我们把 *instructor. ID* 重命名为 *inst. ID*。

假设我们希望找出物理系的所有教师，以及他们教授的所有课程。为了实现这一要求，我们同时需要关系 *instructor* 和关系 *teaches* 中的信息。如果我们写：

$$\sigma_{dept\_name = \text{"Physics"}}(instructor \times teaches)$$

则其结果就是图 6-9 所示关系。

inst.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
22222	Einstein	Physics	95000	10101	CS-437	1	Fall	2009
22222	Einstein	Physics	95000	10101	CS-315	1	Spring	2010
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2010
22222	Einstein	Physics	95000	15151	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2009
22222	Einstein	Physics	95000	32343	HIS-351	1	Spring	2010
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
33456	Gold	Physics	87000	10101	CS-437	1	Fall	2009
33456	Gold	Physics	87000	10101	CS-315	1	Spring	2010
33456	Gold	Physics	87000	12121	FIN-201	1	Spring	2010
33456	Gold	Physics	87000	15151	MU-199	1	Spring	2010
33456	Gold	Physics	87000	22222	PHY-101	1	Fall	2009
33456	Gold	Physics	87000	32343	HIS-351	1	Spring	2010
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

图 6-9 $\sigma_{dept\_name = \text{"Physics"}}(instructor \times teaches)$ 的结果

这里我们得到的关系中只包含物理系的教师。然而，*course_id* 列却可能包含并非这些教师所教授的课程。（如果你不明白为什么会发生这种情况，请不要忘了笛卡儿积中保留了所有可能的由一个来自 *instructor* 的元组和一个来自 *teaches* 的元组构成的元组对。）

由于笛卡儿积运算将 *instructor* 中的每个元组同 *teaches* 中的每个元组进行联系，所以我们可以确定，如果一名教师来自于物理系并教授某门课程（在关系 *teaches* 中有对应的元组），那么 $\sigma_{dept\_name = \text{"Physics"}}(instructor \times teaches)$ 中必定存在某个元组，它包含该教师的名字，并且满足 *instructor.ID* = *teaches.ID*。因此，如果我们用：

$$\sigma_{instructor.ID = teaches.ID}(\sigma_{dept\_name = \text{"Physics"}}(instructor \times teaches))$$

就可以得到在 *instructor* × *teaches* 中那些只与物理系教师以及他们所教课程相关的元组。

最后，由于我们只需要物理系教师的名字，以及他们所教课程的 *course_id*，我们进行投影：

$$\Pi_{name, course\_id}(\sigma_{instructor.ID = teaches.ID}(\sigma_{dept\_name = \text{"Physics"}}(instructor \times teaches)))$$

此表达式的结果如图 6-10 所示，这就是我们的查询的正确答案。可以看到，虽然 Gold 属于物理系，但是（从关系 *teaches* 可知）他不教课，因此未出现在结果中。

name	*course_id*
Einstein	PHY-101

图 6-10 $\Pi_{name, course_id}(\sigma_{instructor.ID = teaches.ID}(\sigma_{dept_name = \text{"Physics"}}(instructor \times teaches)))$的结果

217
|
225

需要注意，用关系代数表达查询的方法并不唯一。考虑如下查询：

$$\Pi_{name, course\_id}(\sigma_{instructor.ID = teaches.ID}((\sigma_{dept\_name = \text{"Physics"}}(instructor)) \times teaches))$$

注意这两个查询的细微差别：上面的查询是对 *instructor* 进行选择运算，把 *dept_name* 限制为 Physics，之后再进行笛卡儿积；相比之下，在前面介绍的查询中，在选择运算之前先计算笛卡儿积。然而这两种查询是**等价的**（equivalent），也就是说，在任何数据库上它们都能得到相同的结果。

6.1.1.7 *更名运算*

关系代数表达式的结果没有可供我们引用的名字，这一点与数据库中的关系有所不同。如果能给它们赋上名字那将是十分有用的；我们可以通过小写希腊字母 rho（ρ）表示的**更名**（rename）运算来完成这一任务。对给定关系代数表达式 E，表达式

$$\rho_x(E)$$

返回表达式 E 的结果，并把名字 x 赋给了它。

对于一个关系 r 来说，它自身被认为是一个(平凡的)关系代数表达式。因此，我们也可以将更名运算运用于关系 r，这样可得到具有新名字的一个相同的关系。

更名运算的另一形式如下。假设关系代数表达式 E 是 n 元的，则表达式

$$\rho_{x(A_1,A_2,\cdots,A_n)}(E)$$

返回表达式 E 的结果，并赋给它名字 x，同时将各属性更名为 A_1，A_2，…，A_n。

我们以查询"找出大学里的最高工资"作为例子来演示关系的更名运算。我们的策略是：首先计算出一个由非最高工资组成的临时关系，然后计算关系 $\Pi_{salary}(instructor)$ 和刚才算出的临时关系之间的集合差，从而得到结果。

1. 步骤 1：为了计算该临时关系，我们需要比较所有工资的值。要做这样的比较，可以通过计算笛卡儿积 *instructor* × *instructor* 并构造一个选择来比较任意两个出现在同一元组中的工资。我们的首要任务是设计一种机制来区别两个 *salary* 属性。我们将使用更名运算来改变其中一个对教师关系进行引用的名字；这样我们就可以无歧义地两次引用这个关系。

226

现在可以把非最高工资构成的临时关系写作：

$$\Pi_{instructor.\ salary}(\sigma_{instructor.\ salary < d.\ salary}(instructor \times \rho_d(instructor)))$$

通过这一表达式可以得到 *instructor* 中满足如下条件的那些工资：在关系 *instructor*(更名为 *d*)中还存在比它更高的工资。结果中包含了除最高工资以外的所有工资。图 6-11 表示这个关系。

salary
65000
90000
40000
60000
87000
75000
62000
72000
80000
92000

图 6-11 子表达式 $\Pi_{instructor.\ salary}(\sigma_{instructor.\ salary < d.\ salary}(instructor \times \rho_d(instructor)))$ 的结果

2. 步骤 2：查找大学里最高工资的查询可写作：

$$\Pi_{salary}(instructor) - \Pi_{instructor.\ salary}(\sigma_{instructor.\ salary < d.\ salary}(instructor \times \rho_d(instructor)))$$

该查询的结果如图 6-12 所示。

salary
95000

图 6-12 大学里的最高工资

更名运算不是必需的，因为可以对属性使用位置标记。我们可以用位置标记隐含地作为关系的属性名，用 $1、$2、…指代第一个属性、第二个属性，以此类推。位置标记也适用于关系代数表达式运算的结果。下面的关系代数表达式演示了位置标记的用法，我们用它来写先前介绍的计算非最高工资的表达式：

$$\Pi_{\$4}(\sigma_{\$4 < \$8}(instructor \times instructor))$$

注意，笛卡儿积把两个关系的属性串接起来。因此，在笛卡儿积(*instructor* × *teaches*)中 $4 代表第一个 *instructor* 的属性 *salary*，而 $8 代表第二个 *instructor* 的属性 *salary*。如果一个二元运算需要在作为其运算对象的两个关系之间进行区分，类似的位置标记也可以用来作为关系名称。例如，$R1 可以指代第一个作为运算对象的关系，而 $R2 可以指代第二个关系。然而，位置标记对人而言不够方便，因为属性的位置是一个数字，不像属性名那样易于记忆。所以在这本书里我们不采用位置标记。

6.1.2 关系代数的形式化定义

6.1.1 节中的运算使我们能够给出关系代数中表达式的完整定义。关系代数中基本的表达式是如下二者之一：

- 数据库中的一个关系。
- 一个常数关系。

常数关系可以用在{}内列出它的元组来表示，例如{(22222, Einstein, Physics, 95000), (76543, Singh, Finance, 80000)}。

关系代数中一般的表达式是由更小的子表达式构成的。设 E_1 和 E_2 是关系代数表达式，则以下这些都是关系代数表达式：

- $E_1 \cup E_2$。
- $E_1 - E_2$。
- $E_1 \times E_2$。
- $\sigma_P(E_1)$，其中 P 是 E_1 的属性上的谓词。
- $\Pi_S(E_1)$，其中 S 是 E_1 中某些属性的列表。
- $\rho_x(E_1)$，其中 x 是 E_1 结果的新名字。

6.1.3 附加的关系代数运算

关系代数的基本运算足以表达任何关系代数查询。但是，如果我们把自己局限于基本运算，某些常用查询表达出来会显得冗长。因此，我们定义附加的一些运算，它们不能增强关系代数的表达能力，却可以简化一些常用的查询。对每个新运算，我们给出一个只采用基本运算的等价表达式。 227 228

6.1.3.1 集合交运算

我们要定义的第一个附加的关系代数运算是**集合交**(intersection)(∩)。假设我们希望找出在2009年秋季和2010年春季都开设的课程。使用集合交运算，我们可以写为

$$\Pi_{course\_id}(\sigma_{semester = \text{"Fall"} \wedge year = 2009}(section)) \cap \Pi_{course\_id}(\sigma_{semester = \text{"Spring"} \wedge year = 2010}(section))$$

此查询产生的关系如图6-13所示。

course_id
CS-101

图6-13 在2009年秋季学期和2010年春季学期都开设的课程

请注意，任何使用了集合交的关系代数表达式，我们都可以通过用一对集合差运算替代集合交运算来重写，如下所示：

$$r \cap s = r - (r - s)$$

因此，集合交不是基本运算，不能增强关系代数的表达能力。只不过写 $r \cap s$ 比写 $r - (r - s)$ 要方便。

6.1.3.2 自然连接运算

对某些要用到笛卡儿积的查询进行简化常常是我们需要的。通常情况下，涉及笛卡儿积的查询中会包含一个对笛卡儿积结果进行选择的运算。该选择运算大多数情况下会要求进行笛卡儿积的两个关系在所有相同属性上的值相一致。

6.1.1.6中的例子把表 *instructor* 和 *teaches* 的信息相结合，匹配条件是要满足 *instructor.ID* 和 *teaches.ID* 相等。在这两个关系中只有它们有相同的属性名。

二元运算自然连接使得我们可以将某些选择和笛卡儿积运算合并为一个运算。我们用**连接**(join)符号⋈来表示它。自然连接运算首先形成它的两个参数的笛卡儿积，然后基于两个关系模式中都出现的属性上的相等性进行选择，最后还要去除重复属性。回到关系 *instructor* 和 *teaches* 的例子中，计算 *instructor* 和 *teaches* 的**自然连接**时，只考虑由 *instructor* 和 *teaches* 中在相同属性 *ID* 上有相同值的元组组成 229 的元组对。结果关系如图6-14所示，只包含13个元组，每个元组记录了某个教师及其实际教授的一门课程的信息。注意，我们并不重复记录那些在两个关系的模式中都出现的属性。还要注意所列出的属性的顺序：排在最前面的是两个关系模式的相同属性，其次是只属于第一个关系模式的属性，最后是只属于第二个关系模式的属性。

尽管自然连接的定义很复杂，这种运算使用起来却很方便。举例来说，考虑一下查询"找出所有教师的姓名，连同他们教的所有课程的 *course_id*"。用自然连接我们可以将此查询表述如下：

$$\Pi_{name, course\_id}(instructor \bowtie teaches)$$

ID	name	dept_name	salary	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010
83821	Brandt	Comp. Sci.	92000	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-190	2	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-319	2	Spring	2010
98345	Kim	Elec. Eng.	80000	EE-181	1	Spring	2009

图 6-14 关系 *instructor* 同 *teaches* 进行自然连接的结果

由于 *instructor* 和 *teaches* 的模式中具有相同属性 *ID*，自然连接运算只考虑在 *ID* 上值相同的元组对。它将每个这样的元组对合并为单一的元组，其模式为两个模式的并，即(*ID*, *name*, *dept_name*, *salary*, *course_id*)。在执行投影后，我们得到的关系如图 6-15 所示。

name	course_id
Srinivasan	CS-101
Srinivasan	CS-315
Srinivasan	CS-347
Wu	FIN-201
Mozart	MU-199
Einstein	PHY-101
El Said	HIS-351
Katz	CS-101
Katz	CS-319
Crick	BIO-101
Crick	BIO-301
Brandt	CS-190
Brandt	CS-319
Kim	EE-181

图 6-15 $\Pi_{name,course\_id}$ (*instructor* ⋈ *teaches*) 的结果

考虑两个关系模式 R 和 S，当然，它们都是属性名的一个列表。如果我们认为模式是集合而不是列表，我们可以用 $R \cap S$ 表示同时出现在 R 和 S 中的那些属性名，用 $R \cup S$ 表示出现在 R 中或 S 中或在二者中都出现的那些属性名。类似地，出现在 R 中而不出现在 S 中的属性名用 $R - S$ 表示，出现在 S 中而不出现在 R 中的属性名用 $S - R$ 表示。请注意这里的并、交、差运算都是属性的集合上的，而不是关系上的。

我们现在就可以给出自然连接的形式化定义。设 $r(R)$ 和 $s(S)$ 是两个关系。r 和 s 的**自然连接**(natural join)表示为 $r \bowtie s$，是模式 $R \cup S$ 上的一个关系，其形式化定义如下：

$$r \bowtie s = \Pi_{R \cup S}(\sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \cdots \wedge r.A_n = s.A_n}(r \times s))$$

其中 $R \cap S = \{A_1, A_2, \cdots, A_n\}$。

请注意如果关系 $r(R)$ 和 $s(S)$ 不含有任何相同属性，即 $R \cap S = \varnothing$，那么 $r \bowtie s = r \times s$。

这里再给出一个使用自然连接的例子，写查询"找出计算机系(Comp. Sci.)的所有教师，以及他们教授的所有课程的名称"。

$$\Pi_{name,title}(\sigma_{dept\_name = \text{"Comp. Sci."}}(instructor \bowtie teaches \bowtie course))$$

此查询的结果关系如图 6-16 所示。

name	title
Brandt	Game Design
Brandt	Image Processing
Katz	Image Processing
Katz	Intro. to Computer Science
Srinivasan	Intro. to Computer Science
Srinivasan	Robotics
Srinivasan	Database System Concepts

图 6-16 $\Pi_{name,title}(\sigma_{dept_name = \text{"Comp. Sci."}}(instructor \bowtie teaches \bowtie course))$的结果

注意上式中我们在 *instructor* ⋈ *teaches* ⋈ *course* 中没有加入括号来表明自然连接在这三个关系间执行的顺序。上述情况下有两种可能：

$$(instructor \bowtie teaches) \bowtie course$$
$$instructor \bowtie (teaches \bowtie course)$$

我们没有说明我们希望的是哪个表达式，因为二者是等价的。也就是说，自然连接是**可结合的**(associative)。

*theta 连接*是自然连接的扩展，它使得我们可以把一个选择运算和一个笛卡儿积运算合并为单独的一个运算。考虑关系 $r(R)$ 和 $s(S)$，并设 θ 是模式 $R \cup S$ 的属性上的谓词，则 **theta 连接**(theta join)运算 $r \bowtie_\theta s$ 定义如下：

$$r \bowtie_\theta s = \sigma_\theta(r \times s)$$

6.1.3.3 赋值运算

有时通过给临时关系变量赋值的方法来写关系代数表达式会很方便。**赋值**(assignment)运算用←表示，与程序语言中的赋值类似。为了说明这个运算，我们来看自然连接运算的定义。我们可以把 $r \bowtie s$ 写作：

$$temp1 \leftarrow R \times S$$
$$temp2 \leftarrow \sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \cdots \wedge r.A_n = s.A_n}(temp1)$$
$$result = \prod_{R \cup S}(temp2)$$

赋值的执行不会使得把某个关系显示给用户看，而是将←右侧的表达式的结果赋给←左侧的关系变量。该关系变量可以在后续的表达式中使用。

使用赋值运算，我们可以把查询表达为一个顺序程序，该程序由一系列赋值加上一个其值被作为查询结果显示的表达式组成。对关系代数查询而言，赋值必须是赋给一个临时关系变量。对永久关系的赋值形成了对数据库的修改。请注意，赋值运算不能增强关系代数的表达能力，但是可以使复杂查询的表达变得简单。

6.1.3.4 外连接运算

外连接(outer-join)运算是连接运算的扩展，可以处理缺失的信息。假设有一些教师不教课。那么关系 *instructor*(图 6-1)中这些教师对应的元组将不满足与关系 *teaches*(图 6-7)进行自然连接的条件，所以在图 6-14 所示的自然连接的结果中找不到他们的数据。以教师 Califieri、Gold 和 Singh 为例，因为他们不教课，所以就不会出现在自然连接的结果中。 230 ~ 232

更一般地，参加连接的一个或两个关系中的某些元组可能会这样地"丢失"。**外连接**(outer join)运算与前面学过的自然连接运算比较类似，不同之处在于它在结果中创建带空值的元组，以此来保留在连接中丢失的那些元组。

我们可以用外连接运算来避免上述的信息丢失。外连接运算有三种形式：*左外连接*，用⟕表示；*右外连接*，用⟖表示；*全外连接*，用⟗表示。这三种形式的外连接都要计算连接，然后在连接结果中添加额外的元组。例如，表达式 *instructor* ⟕ *teaches* 和 *teaches* ⟖ *instructor* 的结果分别如图 6-17 和图 6-18 所示。

ID	*name*	*dept_name*	*salary*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
33456	Gold	Physics	87000	*null*	*null*	*null*	*null*
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
58583	Califieri	History	62000	*null*	*null*	*null*	*null*
76543	Singh	Finance	80000	*null*	*null*	*null*	*null*
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010
83821	Brandt	Comp. Sci.	92000	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-190	2	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-319	2	Spring	2010
98345	Kim	Elec. Eng.	80000	EE-181	1	Spring	2009

图 6-17 *instructor* ⟕ *teaches* 的结果

ID	course_id	sec_id	semester	year	name	dept_name	salary
10101	CS-101	1	Fall	2009	Srinivasan	Comp. Sci.	65000
10101	CS-315	1	Spring	2010	Srinivasan	Comp. Sci.	65000
10101	CS-347	1	Fall	2009	Srinivasan	Comp. Sci.	65000
12121	FIN-201	1	Spring	2010	Wu	Finance	90000
15151	MU-199	1	Spring	2010	Mozart	Music	40000
22222	PHY-101	1	Fall	2009	Einstein	Physics	95000
32343	HIS-351	1	Spring	2010	El Said	History	60000
33456	*null*	*null*	*null*	*null*	Gold	Physics	87000
45565	CS-101	1	Spring	2010	Katz	Comp. Sci.	75000
45565	CS-319	1	Spring	2010	Katz	Comp. Sci.	75000
58583	*null*	*null*	*null*	*null*	Califieri	History	62000
76543	*null*	*null*	*null*	*null*	Singh	Finance	80000
76766	BIO-101	1	Summer	2009	Crick	Biology	72000
76766	BIO-301	1	Summer	2010	Crick	Biology	72000
83821	CS-190	1	Spring	2009	Brandt	Comp. Sci.	92000
83821	CS-190	2	Spring	2009	Brandt	Comp. Sci.	92000
83821	CS-319	2	Spring	2010	Brandt	Comp. Sci.	92000
98345	EE-181	1	Spring	2009	Kim	Elec. Eng.	80000

图 6-18 *teaches* ⟗ *instructor* 的结果

左外连接(left outer join)(⟕)取出左侧关系中所有与右侧关系的任一元组都不匹配的元组，用空值填充所有来自右侧关系的属性，再把产生的元组加到自然连接的结果中。图 6-17 中，元组(58583，Califieri，History，62000，*null*，*null*，*null*，*null*)即是这样的元组。所有来自左侧关系的信息在左外连接结果中都得到保留。

右外连接(right outer join)(⟖)与左外连接相对称：用空值填充来自右侧关系的所有与左侧关系的任一元组都不匹配的元组，将结果加到自然连接的结果中。图 6-18 中，(58583，*null*，*null*，*null*，*null*，Califieri，History，62000)即是这样的元组。因此，所有来自右侧关系的信息在右外连接结果中都得到保留。

全外连接(full outer join)(⟗)既做左外连接又做右外连接，既填充左侧关系中与右侧关系的任一元组都不匹配的元组，又填充右侧关系中与左侧关系的任一元组都不匹配的元组，并把结果都加到连接的结果中。

注意，从左外连接的例子到右外连接的例子，我们交换了运算对象的顺序。这样它们都保留了 *instructor* 的元组，因此包含相同的内容。对于我们的示例关系，*teaches* 的元组在 *instructor* 中总能找到与之对应的元组，所以 *teaches* ⟕ *instructor* 与 *teaches* ⋈ *instructor* 产生的结果是一样的。如果在 *teaches* 中有元组在 *instructor* 中找不到对应元组，该元组将会出现在 *teaches* ⟕ *instructor* 的结果以及 *teaches* ⟗ *instructor* 的结果中，所默认的部分用空值填充。外连接的更多例子(以 SQL 语法表达)请参考 4.1.2 节。

由于外连接可能产生含有空值的结果，我们需要了解不同的关系代数运算是如何处理空值的。3.6 节涉及 SQL 环境中的这个问题。相同的概念也适用于关系代数的情况，我们在这里不再赘述。

233 ~ 234

请注意，很有趣的是外连接运算可以用基本关系代数运算表示。例如，左连接运算 $r ⟕ s$ 可以写成：

$$(r ⋈ s) ∪ (r - \Pi_R(r ⋈ s)) \times \{(null, \cdots, null)\}$$

其中常数关系 $\{(null, \cdots, null)\}$ 的模式是 $S - R$。

6.1.4 扩展的关系代数运算

我们现在来介绍另外几个关系代数运算，它们可以实现一些不能用基本的关系代数运算来表达的查询。这些运算被称为**扩展的关系代数**(extended relational-algebra)运算。

6.1.4.1 广义投影

第一个运算是**广义投影**(generalized-projection)，它通过允许在投影列表中使用算术运算和字符串函数等来对投影进行扩展。广义投影运算形式为：

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

其中 E 是任意关系代数表达式，而 F_1，F_2，…，F_n 中的每一个都是涉及常量以及 E 的模式中属性的算术表达式。最基本的情况下算术表达式可以仅仅是一个属性或常量。通常来说，在表达式中可以使用

对数值属性或者产生数值结果的表达式的 +、-、*、/等代数运算。广义投影还允许其他数据类型上的运算，比如对字符串的串接。

例如，表达式：

$$\Pi_{ID,name,dept\_name,salary/12}(instructor)$$

可以得到每个教师的 *ID*、*name*、*dept_name* 以及每月的工资。

6.1.4.2 聚集

第二个扩展的关系代数运算是聚集运算 $\mathcal{G}$，可以用来对值的集合使用聚集函数，例如计算最小值或者求平均值。

聚集函数(aggregate function)输入值的一个汇集，将单一值作为结果返回。例如，聚集函数 **sum** 输入值的一个汇集，返回这些值的和。因此，将函数 **sum** 用于汇集

$$\{1, 1, 3, 4, 4, 11\}$$

235

上，返回值24。聚集函数 **avg** 返回值的平均，当用于上述汇集，返回值为4。聚集函数 **count** 返回汇集中元素的个数，对上述汇集将返回6。除此之外，常用聚集函数还有 **min** 和 **max**，它们分别返回汇集中的最小值和最大值，对上面的汇集分别返回1和11。

使用聚集函数对其进行操作的汇集中，一个值可以出现多次，值出现的顺序是无关紧要的。这样的汇集称为**多重集**(multiset)。集合(set)是多重集的特例，其中每个值都只出现一次。

我们用关系 *instructor* 来说明聚集的概念。假设我们希望找出所有教师的工资总和，这一查询的关系代数表达式为：

$$\mathcal{G}_{\mathbf{sum}(salary)}(instructor)$$

符号 $\mathcal{G}$ 是字母 G 的书法体；读作“书法体 G”。关系代数运算 $\mathcal{G}$ 表示聚集将被应用，它的下标说明采用的聚集运算。上述表达式的结果是一个单一属性的关系，且只包含单独的一行，其值表示所有教师的工资总和。

有时，在计算聚集函数前我们必须去除重复值。如果我们想去除重复，我们仍然使用前面的函数名，但用连字符将“**distinct**”附加在函数名后(如 **count-distinct**)。以查询“找出在2010年春季学期教课的教师数”为例。在这里，每名教师只应计算一次，而不管他教了几门课程。关系 *teaches* 包含了所需的信息，所以我们把此查询表达如下：

$$\mathcal{G}_{\mathbf{count\_distinct}(ID)}(\sigma_{semester=\text{“Spring”}\wedge year=2010}(teaches))$$

聚集函数 **count-distinct** 可以确保：即使某位教师授课多于一门，在结果中也只对他计数一次。

有时候我们希望对一组元组集合而不是单个元组集合执行聚集函数。作为示例，考虑查询“求出每个系的平均工资”。我们把此查询表达如下：

$$_{dept\_name}\mathcal{G}_{\mathbf{average}(salary)}(instructor)$$

图6-19显示了通过 *dept_name* 属性对关系 *instructor* 进行分组得到的元组，这是计算查询结果的第一步。然后对每个组执行指定的聚集，查询结果如图6-20所示。

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

图6-19 通过属性 *dept_name* 对关系 *instructor* 分组后的元组

dept_name	*salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

图6-20 查询“求出每个系的平均工资”的结果关系

作为对比，对于查询"求出所有教师的平均工资"，我们可以如下表达：

$$\mathcal{G}_{\text{average}(salary)}(instructor)$$

此时，从运算符 $\mathcal{G}$ 的左边去掉了属性 *dept_name*，这样一来整个关系就被当作单个组来执行聚集。

多重集关系代数

与关系代数不同的是，SQL 允许在输入关系以及查询结果中存在元组的多重拷贝。SQL 标准做了如下定义：在一个查询的输出结果中每个元组有多少个拷贝取决于所对应的元组在输入关系中的拷贝个数。

为了映射 SQL 的这种模式，我们定义了一种被称为**多重集关系代数**(multiset relational algebra)的关系代数版本来对多重集(即存在重复的集合)进行操作。多重集关系代数的基本运算有如下定义：

1. 如果在 r_1 中元组 t_1 有 c_1 份拷贝，并且 t_1 满足选择 σ_θ，那么在 $\sigma_\theta(r_1)$ 中元组 t_1 有 c_1 份拷贝。
2. 对于 r_1 中元组 t_1 的每份拷贝，在 $\Pi_A(r_1)$ 中都有一个与之对应的 $\Pi_A(t_1)$，$\Pi_A(t_1)$ 表示单个元组 t_1 的投影。
3. 如果在 r_1 中元组 t_1 有 c_1 份拷贝，在 r_2 中元组 t_2 有 c_2 份拷贝，那么在 $r_1 \times r_2$ 中就有元组 $t_1 . t_2$ 的 $c_1 * c_2$ 份拷贝。

例如，假设关系 r_1 的模式是 (A, B)，r_2 的模式是 (C)，它们是如下的多重集：

$$r_1 = \{(1, a), (2, a)\},\ r_2 = \{(2), (3), (3)\}$$

那么 $\Pi_B(r_1)$ 就得到 $\{(a), (a)\}$，而 $\Pi_B(r_1) \times r_2$ 的结果是

$$\{(a, 2), (a, 2), (a, 3), (a, 3), (a, 3), (a, 3)\}$$

按照 SQL 中的相应含义(见 3.5 节)，我们还可以用相似的方法来定义多重集的并、交以及集合差运算。而对于聚集运算来说，在多重集下的定义并没有什么变化。

聚集运算(aggregation operation)$\mathcal{G}$通常的形式如下：

$$_{G_1, G_2, \cdots, G_n}\mathcal{G}_{F_1(A_1), F_2(A_2), \cdots, F_n(A_n)}(E)$$

其中 E 是任意关系代数表达式，G_1，G_2，…，G_n 是用于分组的一系列属性；每个 F_i 是一个聚集函数，每个 A_i 是一个属性名。运算的含义如下，表达式 E 的结果中元组以如下方式被分成若干组：

1. 同一组中所有元组在 G_1，G_2，…，G_n 上的值相同。
2. 不同组中元组在 G_1，G_2，…，G_n 上的值不同。

236 | 238

因此，各组可以用属性 G_1，G_2，…，G_n 上的值来唯一标识。对每个组 $(g_1, g_2, \cdots, g_n)$ 来说，结果中有一个元组 $(g_1, g_2, \cdots, g_n, a_1, a_2, \cdots, a_m)$，其中对每个 i，a_i 是将聚集函数 F_i 作用于该组的属性 A_i 上的多重值集所得到的结果。

作为聚集运算的特例，属性列 G_1，G_2，…，G_n 可以是空的，在这种情况下，会有唯一一组包含关系中所有的元组。这相当于没有分组。

SQL 与关系代数

通过关系代数运算和 SQL 运算的比较，可以清楚地看到二者之间联系紧密。典型的 SQL 查询样式如下：

select $A_1, A_2, \cdots, A_n$
from $r_1, r_2, \cdots, r_m$
where P

每个 A_i 表示一个属性，每个 r_i 代表一个关系。P 表示一个谓词。这个查询等价于多重集关系代数表达式：

$$\Pi_{A_1,A_2,\cdots,A_x}(\sigma_P(r_1 \times r_2 \times \cdots \times r_m))$$

如果省略掉 **where** 子句，那么谓词 P 就是 **true**。

更复杂的 SQL 查询也可以用关系代数来重写。例如，查询：

select A_1, A_2, **sum**(A_3)
from r_1, r_2, ..., r_m
where P
group by A_1, A_2

等价于：

$$_{A_1,A_2}\mathcal{G}_{sum(A_3)}(\Pi_{A_1,A_2,\cdots,A_x}(\sigma_P(r_1 \times r_2 \times \cdots \times r_m)))$$

from 子句中的连接表达式可以用关系代数中与之等价的连接表达式来写；我们把详细内容留给读者作为练习。然而 **where** 或者 **select** 子句中的子查询却不能像这样直接用关系代数来重写，这是因为默认与子查询结构等价的关系代数运算。为了解决这个问题，有人提出了对关系代数的扩展，但是这些内容不在本书讨论范围之内。

6.2 元组关系演算

当书写关系代数表达式时，提供了产生查询结果的过程序列，这个序列能生成查询的答案。与之相比，元组关系演算是非过程化的(nonprocedural)查询语言。它只描述所需信息，而不给出获得该信息的具体过程。

元组关系演算中的查询表达为：

$$\{t \mid P(t)\}$$ 239

也就是说，它是所有使谓词 P 为真的元组 t 的集合。和前面的记法一样，我们用 $t[A]$ 表示元组 t 在属性 A 上的值，并用 $t \in r$ 表示元组 t 在关系 r 中。

在给出元组关系演算的形式化定义之前，我们先回头看几个在 6.1.1 节中我们用关系代数表达式书写过的查询。

6.2.1 查询示例

我们希望找出所有工资在 80 000 美元以上的教师的 *ID*、*name*、*dept_name* 和 *salary*：

$$\{t \mid t \in instructor \wedge t[salary] > 80000\}$$

假设我们只需要 *ID* 属性，而不是关系 *instructor* 的所有属性。为了用元组关系演算来书写这个查询，我们需要为模式(*ID*)上的关系写一个表达式。我们需要(*ID*)上在 *instructor* 中对应元组的属性 *salary* > 80 000 的那些元组。为了表述这样的要求，我们需要引入数理逻辑中的“存在”这一结构。记法

$$\exists t \in r(Q(t))$$

表示“关系 r 中存在元组 t 使谓词 $Q(t)$ 为真”。

用这种记法，我们可以将查询“找出工资大于 80 000 美元的所有教师的 *ID*”表述为：

$$\{t \mid \exists s \in instructor(t[ID] = s[ID] \wedge s[salary] > 80000)\}$$

我们可以这样来读上述表达式：“它是所有满足如下条件的元组 t 的集合：在关系 *instructor* 中存在元组 s 使 t 和 s 在属性 *ID* 上的值相等，且 s 在属性 *salary* 上的值大于 80 000 美元”。

元组变量 t 只定义在 *ID* 属性上，因为这一属性是对 t 进行限制的条件所涉及的唯一属性。因此，结果得到(*ID*)上的关系。

考虑查询“找出位置在 Watson 楼的系中的所有教师姓名”。这个查询比前一个稍微复杂一些，因为它涉及 *instructor* 和 *department* 两个关系。但是，正如我们将看到的一样，所需的只不过是在元组关系

演算表达式中使用两个“存在”子句，并通过 *and*(∧)把它们连接起来。我们将此查询表述如下：

$$\{t \mid \exists s \in instructor(t[name] = s[name] \\ \wedge \exists u \in department(u[dept\_name] = s[dept\_name] \\ \wedge u[building] = \text{"Watson"}))\}$$

240

元组变量 u 保证该系位于 Watson 楼，元组变量 s 被限制到与 u 的 *dept_name* 相同。查询产生的结果如图 6-21所示。

name
Einstein
Crick
Gold

图 6-21 位置在 Watson 楼的系中的所有教师姓名

为了找出在 2009 年秋季学期或者 2010 年春季学期或者这两个学期都开设的所有课程的集合，我们在关系代数中使用了并运算。在元组关系演算中，我们将用到用 *or*(∨)连接的两个“存在”子句：

$$\{t \mid \exists s \in section(t[course\_id] = s[course\_id]) \\ \wedge s[semester] = \text{"Fall"} \wedge s[year] = 2009) \\ \vee \exists u \in section(u[course\_id] = t[course\_id]) \\ \wedge u[semester] = \text{"Spring"} \wedge u[year] = 2010)\}$$

此表达式给出所有至少满足下面两个条件之一的 *course_id* 元组的集合：

- 在关系 *section* 中满足 *semeter* = Fall 且 *year* = 2009 的某个元组包含该 *course_id*。
- 在关系 *section* 中满足 *semeter* = Spring 且 *year* = 2010 的某个元组包含该 *course_id*。

如果同一门课在 2009 年秋季和 2010 年春季学期都开课，那么它的 *course_id* 在结果中也只出现一次，因为集合的数学定义不允许重复成员。此查询的结果已在前面的图 6-5 中展示。

假设我们现在只想找到那些在 2009 年秋季和 2010 年春季学期都开设的课程的 *course_id*，所需做的只不过是把上述表达式中的 *or*(∨)用 *and*(∧)替代。

$$\{t \mid \exists s \in section(t[course\_id] = s[course\_id]) \\ \wedge s[semester] = \text{"Fall"} \wedge s[year] = 2009) \\ \wedge \exists u \in section(u[course\_id] = t[course\_id]) \\ \wedge u[semester] = \text{"Spring"} \wedge u[year] = 2010)\}$$

此查询结果如图 6-13 所示。

现在考虑查询“找出 2009 年秋季学期开设而 2010 年春季学期不开的所有课程”。这一查询的元组关系演算表达式同上面的表达式类似，只是使用了 *not*(¬)符号：

241

$$\{t \mid \exists s \in section(t[course\_id] = s[course\_id]) \\ \wedge s[semester] = \text{"Fall"} \wedge s[year] = 2009) \\ \wedge \neg \exists u \in section(u[course\_id] = t[course\_id]) \\ \wedge u[semester] = \text{"Spring"} \wedge u[year] = 2010)\}$$

这个元组关系演算表达式用 $\exists s \in section(\cdots)$子句来要求该 *course_id* 开设在 2009 年秋季学期，用 $\neg \exists u \in section(\cdots)$子句来去掉那些作为在 2010 年春季学期开设的课程出现在关系 *section* 中的 *course_id*。

下面我们要考虑的查询将用到蕴含，用 ⇒ 表示。公式 $P \Rightarrow Q$ 表示“P 蕴含 Q”，即“如果 P 为真，则 Q 必然为真”。$P \Rightarrow Q$ 逻辑上等价于 $\neg P \vee Q$。用蕴含而不是 *not* 和 *or* 常常可以更直观地表达查询。

我们来看这样一个查询：“找出所有那些选了生物(Biology)系全部课程的学生”。为了用元组关系演算书写此查询，我们引入“对所有的”结构，用 ∀ 表示。记法

$$\forall t \in r(Q(t))$$

表示“对关系 r 中所有元组 t，Q 均为真”。

此查询的表达式如下：

$$\{t \mid \exists r \in student(r[ID] = t[ID]) \wedge \\ \forall u \in course(u[dept\_name] = \text{"Biology"} \Rightarrow \\ \exists s \in takes(t[ID] = s[ID] \\ \wedge s[course\_id] = u[course\_id]))\}$$

我们可以这样来读上述表达式："它是所有满足如下条件的学生(即(*ID*)上的元组 t)的集合：对关系 *course* 中所有的元组 u，如果 u 在 *dept_name* 属性上的值是'Biology'，那么在关系 *takes* 中一定存在一个包含该学生 *ID* 以及该课程 *course_id* 的元组"。

注意上述查询中有一点很微妙：如果生物系没有开设任何课程，则所有的学生 *ID* 都满足条件。在这种情况下，上述查询表达式的第一行非常关键。如果没有条件：

$$\exists r \in student(r[ID] = t[ID])$$

那么若生物系没有课程，则任何一个 t 值(包括不是关系 *student* 里学生 *ID* 的值)都会符合要求。 242

6.2.2 形式化定义

我们现在可以给出形式化定义了。元组关系演算表达式具有如下形式：

$$\{t \mid P(t)\}$$

其中 P 是一个公式。公式中可以出现多个元组变量。如果元组变量不被 ∃ 或 ∀ 修饰，则称为自由变量。因此，在

$$t \in instructor \wedge \exists s \in department(t[dept\_name] = s[dept\_name])$$

中，t 是自由变量，而元组变量 s 称为受限变量。

元组关系演算的公式由原子构成。原子可以是如下的形式之一：

- $s \in r$，其中 s 是元组变量而 r 是关系(我们不允许使用 $\notin$ 运算符)。
- $s[x] \Theta u[y]$，其中 s 和 u 是元组变量，x 是 s 所基于的关系模式中的一个属性，y 是 u 所基于的关系模式中的一个属性，Θ 是比较运算符($<$，$\leqslant$，$=$，$\neq$，$>$，$\geqslant$)；我们要求属性 x 和 y 所属域的成员能用 Θ 比较。
- $s[x] \Theta c$，其中 s 是元组变量，x 是 s 所基于的关系模式中的一个属性，Θ 是比较运算符，c 是属性 x 所属域中的常量。

我们根据如下规则，用原子构造公式：

- 原子是公式。
- 如果 P_1 是公式，那么 $\neg P_1$ 和 (P_1) 也都是公式。
- 如果 P_1 和 P_2 是公式，那么 $P_1 \vee P_2$、$P_1 \wedge P_2$ 和 $P_1 \Rightarrow P_2$ 也都是公式。
- 如果 $P_1(s)$ 是包含自由元组变量 s 的公式，且 r 是关系，则

  $$\exists s \in r(P_1(s)) \text{ 和 } \forall s \in r(P_1(s))$$

  也都是公式。

正如关系代数中一样，我们也可以写出一些形式上不一样的等价表达式。在元组关系演算中，这种等价性包括如下三条规则：

1. $P_1 \wedge P_2$ 等价于 $\neg(\neg(P_1) \vee \neg(P_2))$。
2. $\forall t \in r(P_1(t))$ 等价于 $\neg \exists t \in r(\neg P_1(t))$。
3. $P_1 \Rightarrow P_2$ 等价于 $\neg(P_1) \vee P_2$。 243

6.2.3 表达式的安全性

最后还要讨论一个问题。元组关系演算表达式可能产生一个无限的关系。例如如果我们写出表达式

$$\{t \mid \neg(t \in instructor)\}$$

不在 *instructor* 中的元组有无限多个，大多数这样的元组所包含的值甚至根本不在数据库中！显然，我们不希望有这样的表达式。

为了帮助我们对元组关系演算进行限制，引入了元组关系公式 P 的域(domain)这一概念。直观地说，P 的域(用 $dom(P)$ 表示)是 P 所引用的所有值的集合。它们既包括 P 自身用到的值，又包括 P 中涉及的关系的元组中出现的所有值。因此，P 的域是 P 中显式出现的值及名称出现在 P 中的那些关系的所有值的集合。例如，$dom(t \in instructor \wedge t[salary] > 80\,000)$ 是包括 80 000 和出现在 *instructor* 中的所

有值的集合。另外，$dom(\neg(t \in instructor))$是 *instructor* 中出现的所有值的集合，因为关系 *instructor* 在表达式中出现。

如果出现在表达式$\{t \mid P(t)\}$结果中的所有值均来自 $dom(P)$，则我们说表达式$\{t \mid P(t)\}$是安全的。表达式$\{t \mid \neg(t \in instructor)\}$不安全，因为 $dom(\neg(t \in instructor))$是所有出现在 *instructor* 中的值的集合，但是很可能有某个不在 *instructor* 中的元组 t，它包含不在 *instructor* 中出现的值。我们在这一节中所写的其他元组关系演算表达式都是安全的。

像$\{t \mid \neg(t \in instructor)\}$这样的不安全表达式中的元组个数是无限的，而安全的表达式一定包含有限的结果。因此我们限定只有那些安全的元组关系演算表达式才被认为是允许的。

6.2.4 语言的表达能力

限制在安全表达式范围内的元组关系演算和基本关系代数(包括运算符∪、－、×、σ、∏和ρ，但是没有扩展的关系代数运算符，例如广义投影和聚集($\mathcal{G}$))具有相同的表达能力。因此，对于每个只运用基本操作的关系代数表达式，都有与之等价的元组关系演算表达式；对于每个元组关系演算表达式，也都有与之等价的关系代数表达式。我们在此不证明这个断言，文献注解中有关于此证明的参考资料。在习题里包含了部分的证明。注意，没有任何一个元组关系演算等价于聚集运算，但是可以
244 对它进行扩展以支持聚集。扩展元组关系演算来处理算术表达式是很简单的。

6.3 域关系演算

关系演算的另一种形式称为域关系演算(domain relational calculus)，使用从属性域中取值的域变量，而不是整个元组的值。尽管如此，域关系演算同元组关系演算联系紧密。

就像关系代数是 SQL 语言的基础一样，域关系演算是被广泛采用的 QBE 语言(参考附录 B.1)的理论基础。

6.3.1 形式化定义

域关系演算中的表达式形式如下：

$$\{ < x_1, x_2, \cdots, x_n > \mid P(x_1, x_2, \cdots, x_n) \}$$

其中 x_1，x_2，…，x_n 代表域变量。和元组关系演算的情况一样，P 代表由原子构成的公式。域关系演算中的原子具有如下形式之一：

- $< x_1, x_2, \cdots, x_n > \in r$，其中 r 是 n 个属性上的关系，而 x_1，x_2，…，x_n是域变量或域常量。
- $x \Theta y$，其中 x 和 y 是域变量，而 Θ 是比较运算符($<$，$\leqslant$，$=$，$\neq$，$>$，$\geqslant$)。我们要求属性 x 和 y 所属域可用 Θ 比较。
- $x \Theta c$，其中 x 是域变量，Θ 是比较运算符，而 c 是 x 作为域变量的那个属性域中的常量。

根据以下规则，我们用原子构造公式：

- 原子是公式。
- 如果 P_1 是公式，那么 $\neg P_1$ 和 (P_1) 也都是公式。
- 如果 P_1 和 P_2 是公式，那么 $P_1 \vee P_2$，$P_1 \wedge P_2$ 和 $P_1 \Rightarrow P_2$ 也都是公式。
- 如果 $P_1(x)$是 x 的一个公式，其中 x 是自由域变量，则

$$\exists x(P_1(x)) \text{ 和 } \forall x(P_1(x))$$

也都是公式。

245 我们把 $\exists a,b,c(P(a,b,c))$ 作为 $\exists a(\exists b(\exists c(P(a,b,c))))$ 的简写。

6.3.2 查询的例子

我们现在用域关系演算对前面讲到的例子给出查询表达式。请注意这些表达式和元组关系演算表达式的相似之处。

- 找出工资在 80 000 美元以上的教师的 *ID*、*name*、*dept_name* 和 *salary*：

$$\{ < i,n,d,s > \mid < i,n,d,s > \in instructor \wedge s > 80000 \}$$

- 找出工资大于 80 000 美元的所有教师的姓名：

$$\{ < i > \mid \exists n,d,s(< i,n,d,s > \in instructor \wedge s > 80000)\}$$

虽然第二个查询看起来同我们在元组关系演算中书写的很相似，但实际上两者有重要区别。在元组演算中，当我们对某个元组变量 s 写 $\exists s$ 时，我们立刻通过 $\exists s \in r$ 将它同某个关系挂钩。可是，当我们在域演算中写 $\exists n$ 时，n 不指代一个元组，而指的是一个域值。因此，在用子公式 $< i, n, d, s > \in instructor$ 将 n 束缚到 *instructor* 关系中的教师以前，变量 n 的域是不受约束的。

下面我们给出一些域关系演算的查询示例：

- 找出在物理系的所有教师姓名，以及他们教授的所有课程的 *course_id*：

$$\{ < n,c > \mid \exists i,a,se,y(< i,c,a,s,y > \in teaches \wedge \exists d,s(< i,n,d,s > \in instructor \wedge d = "Physics"))\}$$

- 找出在 2009 年秋季学期或者 2010 年春季学期或者这两学期都开设的所有课程的集合：

$$\{ < c > \mid \exists a,s,y,b,r,t(< c,a,s,y,b,r,t > \in section \wedge s = "Fall" \wedge y = "2009") \vee \exists a,s,y,b,r,t(< c,a,s,y,b,r,t > \in section \wedge s = "Spring" \wedge y = "2010")\}$$

- 找出选了生物系开设的全部课程的所有学生：

$$\{ < i > \mid \exists n,d,tc(< i,n,d,tc > \in student) \wedge \forall ci,ti,dn,cr(< ci,ti,dn,cr > \in course \wedge dn = "Biology" \Rightarrow \exists si,se,y,g(< i,ci,si,se,y,g > \in takes))\}$$

246

注意，如果生物系没开设任何课程，那么结果将包含所有学生，这与元组关系演算中的示例相一致。

6.3.3 表达式的安全性

我们知道，在元组关系演算(6.2 节)中可能写出产生结果为无限关系的表达式。这使得我们为元组关系演算表达式定义安全性。域关系演算中也有类似的情况。像

$$\{ < i,n,d,s > \mid \neg(< i,n,d,s > \in instructor)\}$$

这样的表达式是不安全的，因为它允许在结果中出现不在表达式的域中的值。

在域关系演算中，我们还必须考虑“存在”和“对所有的”子句中公式的形式。我们来看表达式

$$\{ < x > \mid \exists y(< x,y > \in r) \wedge \exists z(\neg(< x,z > \in r) \wedge P(x,z))\}$$

其中 P 是涉及 x 和 z 的公式。对于公式的第一部分 $\exists y(< x,y > \in r)$，我们在测试时只需考虑 r 中的值。可是，对于公式的第二部分 $\exists z(\neg(< x,z > \in r) \wedge P(x,z))$，我们在测试时必须考虑不在 r 中出现的 z 值。由于所有关系都是有限的，不在 r 中出现的值无限多。因此，通常情况下如果不考虑 z 的取值有无限多种可能，我们就不能完成对第二部分的测试。事实上我们并不这样做。我们通过增加一些约束来限制上面这样的表达式。

在元组关系演算中，我们将所有存在量词修饰的变量限制在某个关系范围内。由于在域关系演算中还没有这样做，我们增加用于定义安全性的规则，以处理类似于上例中出现的情况。如果下列条件同时成立，我们认为表达式

$$\{ < x_1,x_2,\cdots,x_n > \mid P(x_1,x_2,\cdots,x_n)\}$$

是安全的：

1. 表达式的元组中所有值均来自 $dom(P)$。

2. 对每个形如 $\exists x(P_1(x))$ 的“存在”子公式而言，子公式为真当且仅当在 $dom(P_1)$ 中有某个值 x 使 $P_1(x)$ 为真。

3. 对每个形如 $\forall x(P_1(x))$ 的“对所有的”子公式而言，子公式为真当且仅当 $P_1(x)$ 对 $dom(P_1)$ 中所有值 x 均为真。

附加规则的目的是为了保证我们不需要测试无限多的可能性就可以完成对“存在”和“对所有的”子公式的测试。我们看安全性定义中的第二个规则。要使 $\exists x(P_1(x))$ 为真，我们只需找到一个 x 使 $P_1(x)$ 为真。通常，我们需要测试无数个值。但是，如果表达式是安全的，我们就知道可以只注意 $dom(P_1)$ 中的值。这种限制使我们考虑的元组减少到有限个。 247

形如 $\forall x(P_1(x))$ 的子公式情况类似。要断言 $\forall x(P_1(x))$ 为真，需要测试所有可能的值，因此必须检查无限多的值。跟前面一样，如果我们知道表达式是安全的，则只需要对 $dom(P_1)$ 中的值来测试 $P_1(x)$ 就已经足够了。

除了前面我们介绍的不安全查询的例子之外，本节例子中所给出的域关系演算表达式都是安全的。

6.3.4 语言的表达能力

当我们把域关系演算限制在安全表达式范围内时，它与被限制在安全表达式范围内的元组关系演算具有相同的表达能力。我们在前面已经知道受限的元组关系演算与关系代数等价，所以下述三者都是等价的：

- 基本关系代数(不包括扩展关系代数运算)。
- 限制在安全表达式范围内的元组关系演算。
- 限制在安全表达式范围内的域关系演算。

注意没有任何一个域关系演算等价于聚集运算，但是它可以扩展以支持聚集，并且扩展它来处理算术表达式是很简单的。

6.4 总结

- **关系代数**(relational algebra)定义了一套在表上运算且输出结果也是表的代数运算。这些运算可以混合使用来得到表达所希望查询的表达式。关系代数定义了关系查询语言中使用的基本运算。
- 关系代数运算可以分为：
  - □ 基本运算。
  - □ 附加的运算，可以用基本运算表达。
  - □ 扩展的运算，其中的一些扩展了关系代数的表达能力。
- 关系代数是一种简洁的、形式化的语言，不适合于那些偶尔使用数据库系统的用户。因此，商用数据库系统采用有更多"语法修饰"的语言。从第 3 章到第 5 章我们讨论了最有影响力的语言——**SQL**，它是基于关系代数的。 [248]
- **元组关系演算**(tuple relational calculus)和**域关系演算**(domain relational calculus)是非过程化语言，代表了关系查询语言所需的基本能力。基本关系代数是一种过程化语言，在能力上等价于被限制在安全表达式范围内的关系演算的这两种形式。
- 关系演算是简洁的、形式化的语言，并不适合于那些偶尔使用数据库系统的用户。这两种形式化语言构成了两种更易使用的语言 QBE 和 Datalog 的基础，我们将在附录 B 中介绍它们。

术语回顾

- 关系代数
- 关系代数运算
  - □ 选择 σ
  - □ 投影 Π
  - □ 并 $\cup$
  - □ 集合差 $-$
  - □ 笛卡儿积 $\times$
  - □ 更名 ρ
- 附加的运算
  - □ 集合交 $\cap$
  - □ 自然连接 $\bowtie$
  - □ 赋值运算
  - □ 外连接
    - 左外连接 $⟕$
    - 右外连接 $⟖$
    - 全外连接 $⟗$
- 多重集
- 分组
- 空值
- 元组关系演算
- 域关系演算
- 表达式安全性
- 语言的表达能力

实践习题

6.1 使用大学的模式，用关系代数来表达下面这些查询。

a. 找出计算机(Comp. Sci.)系开设的占 3 个学分的课程的名称。

b. 找出选了教师 Einstein 所教课程的所有学生的 ID，注意结果中不能有重复。

c. 找出所有教师的最高工资。

d. 找出收入为最高工资的所有教师(可能有多个人工资相同)。
e. 找出 2009 年秋季学期开设的各个课程段的选课人数。
f. 找出 2009 年秋季学期开设的各个课程段中最大的选课人数。
g. 找出在 2009 年秋季学期选课人数最多的课程段。 249

6.2 考虑图 6-22 所示关系数据库，主码加了下划线。给出关系代数表达式来表示下列每一个查询：
a. 找出与其经理居住在同一城市同一街道的所有员工姓名。
b. 找出此数据库中不在“First Bank Corporation”工作的所有员工姓名。
c. 找出比“Small Bank Corporation”的所有员工收入都高的所有员工姓名。

employee(<u>*person_name*</u>, *street*, *city*)
works(<u>*person_name*</u>, *company_name*, *salary*)
company(<u>*company_name*</u>, *city*)
manages(<u>*person_name*</u>, *manager_name*)

图 6-22　习题 6.2、习题 6.8、习题 6.11、习题 6.13 和习题 6.15 的关系数据库

6.3 外连接是自然连接的一种扩展，它使得参与运算的关系中的元组在连接结果中不丢失。描述 theta 连接运算可以怎样扩展，使得在 theta 连接结果中来自左侧、右侧及两侧关系的元组不被丢失。

6.4 （**除运算**，division operation）关系代数中的除运算符是 ÷，它的定义如下：设 $r(R)$ 和 $s(S)$ 是两个关系，并且 $S \subseteq R$；即模式 S 中的每个属性都在模式 R 中。那么 $r \div s$ 是模式 $R-S$ 上的关系，即此模式中包含所有在 R 中而不在 S 中的属性。元组 t 属于 $r \div s$ 当且仅当以下两个条件同时成立：
1. t 在 $\Pi_{R-S}(r)$ 中。
2. 对 s 中的每一个元组 t_s，在 r 中都有元组 t_r 同时满足以下两个条件：
   a. $t_r[S] = t_s[S]$
   b. $t_r[R-S] = t$

有了以上定义，要求：
a. 用除运算符写一个关系代数表达式来找到选了计算机(Comp. Sci.)系课程的所有学生的 ID。(提示：在除之前，把 *takes* 投影到 ID 和 *course_id*，并用一个选择表达式生成所有计算机系课程 *course_id* 的集合。) 250
b. 如果不用除法，如何在关系代数中表达上述查询。(通过这样做，你就可以了解如何用其他关系代数运算来定义除运算。)

6.5 已知如下关系模式：

$$R = (A, B, C)$$
$$S = (D, E, F)$$

关系 $r(R)$ 和 $s(S)$ 已给出。请给出与下列每个表达式等价的元组关系演算表达式：
a. $\Pi_A(r)$
b. $\sigma_{B=17}(r)$
c. $r \times s$
d. $\Pi_{A,F}(\sigma_{C=D}(r \times s))$

6.6 设 $R = (A, B, C)$，r_1 和 r_2 都是模式 R 上的关系。请分别给出与下列表达式等价的域关系演算表达式：
a. $\Pi_A(r_1)$
b. $\sigma_{B=17}(r_1)$
c. $r_1 \cup r_2$
d. $r_1 \cap r_2$
e. $r_1 - r_2$
f. $\Pi_{A,B}(r_1) \bowtie \Pi_{B,C}(r_2)$

6.7 设 $R = (A, B)$，$S = (A, C)$，$r(R)$ 和 $s(S)$ 为关系。为下列查询写出关系代数表达式：
a. $\{ < a > \mid \exists (b(< a, b > \in r \wedge b = 7)\}$
b. $\{ < a, b, c > \mid < a, b > \in r \wedge < a, c > \in s)\}$

c. $\{ < a > \mid \exists c(< a, c > \in s \wedge \exists b_1, b_2(< a, b_1 > \in r \wedge < c, b_2 > \in r \wedge b_1 > b_2))\}$

6.8 考虑图 6-22 所示的关系数据库，其中主码用下划线标示。为下面每个查询写出元组关系演算表达式：

251 a. 找出所有直接在经理"Jones"下工作的员工。

b. 找出直接在经理"Jones"下工作的员工居住的所有城市。

c. 找出"Jones"的经理的经理的名字。

d. 找出比住在"Mumbai"的所有员工的收入都高的那些员工。

6.9 描述如何将 SQL 中的连接表达式转换成关系代数。

习题

6.10 使用大学的模式，用关系代数表达如下查询。

a. 找出所有至少选了一门计算机(Comp. Sci.)系课程的学生。

b. 找出所有在 2009 年春季之前没有选任何课程的学生的 ID 和姓名。

c. 对于每个系，找到该系教师的最高工资。可以假设每个系至少有一名教师。

d. 在上一问找到每个系的最高工资的基础上，找出所有各系的最高工资的最小值。

6.11 考虑图 6-22 所示关系数据库，主码加了下划线。给出关系代数表达式来表示下列每一个查询：

a. 找出 First Bank Corporation 的所有员工姓名。

b. 找出 First Bank Corporation 所有员工的姓名和居住城市。

c. 找出 First Bank Corporation 所有年收入在 10 000 美元以上的员工姓名和居住的街道、城市。

d. 找出所有居住地与工作的公司在同一城市的员工姓名。

e. 假设公司可以位于几个城市中。找出满足下面条件的所有公司，它位于 Small Bank Corporation 所位

252 于的每一个城市。

6.12 使用大学的例子，用关系代数查询来找出多于一个教师授课的课程段，分别用以下两种方式来表达：

a. 使用聚集函数。

b. 不使用聚集函数。

6.13 考虑图 6- 22 所示的关系数据库。分别给出下列查询的关系代数表达式：

a. 找出员工最多的公司。

b. 找出工资最少的员工所在公司。

c. 找出人均工资比 First Bank Corporation 人均工资高的公司。

6.14 考虑如下关于图书馆的关系模式：

$$memeber(\underline{memb\_no}, name, dob)$$
$$books(\underline{isbn}, title, authors, publisher)$$
$$borrowed(\underline{memb\_no}, \underline{isbn}, date)$$

用关系代数写出下列查询：

a. 找出借了任何由 McGraw-Hill 出版的书的成员的姓名。

b. 找出借了由 McGraw-Hill 出版的所有的书的成员的姓名。

c. 找出借了由 McGraw-Hill 出版的 5 本以上不同的书的成员的姓名和成员号。

d. 对每个出版商，找出借了该出版商的 5 本以上的书的成员的姓名和成员号。

e. 找出平均每个成员借了多少本书。下面的情况需要考虑在内，如果某个成员没有借任何书，那么他就根本不会出现在关系 *borrowed* 中。

6.15 考虑图 6-22 所示的员工数据库。为下面每个查询分别写出元组关系演算和域关系演算表达式：

a. 找出所有为 First Bank Corporation 工作的员工名字。

b. 找出所有为 First Bank Corporation 工作的员工名字和居住城市。

c. 找出所有为 First Bank Corporation 工作且年薪超过 10 000 美元的员工名字、居住街道和城市。

253 d. 找出居住城市和公司所在城市相同的所有员工。

e. 找出居住街道和城市与其经理相同的所有员工。

f. 找出数据库中所有不为 First Bank Corporation 工作的员工。

g. 找出工资高于 Small Bank Corporation 的每一位员工的员工。

h. 假设一个公司可以位于好几个城市。找出满足下面条件的所有公司，它位于 Small Bank Corporation 所位于的每一个城市。

6.16　设 $R=(A, B)$ 且 $S=(A, C)$，$r(R)$ 和 $s(S)$ 是关系。分别给出与下列域关系演算表达式等价的关系代数表达式：

a. $\{ < a > \mid \exists b(< a,b > \in r \wedge b = 17)\}$

b. $\{ < a,b,c > \mid < a,b > \in r \wedge < a,c > \in s\}$

c. $\{ < a > \mid \exists b(< a,b > \in r) \vee \forall c(\exists d(< d,c > \in s) \Rightarrow < a,c > \in s)\}$

d. $\{ < a > \mid \exists c(< a,c > \in s \wedge \exists b_1,b_2(< a,b_1 > \in r \wedge < c,b_2 > \in r \wedge b_1 > b_2))\}$

6.17　用 SQL 查询代替关系代数表达式来重做习题 6.16。

6.18　设 $R=(A, B)$ 且 $S=(A, C)$，$r(R)$ 和 $s(S)$ 是关系。使用特殊常量 *null*，分别书写等价于下列表达式的元组关系演算表达式：

a. $r \bowtie\!\!\!\subset s$

b. $r \supset\!\!\!\bowtie\!\!\!\subset s$

c. $r \supset\!\!\!\bowtie s$

6.19　给出元组关系表达式，找出关系 $r(A)$ 中的最大值。

文献注释

关系代数最初的定义在 Codd[1970] 中给出；关系模型的扩展、关于在关系代数中引入空值的论述（RM/T 模型），以及外连接，这些内容都出现在 Codd[1979] 中。Codd[1990] 是 E. F. Codd 关于关系模型的所有文章的一个概括。外连接在 Date[1993b] 中也进行了讨论。 254

元组关系演算最初的定义在 Codd[1972] 中给出。元组关系演算和关系代数等价性的形式化证明在 Codd[1972] 中给出。关系演算的一些扩展已经被提出。Klug[1982] 和 Escobar-Molano 等 [1993] 描述了向分级聚集函数的扩展。 255 ~ 256