第2章
Database System Concepts, 6E

关系模型介绍

在商用数据处理应用中，关系模型已经成为当今主要的数据模型。之所以占据主要位置，是因为和早期的数据模型如网络模型或层次模型相比，关系模型以其简易性简化了编程者的工作。

本章我们先学习关系模型的基础知识。关系数据库具有坚实的理论基础，我们在第6章学习关系数据库理论中与查询相关的部分，从第7章到第8章我们将考察其中用于关系数据库模式设计的部分，在第12章和第13章我们将讨论高效处理查询的理论。

2.1 关系数据库的结构

关系数据库由表(table)的集合构成，每个表有唯一的名字。例如，图2-1中的 *instructor* 表记录了有关教师的信息，它有四个列首：*ID*、*name*、*dept_name* 和 *salary*。该表中每一行记录了一位教师的信息，包括该教师的 *ID*、*name*、*dept_name* 以及 *salary*。类似地，图2-2中的 *course* 表存放了关于课程的信息，包括每门课程的 *course_id*、*title*、*dept_name* 和 *credits*。注意，每位教师通过 *ID* 列的取值进行标识，而每门课程则通过 *course_id* 列的取值来标识。

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2-1 *instructor* 关系

图2-3给出的第三个表是 *prereq*，它存放了每门课程的先修课程信息。该表具有 *course_id* 和 *prereq_id* 两列，每一行由一个课程对组成，这个课程对表示了第二门课程是第一门课程的先修课。

由此，*prereq* 表中的每行表示了两门课程之间的联系：其中一门课程是另一门课程的先修课。作为另一个例子，我们考察 *instructor* 表，表中的行可被认为是代表了从一个特定的 *ID* 到相应的 *name*、*dept_name* 和 *salary* 值之间的联系。

一般说来，表中一行代表了一组值之间的一种联系。由于一个表就是这种联系的一个集合，表这个概念和数学上的关系这个概念是密切相关的，这也正是关系数据模型名称的由来。在数学术语中，元组(tuple)只是一组值的序列(或列表)。在 n 个值之间的一种联系可以在数学上用关于这些值的一个 n 元组(n-tuple)来表示，换言之，n 元组就是一个有 n 个值的元组，它对应于表中的一行。

course_id	*title*	*dept_name*	*credits*
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

图2-2 *course* 关系

这样，在关系模型的术语中，**关系**(relation)用来指代表，而元组(tuple)用来指代行。类似地，**属性**(attribute)指代的是表中的列。

考察图 2-1，我们可以看出 *instructor* 关系有四个属性：*ID*、*name*、*dept_name* 和 *salary*。

BIO-301	BIO-101
BIO-399	BIO-101
CS-190	CS-101
CS-315	CS-101
CS-319	CS-101
CS-347	CS-101
EE-181	PHY-101

图 2-3　*prereq* 关系

我们用关系实例(relation instance)这个术语来表示一个关系的特定实例，也就是所包含的一组特定的行。图 2-1 所示的 *instructor* 的实例有 12 个元组，对应于 12 个教师。

本章我们将使用多个不同的关系来说明作为关系数据模型基础的各种概念。这些关系代表一个大学的一部分。它们并没有包含真实的大学数据库中的所有数据，这主要是为了简化表示。在第 7 章和第 8 章里我们将详细讨论如何判断适当的关系结构的相关准则。

由于关系是元组集合，所以元组在关系中出现的顺序是无关紧要的。因此，无论关系中的元组是像图 2-1 那样被排序后列出，还是像图 2-4 那样无序的，都没有关系；在上述两图中的关系是一样的，因为它们具有同样的元组集合。为便于说明，当我们在显示关系时，大多数情况下都按其第一个属性排序。

对于关系的每个属性，都存在一个允许取值的集合，称为该属性的**域**(domain)。这样 *instructor* 关系的 *salary* 属性的域就是所有可能的工资值的集合，而 *name* 属性的域是所有可能的教师名字的集合。

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

图 2-4　*instructor* 关系的无序显示

我们要求对所有关系 r 而言，r 的所有属性的域都是原子的。如果域中元素被看作是不可再分的单元，则域是**原子的**(atomic)。例如，假设 *instructor* 表上有一个属性 *phone_number*，它存放教师的一组联系电话号码。那么 *phone_number* 的域就不是原子的，因为其中的元素是一组电话号码，是可以被再分为单个电话号码这样的子成分的。

重要的问题不在于域本身是什么，而在于我们怎样在数据库中使用域中元素。现在假设 *phone_number* 属性存放单个电话号码。即便如此，如果我们把电话号码的属性值拆分成国家编号、地区编号以及本地号码，那么我们还是把它作为非原子值来对待。如果我们把每个电话号码作为不可再分的单元，那么 *phone_number* 属性才会有原子的域。

在本章，以及第 3 章～第 6 章，我们假设所有属性的域都是原子的。在第 22 章中，我们将讨论对关系数据模型进行扩展以便允许非原子域。

空(null)值是一个特殊的值，表示值未知或不存在。如前所述，如果我们在关系 *instructor* 中包括属性 *phone_number*，则可能某教师根本没有电话号码，或者电话号码未提供。这时我们就只能使用空值来强调该值未知或不存在。以后我们会看到，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，然后在 3.6 节中我们将描述空值对不同操作的影响。

2.2　数据库模式

当我们谈论数据库时，我们必须区分**数据库模式**(database schema)和**数据库实例**(database instance)，前者是数据库的逻辑设计，后者是给定时刻数据库中数据的一个快照。

关系的概念对应于程序设计语言中变量的概念，而**关系模式**(relation schema)的概念对应于程序设计语言中类型定义的概念。

一般说来，关系模式由属性序列及各属性对应域组成。等第 3 章讨论 SQL 语言时，我们才去关心每个属性的域的精确定义。

关系实例的概念对应于程序设计语言中变量的值的概念。给定变量的值可能随时间发生变化；类似地，当关系被更新时，关系实例的内容也随时间发生了变化。相反，关系的模式是不常变化的。

尽管知道关系模式和关系实例的区别非常重要，我们常常使用同一个名字，比如 *instructor*，既指

代模式，也指代实例。在需要的时候，我们会显示地指明模式或实例。例如“*instructor* 模式”或“*instructor* 关系的一个实例”。然而，在模式或实例的含义清楚的情况下，我们就简单地使用关系的名字。

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

图 2-5 *department* 关系

考察图 2-5 中的 *department* 关系，该关系的模式是：

department (*dept_name*, *building*, *budget*)

请注意属性 *dept_name* 既出现在 *instructor* 模式中，又出现在 *department* 模式中。这样的重复并不是一种巧合。实际上，在关系模式中使用相同属性正是将不同关系的元组联系起来的一种方法。例如，假设我们希望找出在 Watson 大楼工作的所有教师的相关信息。我们首先在 *department* 关系中找出所有位于 Watson 的系的 *dept_name*。接着，对每一个这样的系，我们在 *instructor* 关系中找出与 *dept_name* 对应的教师信息。

我们继续看大学数据库的例子。

大学里的每门课程可能要讲授多次，可以在不同学期授课，甚至可能在同一个学期授课。我们需要一个关系来描述每次课的授课情况或分段情况。该关系模式为：

section (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *time_slot_id*)

图 2-6 给出了 *section* 关系的一个示例。

我们需要一个关系来描述教师和他们所讲授的课程段之间的联系。描述此联系的关系模式是：

43

teaches (*ID*, *course_id*, *sec_id*, *semester*, *year*)

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 2-6 *section* 关系

图 2-7 给出了 *teaches* 关系的一个示例。

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 2-7 *teaches* 关系

正如你可以料想的，在一个真正的大学数据库中还维护了更多的关系。除了我们已经列出的这些关系：*instructor*、*department*、*course*、*section*、*prereq* 和 *teaches*，在本书中我们还要使用下列关系：

- *student* (*ID*, *name*, *dept_name*, *tot_cred*)
- *advisor* (*s_id*, *i_id*)
- *takes* (*ID*, *course_id*, *sec_id*, *semester*, *year*, *grade*)
- *classroom* (*building*, *room_number*, *capacity*)
- *time_slot* (*time_slot_id*, *day*, *start_time*, *end_time*)

2.3 码

我们必须有一种能区分给定关系中的不同元组的方

法。这用它们的属性来表明。也就是说，一个元组的属性值必须是能够唯一区分元组的。换句话说，一个关系中没有两个元组在所有属性上的取值都相同。

超码(superkey)是一个或多个属性的集合，这些属性的组合可以使我们在一个关系中唯一地标识一个元组。例如，*instructor* 关系的 *ID* 属性足以将不同的教师元组区分开来，因此，*ID* 是一个超码。另一方面，*instructor* 的 *name* 属性却不是一个超码，因为几个教师可能同名。

形式化地描述，设 R 表示关系 r 模式中的属性集合。如果我们说 R 的一个子集 K 是 r 的一个超码，则限制了关系 r 中任意两个不同元组不会在 K 的所有属性上取值完全相等，即如果 t_1 和 t_2 在 r 中且 $t_1 \neq t_2$，则 $t_1.K \neq t_2.K$。

超码中可能包含无关紧要的属性。例如，*ID* 和 *name* 的组合是关系 *instructor* 的一个超码。如果 K 是一个超码，那么 K 的任意超集也是超码。我们通常只对这样的一些超码感兴趣，它们的任意真子集都不能成为超码。这样的最小超码称为**候选码**(candidate key)。

几个不同的属性集都可以做候选码的情况是存在的。假设 *name* 和 *dept_name* 的组合足以区分 *instructor* 关系的各个成员，那么{*ID*}和{*name*, *dept_name*}都是候选码。虽然属性 *ID* 和 *name* 一起能区分 *instructor* 元组，但它们的组合{*ID*, *name*}并不能成为候选码，因为单独的属性 *ID* 已是候选码。

我们用**主码**(primary key)这个术语来代表被数据库设计者选中的、主要用来在一个关系中区分不同元组的候选码。码(不论是主码、候选码或超码)是整个关系的一种性质，而不是单个元组的性质。关系中的任意两个不同的元组都不允许同时在码属性上具有相同的值。码的指定代表了被建模的事物在现实世界中的约束。

主码的选择必须慎重。正如我们所注意到的那样，人名显然是不足以作主码的，因为可能有多个人重名。在美国，人的社会保障号可以作候选码。而非美国居民通常不具有社会保障号，所以跨国企业必须设置他们自己的唯一标识符。另外也可以使用另一些属性的唯一组合作为码。

44
~
45

主码应该选择那些值从不或极少变化的属性。例如，一个人的地址就不应该作为主码的一部分，因为它很可能变化。另一方面，社会保障号却可以保证决不变化。企业产生的唯一标识符通常不变，除非两个企业合并了，这种情况下可能在两个公司中会使用相同的标识符，因此需要重新分配标识符以确保其唯一性。

习惯上把一个关系模式的主码属性列在其他属性前面；例如，*department* 中的 *dept_name* 属性最先列出，因为它是主码。主码属性还加上了下划线。

一个关系模式(如 r_1)可能在它的属性中包括另一个关系模式(如 r_2)的主码。这个属性在 r_1 上称作参照 r_2 的**外码**(foreign key)。关系 r_1 也称为外码依赖的**参照关系**(referencing relation)，r_2 叫做外码的**被参照关系**(referenced relation)。例如，*instructor* 中的 *dept_name* 属性在 *instructor* 上是外码，它参照 *department*，因为 *dept_name* 是 *department* 的主码。在任意的数据库实例中，从 *instructor* 关系中任取一个元组，比如 t_a，在 *department* 关系中必定存在某个元组，比如 t_b，使得 t_a 在 *dept_name* 属性上的取值与 t_b 在主码 *dept_name* 上的取值相同。

现在考察 *section* 和 *teaches* 关系。如下需求是合理的：如果一门课程是分段授课的，那么它必须至少由一位教师来讲授；当然它可能由不止一位教师来讲授。为了施加这种约束，我们需要保证如果一个特定的(*course_id*, *sec_id*, *semester*, *year*)组合出现在 *section* 中，那么该组合也必须出现在 *teaches* 中。可是，这组值并不构成 *teaches* 的主码，因为不止一位教师可能讲授同一个这样的课程段。其结果是，我们不能声明从 *section* 到 *teaches* 的外码约束(尽管我们可以在相反的方向上声明从 *teaches* 到 *section* 的外码约束)。

从 *section* 到 *teaches* 的约束是**参照完整性约束**(referential integrity constraint)的一个例子。参照完整性约束要求在参照关系中任意元组在特定属性上的取值必然等于被参照关系中某个元组在特定属性上的取值。

2.4 模式图

一个含有主码和外码依赖的数据库模式可以用**模式图**(schema diagram)来表示。图 2-8 展示了我们

大学组织的模式图。每一个关系用一个矩形来表示，关系的名字显示在矩形上方，矩形内列出各属性。主码属性用下划线标注。外码依赖用从参照关系的外码属性到被参照关系的主码属性之间的箭头来表示。

46

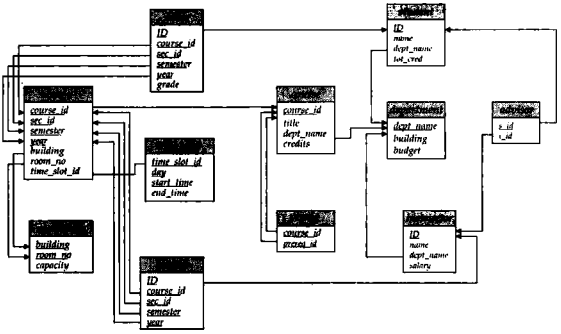

图 2-8 大学数据库的模式图

除外码约束之外，模式图中没有显示表示出参照完整性约束。在后面第 7 章，我们将学习一种不同的、称作实体 - 联系图的图形化表示。实体 - 联系图有助于我们表示几种约束，包括通用的参照完整性约束。

很多数据库系统提供图形化用户界面设计工具来建立模式图。我们将在第 7 章详细讨论模式的图形化表示。

在后面的章节中我们使用大学作为例子。图 2-9 给出了我们在例子中使用的关系模式，其中主码属性被标上了下划线。正如我们将在第 3 章中看到的一样，这对应于在 SQL 的数据定义语言中定义关系的方法。

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*)

图 2-9 大学数据库模式

2.5 关系查询语言

查询语言(query language)是用户用来从数据库中请求获取信息的语言。这些语言通常比标准的程序设计语言层次更高。查询语言可以分为过程化的和非过程化的。在**过程化语言**(procedural language)中，用户指导系统对数据库执行一系列操作以计算出所需结果。在**非过程化语言**(nonprocedural language)中，用户只需描述所需信息，而不用给出获取该信息的具体过程。

实际使用的查询语言既包含过程化方式的成分，又包含非过程化方式的成分。我们从第 3 章到第 5 章学习被广泛应用的查询语言 SQL。

有一些“纯”查询语言：关系代数是过程化的，而元组关系演算和域关系演算是非过程化的。这些语言简洁且形式化，默认商用语言的“句法修饰”，但它们说明了从数据库中提取数据的基本技术。在

第 6 章，我们详细研究关系代数和关系演算的两种形式，即元组关系演算和域关系演算。关系代数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。关系演算使用谓词逻辑来定义所需的结果，但不需给出获取结果的特定代数过程。

2.6　关系运算

所有的过程化关系查询语言都提供了一组运算，这些运算要么施加于单个关系上，要么施加于一对关系上。这些运算具有一个很好的，并且也是所需的性质：运算结果总是单个的关系。这个性质使得人们可以模块化的方式来组合几种这样的运算。特别是，由于关系查询的结果本身也是关系，所以关系运算可施加到查询结果上，正如施加到给定关系集上一样。

在不同语言中，特定的关系运算的表示是不同的，但都符合我们在本章所描述的通用结构。在第 3 章，我们将给出在 SQL 中表达关系运算的特殊方式。

最常用的关系运算是从单个关系(如 *instructor*)中选出满足一些特定谓词(如 *salary* > 85 000 美元)的特殊元组。其结果是一个新关系，它是原始关系(*instructor*)的一个子集。例如，如果我们从图 2-1 的 *instructor* 关系中选择满足谓词"工资大于 85 000 美元"的元组，我们得到的结果如图 2-10 所示。

47 / 48

ID	*name*	*dept_name*	*salary*
12121	Wu	Finance	90000
22222	Einstein	Physics	95000
33456	Gold	Physics	87000
83821	Brandt	Comp. Sci.	92000

图 2-10　选择工资大于 85 000 美元的 *instructor* 元组的查询结果

另一个常用的运算是从一个关系中选出特定的属性(列)。其结果是一个只包含那些被选择属性的新关系。例如，假设我们从图 2-1 的 *instructor* 关系中只希望列出教师的 *ID* 和工资，但不列出 *name* 和 *dept_name* 的值，那么其结果有 *ID* 和 *salary* 两个属性，如图 2-11 所示。结果中的每个元组都是从 *instructor* 关系中的某个元组导出的，不过只具有被选中的属性。

ID	*salary*
10101	65000
12121	90000
15151	40000
22222	95000
32343	60000
33456	87000
45565	75000
58583	62000
76543	80000
76766	72000
83821	92000
98345	80000

图 2-11　从 *instructor* 关系中选取属性 *ID* 和 *salary* 的查询结果

连接运算可以通过下述方式来结合两个关系：把分别来自两个关系的元组对合并成单个元组。有几种不同的方式来对关系进行连接(正如我们将在第 3 章中看到的)。图 2-12 显示了一个连接来自 *instructor* 和 *department* 表中元组的例子，新元组给出了有关每个教师及其工作所在系的信息。此结果是通过把 *instructor* 关系中的每个元组和 *department* 关系中对应于教师所在系的元组合并形成的。

图 2-12 所示的连接被称作自然连接，在这种连接形式中，对于来自 *instructor* 关系的一个元组与 *department* 关系中的一个元组来说，如果它们在 *dept_name* 属性上的取值相同，那它们就是匹配的。所有这样匹配的元组对都会在连接结果中出现。通常说来，两个关系上的自然连接运算所匹配的元组在两个关系共有的所有属性上取值相同。

笛卡儿积运算从两个关系中合并元组，但不同于连接运算的是，其结果包含来自两个关系元组的所有对，无论它们的属性值是否匹配。

ID	*name*	*salary*	*dept_name*	*building*	*budget*
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
12121	Wu	90000	Finance	Painter	120000
15151	Mozart	40000	Music	Packard	80000
22222	Einstein	95000	Physics	Watson	70000
32343	El Said	60000	History	Painter	50000
33456	Gold	87000	Physics	Watson	70000
45565	Katz	75000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
76543	Singh	80000	Finance	Painter	120000
76766	Crick	72000	Biology	Watson	90000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000

图 2-12　*instructor* 关系和 *department* 关系的自然连接结果

因为关系是集合，所以我们可以在关系上施加标准的集合运算。并运算在两个“相似结构”的表（比如一个所有毕业生的表和一个所有大学生的表）上执行集合并。例如，我们可以得到一个系中所有学生的集合。另外的集合运算如交和集合差也都可以被执行。

正如我们此前讲到的，我们可以在查询结果上施加运算。例如，如果我们想找出工资超过85 000美元的那些教师的 *ID* 和 *salary*，我们可以执行上述例子中的前两种运算。首先我们从 *instructor* 关系中选出 *salary* 值大于85 000美元的元组，然后从结果中选出 *ID* 和 *salary* 两个属性，结果关系如图2-13所示，由 *ID* 和 *salary* 构成。在此例中，我们可以任一次序来执行运算，但正如我们将看到的，并非在所有情况下均可如此。

有时侯，查询结果中包含重复的元组。例如，如果我们从 *instructor* 关系中选出 *dept_name* 属性，就会有好几种重复的情况，其中包括“Comp. Sci.”，它出现了三次。一些关系语言严格遵守集合的数学定义，去除了重复。另一些考虑到从大的结果关系中去除重复需要大量相关的处理，就保留了重复。在后面这类情况中，关系并非是纯粹数学意义上的真正关系。

ID	*salary*
12121	90000
22222	95000
33456	87000
83821	92000

图2-13 选择工资大于85 000美元的教师的 *ID* 和 *salary* 属性的结果

当然，数据库中的数据是随时间而改变的。关系可以随新元组的插入、已有元组的删除或更改元组在特定属性上的值而更新。整个关系可被删除，新的关系可被创建。

从第3章到第5章，我们将讨论如何使用SQL语言来表示关系的查询和更新。

关系代数

关系代数定义了在关系上的一组运算，对应于作用在数字上的普通代数运算，如加法、减法或乘法。正如作用在数字上的代数运算以一个或多个数字作为输入，返回一个数字作为输出，关系代数运算通常以一个或两个关系作为输入，返回一个关系作为输出。

第6章将详细介绍关系代数，下面我们给出几个运算的概述：

符号（名字）	使用示例
σ（选择）	$\sigma_{salary>=85\,000}(instructor)$ 返回输入关系中满足谓词的行
Π（投影）	$\Pi_{ID,salary}(instructor)$ 对输入关系的所有行输出指定的属性。从输出中去除重复元组
⋈（自然连接）	$instructor \bowtie department$ 从两个输入关系中输出这样的元组对：它们在具有相同名字的所有属性上取值相同
×（笛卡儿积）	$instructor \times department$ 从两个输入关系中输出所有的元组对（无论它们在共同属性上的取值是否相同）
∪（并）	$\Pi_{name}(instructor) \cup \Pi_{name}(student)$ 输出两个输入关系中元组的并

2.7 总结

- **关系数据模型**（relational data model）建立在表的集合的基础上。数据库系统的用户可以对这些表进行查询，可以插入新元组、删除元组以及更新（修改）元组。表达这些操作的语言有几种。
- 关系的**模式**（schema）是指它的逻辑设计，而关系的**实例**（instance）是指它在特定时刻的内容。数据库的模式和实例的定义是类似的。关系的模式包括它的属性，还可能包括属性类型和关系上的约束，比如主码和外码约束。

- 关系的**超码**(superkey)是一个或多个属性的集合，这些属性上的取值保证可以唯一识别出关系中的元组。候选码是一个最小的超码，也就是说，它是一组构成超码的属性集，但这组属性的任意子集都不是超码。关系的一个候选码被选作**主码**(primary key)。
- 在参照关系中的**外码**(foreign key)是这样的一个属性集合：对于参照关系中的每个元组来说，它在外码属性上的取值肯定等于被参照关系中某个元组在主码上的取值。
- **模式图**(schema diagram)是数据库中模式的图形化表示，它显示了数据库中的关系，关系的属性、主码和外码。
- **关系查询语言**(relational query language)定义了一组运算集，这些运算可作用于表上，并输出表作为结果。这些运算可以组合成表达式，表达所需的查询。
- **关系代数**(relational algebra)提供了一组运算，它们以一个或多个关系为输入，返回一个关系作为输出。诸如 SQL 这样的实际查询语言是基于关系代数的，但增加了一些有用的句法特征。

术语回顾

- 表
- 关系
- 元组
- 空值
- 数据库模式
- 数据库实例
- 关系模式
- 关系实例
- 码
  - 超码
  - 候选码
  - 主码
- 外码
  - 参照关系
  - 被参照关系
- 属性
- 域
- 原子域
- 参照完整性约束
- 模式图
- 查询语言
  - 过程化语言
  - 非过程化语言
- 关系运算
  - 选择元组
  - 选择属性
  - 自然连接
  - 笛卡儿积
  - 集合运算
- 关系代数

实践习题

2.1 考虑图 2-14 所示关系数据库。这些关系上适当的主码是什么？

2.2 考虑从 *instructor* 的 *dept_name* 属性到 *department* 关系的外码约束，给出对这些关系的插入和删除示例，使得它们破坏外码约束。

2.3 考虑 *time_slot* 关系。假设一个特定的时间段可以在一周之内出现多次，解释为什么 *day* 和 *start_time* 是该关系主码的一部分，而 *end_time* 却不是。

2.4 在图 2-1 所示 *instructor* 的实例中，没有两位教师同名。我们是否可以据此断定 *name* 可用来作为 *instructor* 的超码(或主码)？

2.5 先执行 *student* 和 *advisor* 的笛卡儿积，然后在结果上执行基于谓词 $s\_id = \text{ID}$ 的选择运算，最后的结果是什么？(采用关系代数的符号表示，此查询可写作 $\sigma_{s\_id = ID}(student \times advisor)$。)

employee(person-name, *street*, *city*)
works(person-name, *company-name*, *salary*)
company(*company-name*, *city*)

图 2-14 习题 2.1、习题 2.7 和习题 2.12 的关系数据库

2.6 考虑下面的表达式，哪些使用了关系代数运算的结果来作为另一个运算的输入？对于每个表达式，说明表达式的含义。

a. $(\sigma_{year \geq 2009}(takes) \bowtie student)$

b. $(\sigma_{year \geq 2009}(takes \bowtie student))$

c. $\prod_{ID, name, course\_id}(student \bowtie takes)$

2.7 考虑图 2-14 所示关系数据库。给出关系代数表达式来表示下列每一个查询：

a. 找出居住在“Miami”城市的所有员工姓名。

b. 找出工资在 100 000 美元以上的所有员工姓名。

c. 找出居住在“Miami”并且工资在 100 000 美元以上的所有员工姓名。

2.8 考虑图 2-15 所示银行数据库。对于下列每个查询，给出一个关系代数表达式：

a. 找出位于“Chicago”的所有支行名字。

b. 找出在支行“Downtown”有贷款的所有贷款人姓名。

```
branch(branch_name, branch_city, assets)
customer (customer_name, customer_street, customer_city)
loan (loan_number, branch_name, amount)
borrower (customer_name, loan_number)
account (account_number, branch_name, balance)
depositor (customer_name, account_number)
```

图 2-15 习题 2.8、习题 2.9 和习题 2.13 的银行数据库

习题

2.9 考虑图 2-15 所示银行数据库。

a. 适当的主码是什么？

b. 给出你选择的主码，确定适当的外码。

2.10 考虑图 2-8 所示 *advisor* 关系，*advisor* 的主码是 *s_id*。假设一个学生可以有多位指导老师。那么，*s_id* 还是 *advisor* 关系的主码吗？如果不是，*advisor* 的主码会是什么呢？

54

2.11 解释术语关系和关系模式在意义上的区别。

2.12 考虑图 2-14 所示关系数据库。给出关系代数表达式来表示下列每一个查询：

a. 找出为“First Bank Corporation”工作的所有员工姓名。

b. 找出为“First Bank Corporation”工作的所有员工的姓名和居住城市。

c. 找出为“First Bank Corporation”工作且挣钱超过 10 000 美元的所有员工的姓名、街道地址和居住城市。

2.13 考虑图 2-15 所示银行数据库。对于下列每个查询，给出一个关系代数表达式：

a. 找出贷款额度超过 10 000 美元的所有贷款号。

b. 找出所有这样的存款人姓名，他拥有一个存款额大于 6000 美元的账户。

c. 找出所有这样的存款人姓名，他在“Uptown”支行拥有一个存款额大于 6000 美元的账户。

2.14 列出在数据库中引入空值的两个原因。

2.15 讨论过程化和非过程化语言的相对优点。

文献注解

IBM San Jose 研究实验室的 E. F. Codd 于 20 世纪 60 年代末提出了关系模型(Codd [1970])。这一工作使 Codd 在 1981 年获得了声望很高的 ACM 图灵奖(Codd[1982])。

在 Codd 最初的论文发表之后，几个研究项目开始进行，它们的目标是构造实际的关系数据库系统，其中包括 IBM San Jose 研究实验室的 System R、加州大学 Berkeley 分校的 Ingres，以及 IBM T. J. Watson 研究实验中心的 Query-by-Example。

大量关系数据库产品现在可以从市场上购得。其中包括 IBM 的 DB2 以及 Informix、Oracle、Sybase 和微软的 SQL Server。开源关系数据库系统包括 MySQL 和 PostgreSQL。微软的 Access 是一个单用户的数据库产品，它作为微软 Office 套件的一部分。

55 / 56

Atzeni 和 Antonellis[1993]、Maier[1983]以及 Abiteboul 等[1995]是专门讨论关系数据模型的文献。