第 3 章　关系数据库理论

在日常生活和科学技术领域中，我们经常会碰到各种各样的具体“关系”。人与人之间有父子、兄弟、师生等关系；两数之间有大于、等于、小于关系；电学中有电压、电阻与电流间的关系。宇宙万物之间存在着错综复杂的关系，这种关系正是各门学科所关注的问题。关系概念是对事物间多值依赖的一种描述，大家熟知的函数是关系的特例。有许多表述关系的数学模型，如在高等代数中的矩阵、离散数学中的图。集合理论为描述这种关系提供了“关系”的概念。集合理论中的“关系”本身也是一个集合，以具有某种联系的对象组合——“序组”为其成员。

换言之，在离散结构的表示中，关系不是通过揭示其内涵来描述事物间联系的，而是通过列举其外延（具有那种联系的对象组合全体）来描述这种联系。这使关系的研究可以方便地使用集合论的概念、运算及研究方法和研究成果。

3.1　关系模型概述

本节将从数据模型的三要素：数据结构、数据操作和完整性约束这三个方面详细介绍关系数据模型。

3.1.1　关系的数据结构

1. 关系的定义

域（domain）：一组具有相同数据类型的值的集合。

例如，自然数、整数、实数、长度小于 20B 的字符串集合、{0,1}等都可以是域。

笛卡儿积（Cartesian product）：给定一组域 $D_1, D_2, \cdots, D_n$，它们之中可以有相同的域。$D_1, D_2, \cdots, D_n$ 这 n 个域的笛卡儿积可以表示为

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

其中，每一个元素$(d_1, d_2, \cdots, d_n)$称为一个 n 元组（n-tuple），或简称为**元组（tuple）**。元素中的每一个值 d_i 称为一个元组**分量（component）**。

若 $D_i (i=1,2,\cdots,n)$为有限集，假设其基数（cardinal number）为 $m_i (i=1,2,\cdots,n)$，则 $D_1 \times D_2 \times \cdots \times D_n$ 的基数 M 为

$$M = \prod_{i=1}^{n} m_i$$

域的笛卡儿积可以用二维表直观地表示。表中的每一行对应一个元组，表中每一列的取值来自一个域。

【例 3-1】 给定三个域：

D_1 为学生姓名集合＝{张山，李斯，王武}；

D_2 为性别集合＝{男，女}；

D_3 为年龄集合＝{19，20}

则 D_1、D_2、D_3 的笛卡儿积是所有可能的(姓名，性别，年龄)元组集合：

$D_1 \times D_2 \times D_3$ ＝ {(张山，男，19)，(张山，男，20)，(张山，女，19)，(张山，女，20)，
(李斯，男，19)，(李斯，男，20)，(李斯，女，19)，(李斯，女，20)，
(王武，男，19)，(王武，男，20)，(王武，女，19)，(王武，女，20)}

其中，“(张山，男，19)”、“(李斯，男，20)”等都是元组；“张山”、“男”等都是元组的分量。

该笛卡儿积的基数为 3×2×2＝12，也就是说 $D_1 \times D_2 \times D_3$ 一共有 12 个元组。$D_1 \times D_2 \times D_3$ 可表示成二维表的形式，如表 3-1 所示。

表 3-1 $D_1 \times D_2 \times D_3$ 的二维表表示

学生	性别	年龄	学生	性别	年龄
张山	男	19	李斯	女	19
张山	男	20	李斯	女	20
张山	女	19	王武	男	19
张山	女	20	王武	男	20
李斯	男	19	王武	女	19
李斯	男	20	王武	女	20

由于一个学生只有一个姓名、性别和年龄，若用一个元组表示一个学生姓名、性别和年龄，则笛卡儿积中的许多元组是没有实际意义的(在这里不考虑有重名的情况)。从笛卡儿积中取出那些有一定含义的元组构成一个集合，我们称为关系，也即关系是笛卡儿积的某个有意义的子集。如表 3-2 所示，该二维表可表示域 D_1 中每个学生的基本情况。

表 3-2 学生关系的二维表表示

姓名	性别	年龄	姓名	性别	年龄
张山	男	19	王武	男	20
李斯	女	20			

至此，可以给出关系的定义。

关系(relation)：$D_1 \times D_2 \times \cdots \times D_n$ 中某个有一定语义的子集叫做在域 D_1、D_2、…、D_n 上的关系，表示为 $R(D_1, D_2, \cdots, D_n)$。其中，R 为关系的名字，n 是关系的目或度(degree)。

从表 3-2 中可以看到，关系模型的数据结构，即关系，可以表示一个学生实体的信息。数据模型的数据结构还应能描述实体以及实体之间的联系，那么关系模型如何表示实体以及实体之间的联系呢？

【例 3-2】 给出三个域：

D_1 为导师姓名集合＝{张明，李良}；

D_2 为专业名称集合＝{计算机应用技术，系统工程}；

D_3 为研究生姓名集合＝{王敏，刘勇，李新}

$D_1 \times D_2 \times D_3$ 是个三元组集合，元组个数(基数)为 $3 \times 2 \times 2$，是所有可能的(导师姓名，专业名称，学生姓名)的元组集合。

$D_1 \times D_2 \times D_3$ 中许多元组是没有意义的。因为在学校中，一名研究生只有一个导师，研究某一个专业方向(导师与研究生是一对多的联系)。

$D_1 \times D_2 \times D_3$ 的一个子集可表示导师与研究生的指导关系。这个关系可用表 3-3 所示的二维表来表示。

表 3-3 导师与研究生的指导关系

导师姓名	专业名称	学生姓名	导师姓名	专业名称	学生姓名
张明	计算机应用技术	王敏	李良	系统工程	刘勇
张明	计算机应用技术	李新			

从表 3-3 中可以看到，关系可以表示导师实体和学生实体之间的指导关系。由此可见，关系模型的数据结构非常简单，只包含单一的数据结构——关系。关系既可以表示概念模型中的实体，也可以用来描述实体间的各种联系。数据结构简单正是关系模型最大的优点。

2. 关系模型的相关概念

概念模型中实体的属性、域、实体型、实体集在关系模型中分别用关系的属性、域、关系模式、关系实例来表示。

属性(attribute)：关系所对应的域命名为属性，属性用属性名表示。在同一关系中，属性名不能相同。如表 3-3 中的属性分别是导师姓名、专业名称和研究生姓名。

若关系对应一个实体，关系的属性就是所要描述的实体的属性，即实体所对应的事物对象的特征。如表 3-2 中，“学生”关系可用姓名、性别和年龄等属性描述。“图书”关系可用图书编号、书名、作者、出版社、价格等属性来描述。

域：属性的取值范围。不同的属性可以有相同的域。如表 3-3 中，“导师姓名”和“研究生姓名”这两个属性的域都可以是由若干字符组成的字符串的集合，但属性名称不能相同。

在关系数据模型中，一般要求所有的域都是原子数据的集合。这种限制被称为第一范式条件(参见第 5 章 5.2 节)。

关系模式(relation schema)：关系的描述称为关系模式。关系模式必须指出关系的结构，即它由哪些属性构成，这些属性来自哪些域，以及属性与域之间的映像。

关系模式可以形式化地表示为

$$R(U, D, \mathrm{Dom}, F)$$

其中，R 为关系名，U 为组成关系 R 的属性集合，D 为属性组 U 中属性来自的域，Dom 为

属性向域的映像的集合，F 为属性间数据的依赖关系集合。

关系模式通常可以表示为

$$R(D_1/A_1, D_2/A_2, \cdots, D_n/A_n)$$

或

$$R(A_1, A_2, \cdots, A_n)$$

$A_1, A_2, \cdots, A_n$ 为属性名，域名及属性向域的映像常常直接说明为属性的类型和长度。

对于表 3-2 中的关系可定义关系名为“学生”，则关系模式表示为

学生(姓名，性别，年龄)

关系实例(relation instance)：一个给定关系的某一时刻的元组的集合，即当前关系的值。关系 R 的实例记为 $r(R)$。

关系模式是关系的型的描述，是静态的、稳定的。关系实例(值)是关系的“当前”元组的集合，是动态的、随时间不断变化的，其变化通过关系的元组的改变表现出来。如表 3-2、表 3-3 的内容就分别是两个关系的一个实例。

在实际使用中，人们常常把关系模式和关系实例都笼统地称为关系，这不难从上下文中加以区别。从程序设计语言角度看，“关系”的概念对应于“变量”，“关系模式”和“关系实例”相当于变量的类型定义和变量的值。

候选键(candidate key，候选码，简称为键或码)：若关系中的某一属性或属性集能唯一标识一个元组，而其任意一个真子集无此性质，则称该属性或属性集为关系的候选键。也就是说，候选键是能唯一标识一个元组的最小属性集。

候选键可以保证关系实例上任何两个元组的值在候选键的属性(集)上取值不同。需要注意的是，构成候选键的属性(集)的值对于关系的所有实例都具有唯一性，而不是只针对某一个实例。

通常在关系模式中在构成候选键的属性(集)下面画上下划线，来表明它是键的组成部分。如表 3-2 所对应的“学生”关系可写成如下形式，其中“姓名”是候选键。

学生(<u>姓名</u>，性别，年龄)

每一个关系都至少存在一个候选键。若一个关系有多个候选键，可选择其中的一个作为**主键(primary key)**。主键是数据库设计者选中用来在一个关系中区分不同元组的候选键，主键的选择会影响某些实现问题，例如索引文件的建立(参见第 6 章)。

包含候选键的属性集称为**超键(superkey)**。超键能唯一标识元组，但不具有最小化性质。

若关系只有一个候选键，且这个候选键包含了关系的所有属性，称该候选键为**全键(all-key)**。

主属性(prime attribute)：构成候选键的每个属性称为主属性。不包含在任何候选键中的属性称为**非主属性(non-prime attribute)**或**非码属性(non-key attribute)**。

外键(foreign key，外码)：若关系 R 的一个属性(集) F 与关系 S 的主键 K_s 对应，即关系 R 中的某个元组的 F 上的值也是关系 S 中某个元组的 K_s 上的值，则称该属性(集) F 为关系 R 的外键。

上述的关系 R 为参照关系(referencing relation,引用关系),关系 S 为被参照关系(referenced relation)或目标关系(target relation)。关系 R 和关系 S 可以是同一个关系。目标关系的主键 K_s 和参照关系 R 的外键 F 的命名可以不同,但必须定义在同一(或同一组)域上。

【例 3-3】 学生实体和课程实体分别用关系“学生”和“课程”来表示,它们之间的联系用关系“选课”来表示。

学生(学号,姓名,性别,出生时间,所在系)

课程(课程号,课程名,先修课程号)

选课(学号,课程号,成绩)

“学生”关系的候选键(主键)为“学号”;“课程”关系的候选键(主键)为“课程号”,“课程”关系的“先修课程号”引用了“课程”关系的“课程号”属性,是“课程”关系的外键,被引用关系和引用关系是同一个关系。“选课”关系中的“学号”和“课程号”共同构成关系的候选键(主键),又分别是“选课”关系的外键,它们分别对应“学生”关系和“课程”关系的主键。

3. 关系的性质

在集合论中,关系可以是无限集合;而且,关系中每个元组是“序组”,即元组中的分量有前后顺序,元组$(d_1,d_2,\cdots,d_n)$和$(d_2,d_1,\cdots,d_n)$不同。当关系作为关系数据模型的数据结构时,需要给予如下的限定和扩充:

(1) 限定关系数据模型中的关系必须是有限集合。无限关系在数据库系统中是没有意义的。

(2) 通过为关系的每个属性附加一个属性名来取消元组分量的有序性,即关系 $R(A_1,\cdots,A_i,A_j,\cdots,A_n)$ 与关系 $R(A_1,\cdots,A_j,A_i,\cdots,A_n)$ 为同一关系模式,其对应的元组 $(d_1,d_2,\cdots,d_i,d_j,\cdots,d_n)$ 和 $(d_1,d_2,\cdots,d_j,d_i,\cdots,d_n)$ 相同。

归纳起来,关系具有如下一些性质:

(1) 元组个数有限性。

(2) 属性名唯一性,即关系中不能有重名属性。

(3) 属性的次序无关性,即属性列的次序可以任意交换。

(4) 元组的唯一性,即关系中不能出现完全相同的两个元组。

(5) 元组的次序无关性,即元组的顺序可以任意交换。关系是元组的集合,而不是元组的列表。

(6) 元组分量的原子性,即每个分量都必须是不可分割的数据项。

(7) 分量值域同一性。每一属性列中的元组分量是同一数据类型,来自同一个域。

4. 关系与二维表

从用户的角度来看,关系模型的数据结构就是一张二维表,表中的每行对应一个元组,表中的每列对应一个取值域。

在数据库关系模型中,经常将关系与一张二维表等同起来。二维表的表头由各属性

名构成，每一列表示一个属性，每一行表示一个元组，所有行的集合构成了关系 R 的实例。

关系是一种抽象的对象，表则是一种具体的图形，关系这种抽象对象能在平面上以表的形式简单地表示出来，使得关系模型容易理解和使用，这是关系模型的一个巨大优势。

但表和关系实际上是不同的，注意加以区分，有助于对关系的理解。表和关系的不同之处具体体现在以下几方面：

(1) 表中各列从左到右是有序的，关系中属性的次序是任意的。

(2) 表中各行从上到下是有序的，关系中元组的次序是任意的。

(3) 表中可能包含重复的行，关系中不能有相同的元组。

(4) 表中至少含有一个列，但可存在不含任何属性的关系，相当于空集合。

(5) 表中允许包含空行(例在 SQL 中)，而关系中不允许。

(6) 表是“平面的”或是“二维的”，而关系却是“n 维的”，是 n 个域上的一个 n 元组的集合。

一般情况下，理论研究侧重关系的概念，而实际的 DBMS、数据库语言等更多地支持表的概念，而不是关系。

例如，在数据库语言和宿主语言中支持表的概念，会有“取出第 N 个元组的操作”，查询结果的呈现涉及元组的有序排列(游标、ORDER BY)等问题。在实际的数据库系统中，定义关系模式时，属性是没有顺序的，但定义后，在系统中就有了顺序。但这些问题不属于关系模型本身的问题，而是关系系统的实现问题。

5. 关系数据库(Relation DataBase，RDB)

在关系模型中，数据库是由一个或多个关系组成的。数据库的关系模式集合叫做关系数据库模式(relational database schema)，或者简称为数据库模式(database schema)，是对关系数据库的型的描述，包括若干域的定义以及在这些域上定义的若干关系模式。关系数据库的实例(值)是这些关系模式在某一时刻对应关系实例的集合。

在某一应用领域中，描述所有实体集及实体之间联系所形成的关系的集合就构成了一个关系数据库。

3.1.2 关系的完整性约束

关系模型的完整性约束是关系模型对于存储在数据库中的数据具有的约束能力，也就是关系的值随着时间变化应该满足的一些约束条件。这些约束条件实际上是现实世界对关系数据的语义要求，关系数据库中的任何关系在任何时刻都需要满足这些语义。完整性约束保证授权用户对数据库的操作不会破坏数据的一致性。

关系模型中有三类完整性约束：实体完整性、参照完整性和用户定义的完整性。实体完整性和参照完整性是关系模型必须满足的完整性约束条件，被称为关系的两个不变性，一般由关系型数据库管理系统(RDBMS)自动支持。用户定义的完整性是应用

领域需要遵循的约束条件,体现了具体应用领域中的语义约束,RDBMS应提供定义和检验完整性约束的机制,以便用统一的系统的方法处理它们,而不应由应用程序承担这一功能。

1. 实体完整性(entity integrity)

在关系模型中,实体用关系来描述,关系是元组的集合。为使候选键能唯一标识一个元组,需对构成候选键的每个主属性进行如下约束。

实体完整性约束规则:若属性 A 是关系 R 的主属性,则属性 A 的值不能为空值。

属性值为空的含义是该属性值"不知道"、"不清楚"、"不存在"或"无意义"等。在关系模型中使用空缺符 NULL 来表示。

例如,在例3-3中,"学生"关系的主属性"学号"、"课程"关系的主属性"课程号"不能为空;选课(学号,课程号,成绩)关系中主属性"学号"、"课程号"都不能为空。

这条约束规则的实质是体现了关系模型中的键约束特性,主属性为空,说明存在某个不可标识的元组,即存在不可区分的实体值。

对于实体完整性约束规则的使用做如下几点说明:

(1) 实体完整性是针对系统中定义的基本关系(存储的关系表)而言的,并不对查询的结果关系(临时表)、视图(参见第4章4.5节)等进行约束。

(2) 如果关系的候选键由若干属性组成,则所有构成候选键的属性即主属性都不能为空。

2. 参照完整性(referential integrity)

现实世界中实体之间往往存在某种联系,在关系模型中实体以及实体间的联系都是用关系来描述的。这样就自然存在着关系与关系之间的参照,关系之间的参照一般通过外键来描述,并遵循如下约束规则。

参照完整性约束规则:若属性(或属性集)F 是关系 R 的外键,它与关系 S 的主键 K_s 对应,则对于 R 中元组在 F 上的取值只能有两种可能:或者取空值,或者等于 S 中某个元组的 K_s 值。

【例3-4】 若学生实体和专业实体可以用下面的关系来表示:

学生(学号,姓名,性别,专业号,出生时间)

专业(专业号,专业名)

若属性"专业号"是"学生"关系的外键,又是"专业"关系的主键,则"学生"关系中每个元组的"专业号"属性值只能是下面两种情况:

(1) 空值,表示尚未给学生分配专业。

(2) 非空值,这时元组在"专业号"属性上的元组分量值必须是"专业"关系中某个元组的"专业号"值,表示该学生只能就读某个存在的专业。

这条约束规则的实质是不允许引用不存在的实体,在某个关系中出现的值也必须在另外一个相关的关系中出现。

对于参照完整性规则的使用做如下几点说明:

(1) 关系 R 和 S 可以是同一关系，表明同一关系中不同元组之间的参照关系。

例如，在例 3-3 中，在“课程(课程号，课程名，先修课程号)”关系中，如果规定每门课程的直接先修课程只能是所开设课程中的一门，那么“先修课程号”就是“课程”关系中的外键，其对应的主键为本关系的主键“课程号”。

(2) 外键并不一定要与相应的主键同名，如例 3-3 中的“课程号”和“先修课程号”，但必须定义在相同的值域上。在实际应用中，为便于识别，当外键与相应的主键属于不同关系时，往往给它们取相同的名字。

(3) 若外键 F 为一属性集且为空值，则 F 中的每个属性值均为空值。F 是否能为空值，应视具体问题而定。

例如，在例 3-3 中，“选课”关系中的“学号”和“课程号”分别是该关系的外键，按照参照完整性约束规则，属性值可以为空值或被参照关系“学生”和“课程”关系中某个元组的主键值。但由于“学号”和“课程号”又分别是“选课”关系的主属性，按照实体完整性约束规则，它们均不能取空值，只能取相应被参照关系中已经存在的某个元组的主键值。

3. 用户定义的完整性(user-defined integrity)

任何 RDBMS 都应该支持实体完整性和参照完整性，这是关系模型所要求的。除此之外，不同的关系数据库根据其应用环境的不同，往往还需要一些特殊的约束条件，反映某一具体应用所涉及的数据必须满足的语义要求。

比如，每个属性都有一个类型约束，使得该属性的每个取值都只能是该类型，存在着“只能取整数”、“字符串长度最大为 30”等域约束条件；可能对属性值的取值范围进行约束，如“学生考试成绩在 0～100 之间”、“在职职工的年龄不能大于 60 岁”等；可能对同一关系中的元组进行约束，要求“不允许出现两个不同的学生拥有相同的姓名”；可能对同一元组的各属性间进行约束，要求“职工工资与职工的工龄和职务满足一定的算术关系”；可能对数据库的各关系间进行约束，要求“不允许状态值小于 20 的供应商供应任何数量多于 500 的零件”，等等。

一般来说，一个自定义的完整性约束可以是关于数据库的任意谓词。但因检测任意谓词的代价太高，大多数 DBMS 允许用户指定只需极小开销就可以检测的完整性约束条件。

4. 完整性控制机制

关系模型的完整性约束是为了防止关系数据库中存在不符合语义的数据，也就是防止数据库中存在不正确的数据。为了保证数据库的完整性，DBMS 必须提供定义、检查和控制数据完整性的机制(称为完整性子系统)。

(1) 定义功能，即提供定义完整性约束条件的机制。

(2) 检查功能，即检查用户发出的操作请求，看其是否违背了完整性约束条件。

(3) 保护功能，即监视数据操作的整个过程，如果发现有违背了完整性约束条件的情况，则采取一定的动作来保护数据的完整性。

完整性约束的定义通常被看成是数据库模式设计的一部分存入数据库中。实体完整

性和参照完整性是关系模型必须满足的完整性约束条件,一般在关系模式定义时进行定义,用户定义的一些关于属性、域的约束也可在关系模式定义时进行定义,数据库管理系统(RDBMS)自动完成检查和保护(参见4.2节和4.4节)。对数据库中数据状态变化所施加的约束,可以通过定义触发器(参见4.5节)和定义事务(参见第8章)来实现。而同一关系属性间的约束将以函数依赖的形式在第5章中讨论。

在早期的RDBMS中,没有提供定义和检验这些完整性的机制,因此需要应用开发人员在应用系统的程序中进行完整性检查。

例如,对于例3-3中的"选课"关系,每插入一条记录必须在应用程序中写一段程序来检查其中的"学号"、"课程号"是否与"学生"、"课程"关系的相应属性一致。现在只需定义和实现参照完整性就可以了。

3.1.3 关系操作

关系模型给出了关系操作的能力说明,早期的关系操作能力通常用代数方式或逻辑方式来表示,分别称为关系代数(relational algebra)和关系演算(relational calculus)。

关系代数是用对关系的代数运算来表达查询要求的。用户可以指定基本的检索请求,检索的结果是一个新的关系,这个新关系可能由一个或多个关系所构成。可以使用同样的代数操作进一步操纵这些新关系。关系代数操作的一个序列构成一个关系代数表达式,其结果还是一个关系,它表示一个数据库查询(或检索请求)的结果。

关系代数的重要性体现在以下几个方面:

(1) 它为关系模型的数据操作提供了一个形式化的基础,因此经常被用作衡量另一种关系模型语言表达能力的尺度。当一种语言至少拥有代数的作用,即它的表达式允许通过代数的形式来定义每一个关系时,我们就说该语言是关系完备的。

(2) 关系代数被用在关系数据库管理系统中,作为实现和优化查询的基础(参见第7章),用来说明从数据库中提取数据的基本技术。

(3) 面向RDBMS的SQL标准查询语言中结合了关系代数中的一些概念。

与关系代数不同,关系演算为关系查询提供了一个更高级的描述性表示法。关系演算是用查询得到的元组应满足的谓词条件来表达查询要求。关系演算表达式创建了一个新关系,这个新关系以变量形式指定。而变量的取值范围为数据库关系中的元组(元组演算)或属性(域演算)。在演算表达式中,对指定如何检索查询结果的操作没有次序上的要求,演算表达式只指定了结果中应当包含什么信息。关系演算的重要性体现在其有坚实的数理逻辑基础,同时面向RDBMS的SQL标准查询语言也以元组关系演算作为其部分基础。

本章所讨论的关系代数、元组关系演算和域关系演算均是抽象的查询语言(query language)。这些语言是用户用来从数据库中请求获取信息的语言,通常比标准的程序设计语言层次更高,可以分为过程化语言和非过程化语言。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。这些语言与具体的RDBMS中实现的实际语言并不完全相同，不对RDBMS语言给出具体的语法要求，即不同的RDBMS可以定义和开发不同的语言来实现这些操作。但关系代数和关系演算能用作评估实际系统中查询语言能力的标准和基础。实际的查询语言除了提供关系代数语言和关系演算语言所表达的功能外，还提供许多附加的功能。

曾经出现的一些RDBMS实际查询语言有：

(1) ISBL(Information System Base Language)是IBM公司英格兰底特律科学中心在1976年研制的，用在一个实验系统PRTV(Peterlee Relational Test Vehicle)上。ISBL语言的每个查询语句都近似一个关系代数表达式。

(2) QUEL(Query Language)是美国加州大学伯克利分校研制的关系数据库系统INGRES使用的查询语言。QUEL语言参照E. F. Codd提出的APLHA元组演算语言研制的，是一种基于元组关系演算并具有完善的数据定义、检索、更新等功能的数据库语言。

(3) QBE(Query By Example)是IBM高级研究实验室的M. M. Zloof提出的，为图形终端用户设计的一种域演算语言，1978年在IBM 370上实现。QBE属于人机交互语言，使用方便。其思想已渗入到许多DBMS中。

关系模型与其他数据模型相比，最具特色的是关系数据操作语言。关系操作语言灵活方便，表达能力和功能都非常强大。

目前使用的是一种结构化的SQL查询语言，不仅具有丰富的查询功能，而且具有数据定义和控制功能。它具有语言简洁，易学易用的特点，是关系数据库的标准语言和主流语言。

我们将在第3.2节讨论关系代数的操作，在第3.3节讨论关系演算的内容，在第4章讨论SQL语言的功能。

3.2 关系代数

关系代数是一种过程化的查询语言，它用对关系的运算来表达查询要求。一门代数总是由一些操作运算符和一些原子操作数组成的。比如，算术代数中的原子操作数是像常量5和变量x这样的操作数，而加(+)、减(−)、乘(×)、除(÷)是其中的操作运算符。任何一门代数都允许把运算符用在原子操作数或者是其他代数表达式上构造表达式，括号一般被用来组合操作数和运算符。关系代数也是一门代数，它基于一组为数不多的以关系为操作对象的运算符。每个运算符对一个或两个关系进行运算，产生的结果是另外一个关系，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)，如同将算术运算组合成算术表达式一样。数据查询就是一个关系代数运算表达式。表达式的结果是关系，也就是这个查询的答案。

关系代数的原子操作数包括代表关系的变量和代表关系实例的常量。

关系代数的运算符可分为两类：传统的集合运算和专门的关系运算。传统的集合运算将关系看成元组的集合，其运算是从关系的“水平”方向即元组的角度来进行的。而专

门的关系运算不仅涉及元组,而且涉及属性列。在关系代数表达式中还使用比较运算符和逻辑运算符来辅助专门的关系运算。

传统的集合运算主要包括并、差、交、广义笛卡儿积。

专门的关系运算主要包括投影、选择、连接、除。

3.2.1 传统的集合运算

传统的集合运算是二目运算,主要包括并、差、交、广义笛卡儿积4种运算。

设两个关系R和S具有相同的类型,或R和S是相容关系,即关系R和S具有相同的目,且相应的属性取自同一个域,则定义并、差、交和广义笛卡儿积运算如下。

1. 并(union)运算

关系R与S的并是一个与R、S相容的关系,且其元组由属于R或S的元组组成,表示为$R \cup S$。

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

并运算可用于实现两个关系的合并,建立多关系间的查询和定位,实现元组的插入操作。

2. 差(difference)运算

关系R与S的差是一个与R、S相容的关系,且其元组由属于R但不属于S的元组组成,表示为$R-S$。

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

注意:$S-R$与$R-S$是不同的,$S-R$表示由只在S中出现而不在R中出现的元组构成的关系。

差运算可用来实现元组的删除操作。

3. 交(intersection)运算

R和S的交是一个与R、S相容的关系,其元组由既属于R又属于S的所有元组组成,表示为$R \cap S$。

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

关系的交运算可以用差运算来实现:

$$R \cap S = R-(R-S)$$

或

$$R \cap S = S-(S-R)$$

4. 广义笛卡儿积(Cartesian product)

关系的笛卡儿积运算可以将任意两个关系的信息组合在一起。关系R和S的广义笛卡儿积(或者称为笛卡儿积、叉积,或者就叫做积)是一个有序对的集合,有序对的第一个元

素是关系 R 中的任何一个元组，第二个元素是关系 S 中的任何一个元组，表示为 $R\times S$。

$$R\times S=\{\widehat{t_r t_s}\mid t_r\in R\wedge t_s\in S\}$$

设关系 R 和关系 S 分别是 m 目和 n 目关系，R 中有 k_1 个元组，S 中有 k_2 个元组，则 $R\times S$ 为一个 $m+n$ 目的新关系，共有 $k_1\times k_2$ 个元组，且每个元组的前 m 个分量是关系 R 的一个元组，后 n 个分量是关系 S 的一个元组。结果关系模式是 R 和 S 关系模式的并，但是如果 R 和 S 恰好有同名的属性，就需要把至少一个关系中相应的属性名更改为不同的名称。为了使含义清楚，如果属性 A 在关系 R 和 S 中均出现，则结果关系模式中分别用 $R.A$ 和 $S.A$ 表示来自 R 和 S 的属性。当某个关系如果需要与自身作笛卡儿积运算时怎么办呢？在下一节，将提供一种改名运算来解决这个问题。

广义笛卡儿积是连接操作的基础。

【例 3-5】 给定关系 R、S，$R\cup S$、$R\cap S$、$R-S$、$R\times S$ 的结果如图 3-1 所示。

关系R

A	B	C
a	b	c
d	a	f
c	b	d

关系S

A	B	C
d	a	f
b	g	a

R∩S

A	B	C
d	a	f

R×S

R.A	R.B	R.C	S.A	S.B	S.C
a	b	c	d	a	f
a	b	c	b	g	a
d	a	f	d	a	f
d	a	f	b	g	a
c	b	d	d	a	f
c	b	d	b	g	a

R∪S

A	B	C
a	b	c
d	a	f
c	b	d
b	g	a

R−S

A	B	C
a	b	c
c	b	d

图 3-1 集合运算示例

【思考】 给定关系实例 S_1、S_2（如表 3-4、表 3-5），若要实现如下数据操作，需对关系做什么运算？

（1）查询两个关系中所有学生的信息。

（2）查询两个关系中相同学生的信息。

（3）删除关系 S_1 中所包含的关系 S_2 中的学生信息。

表 3-4 关系实例 S_1

学号	姓名	性别	出生日期	所在系
98101	张山	男	1990-02-14	计算机
98105	李斯	女	1991-10-15	计算机
98204	王武	男	1990-08-19	数学

表 3-5 关系实例 S_2

学号	姓名	性别	出生日期	所在系
98204	王武	男	1990-08-19	数学
98205	赵路	女	1989-09-30	数学

3.2.2 专门的关系运算

专门的关系运算包括投影、选择、连接、除法运算。

1. 投影(projection)运算

关系 R 上的投影运算是从 R 中选择若干属性列组成一个新的关系。投影运算是一个一元的、对属性进行操作的运算。投影运算符用大写希腊字母 π(pi)表示,在结果中出现的属性名作为 π 的下标,参与运算的关系作为 π 后括号内的参数。在投影运算中,可以用属性的位置标记隐含地作为关系的属性名,也适用于关系代数表达式运算的结果中。

设关系 R 为 n 目关系,$A_{i1},A_{i2},\cdots,A_{im}$ 是关系 R 的属性 $A_1,A_2,\cdots,A_n$ 的一部分,则关系 R 在 $A_{i1},A_{i2},\cdots,A_{im}$ 上的投影是一个 m 目关系,其属性为 $A_{i1},A_{i2},\cdots,A_{im}$,表示为

$$\pi_{A_{i1},A_{i2},\cdots,A_{im}}(R)$$

或

$$\pi_{i1,i2,\cdots,im}(R)$$

投影操作提取了原关系的某些属性,而且与原关系相比,元组数可能会减少,因可能有重复元组被去除。位置标记不像属性名易于理解,本教材后续基本不采用位置标记。通过投影运算,可以对关系内的任意属性的数据进行查询。

2. 选择(selection)运算

选择运算是一个对元组操作的一元运算。选择运算在关系 R 中选择满足给定条件的元组。用小写希腊字母 σ(sigma)来表示选择运算符,将谓词条件写作 σ 的下标,参与运算的关系作为 σ 后括号内的参数。

设 F 为一逻辑表达式,则在关系 R 上的 F 选择是在 R 中挑选满足条件 F 的所有元组组成一个新的关系。这个新关系与 R 具有相同的模式,是 R 的一个子集,表示为

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

表达的含义是:假设 t 是 R 中任意一个元组,把 t 代入到条件 F 中,如果代入的结果为真,那么这个元组就是 $\sigma_F(R)$ 中的一个元组,否则此元组不在结果中出现。

逻辑表达式 F 由下面的规则组成:

(1) 由基本逻辑表达式 $a\theta b$ 组成,a、b 可为属性名或常量,但不能同时为常量。θ 为比较符$<$、$>$、$=$、$\leqslant$、$\geqslant$和$\neq$。

注意:参与比较的每个属性必须是选择运算符的关系操作数里的一个属性,否则就是语法上的错误。

(2) 由基本逻辑表达式经逻辑运算 $\neg$(非)、$\wedge$(与)和 $\vee$(或)组成,称为复合逻辑表达式。

通过选择运算,可以对关系内的任意元组的数据进行查询。

【例 3-6】 给出关系 R，$\pi_{C,A}(R)$、$\sigma_{A>'c' \vee C<'d'}(R)$ 的结果如图 3-2 所示。

关系R

A	B	C
a	b	c
d	a	f
c	b	d

$\pi_{C,A}(R)$

C	A
c	a
f	d
d	c

$\sigma_{A>'c' \vee C<'d'}(R)$

A	B	C
a	b	c
d	a	f

图 3-2 投影、选择运算示例

3. 连接(join)运算

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运算的两个关系在某些属性上可以进行比较。θ-连接运算是在 R 和 S 的广义笛卡儿积 $R\times S$ 中选取符合 $A\theta B$ 条件的元组，即选择在关系 R 中 A 属性组上的值与在关系 S 中 B 属性组上的值满足比较操作 θ(theta)的元组。表示为

$$R \bowtie_{A\theta B} S = \{\widehat{t_r t_s} \mid t_r \in R \wedge t_s \in S \wedge t_r[A]\theta t_s[B]\}$$

其中，A 和 B 分别是 R 和 S 上属性个数相等且可比的属性组，θ 是比较运算符。$t_r[A]$、$t_s[B]$分别表示关系 R、S 的元组 t 在属性列 A、B 上诸分量的集合。

就像在笛卡儿积操作中一样，θ-连接的结果关系的模式是模式 R 和模式 S 的并。如果有必要的话，需要在重名的属性前面加上“$R.$”或“$S.$”。

连接运算中有两种最为重要、最为常用的连接：等值连接和自然连接。

(1) 当 θ 为“=”时，θ-连接运算称为等值连接。表示为

$$R \bowtie_{A=B} S = \{\widehat{t_r t_s} \mid t_r \in R \wedge t_s \in S \wedge t_r[A] = t_s[B]\}$$

$t_r[A]=t_s[B]$表示 $R.A_1=S.B_1 \wedge R.A_2=S.B_2 \wedge \cdots \wedge R.A_k=S.B_k$

(2) 自然连接是一种特殊的等值连接，它要求两个关系中进行比较的分量必须是相同的属性组，即 A 和 B 相同，并且在结果中把重复的属性列去掉。

$$R \bowtie_{A=A} S = \{\widehat{t_r t_s} \mid t_r \in R \wedge t_s \in S \wedge t_r[A] = t_s[A]\}$$

$t_r[A]=t_s[A]$表示 $R.A_1=S.A_1 \wedge R.A_2=S.A_2 \wedge \cdots \wedge R.A_k=S.A_k$

R 和 S 的自然连接表示为

$$R \bowtie S = \pi_{Z_1, Z_2, \cdots, Z_m}(\sigma_{R.A_1=S.A_1 \wedge R.A_2=S.A_2 \wedge \cdots \wedge R.A_k=S.A_k}(R\times S))$$

其中，$Z_1, Z_2, \cdots, Z_m$ 是从 $R\times S$ 中去掉重复属性 $S.A_1, S.A_2, \cdots, S.A_k$ 后的诸属性。

一般的连接操作是从元组的角度进行运算，但自然连接还需要取消重复属性，所以它是同时从元组和属性列两个角度进行运算的。

在连接运算中，现在一般用逻辑表达式 F 来代替 $A\theta B$，称为 F 连接，表示为 $R \bowtie_F S$。F 连接运算就是在 R 和 S 的广义笛卡儿积 $R\times S$ 中，选取符合 F 条件的元组，则 $R \bowtie_F S$ 与 $\sigma_F(R\times S)$ 等价，表示为

$$R \bowtie_F S \equiv \sigma_F(R\times S)$$

则自然连接 $R \bowtie S$ 与 $\pi_L(\sigma_F(R\times S))$ 等价，其中条件 F 代替 $A=B$，L 是所有 R 中的属性和在 S 中但不在 R 中的属性的列表。表示为

$$R \bowtie S \equiv \pi_L(\sigma_F(R\times S))$$

【例 3-7】 给出关系 R、S、$R \bowtie_{D>E} S$、$R \bowtie_{D=E} S$ 以及 $R \bowtie S$，如图 3-3 所示。

在两个关系 R 和 S 做 $R \bowtie S$ 时，选择两个关系在相同属性上值相等的元组构成新的关系。R 中的某些元组可能因在 S 中不存在相同属性上值相等的元组，从而使这些元

关系R

A	B	D
1	2	4
2	4	6
1	1	7

关系S

D	E
7	5
6	7
8	4

$R\bowtie_{D>E}S$

$R.A$	$R.B$	$R.D$	$S.D$	$S.E$
2	4	6	7	5
2	4	6	8	4
1	1	7	7	5
1	1	7	8	4

$R\bowtie_{D=E}S$

$R.A$	$R.B$	$R.D$	$S.D$	$S.E$
1	2	4	8	4
1	1	7	6	7

$R\bowtie S$

A	B	D	E
2	4	6	7
1	1	7	5

图 3-3　连接运算示例

组的信息不能保留在连接结果中,同样,S 中的某些元组也可能被舍弃。在一个自然连接中,如果一个元组不能和另外一个关系中的任何一个元组在相同属性上值相等的话,这个元组就被称为悬浮元组(dangling tuple)。例如,对于例 3-7 中的 $R\bowtie S$,R 的第 1 个元组(1,2,4)、S 的第3 个元组(8,4)就是悬浮元组。

如果在自然连接结果中把舍弃的这些悬浮元组保留下来,并且在这些元组新增加的属性上赋空值 NULL,这种连接称做外连接(outer join)。如果在结果中只保留运算符左边(第 1 个)关系中的悬浮元组,称做左外连接;如果在结果中保留运算符右边(第 2 个)关系中的悬浮元组,称做右外连接;把两个关系中的悬浮元组都保留下来,称做完全外连接。

【例 3-8】 对例 3-7 中的关系 R、S 进行完全外连接、左外连接、右外连接。连接结果如图 3-4 所示。

左外连接

A	B	D	E
1	2	4	NULL
2	4	6	7
1	1	7	5

完全外连接

A	B	D	E
1	2	4	NULL
2	4	6	7
1	1	7	5
NULL	NULL	8	4

右外连接

A	B	D	E
2	4	6	7
1	1	7	5
NULL	NULL	8	4

图 3-4　外连接运算示例

4. 除(division)运算

设有关系 $R(X,Y)$和 $S(Y)$,其中 X、Y 为属性组,$S(Y)\neq\Phi$,则 R 除以 S 也是一个关系,称为 R 除以 S 的商,可记为 $R\div S$。

R 能被 S 除必须满足下面的前提条件:

(1) R 中的属性包含 S 中的所有属性。

(2) R 中有一些属性不出现在 S 中。

为了理解除法运算，下面先介绍像集的概念：

给定一个关系 $R(X,Y)$，X 和 Y 为属性组。当 $t[X]=x$ 时，x 在 R 中的像集(images set)为：$Y_X=\{t[Y]|t\in R,\ t[X]=x\}$。像集实际表示了 R 中属性组 X 上值为 x 的诸元组在 Y 上分量的集合。

【例 3-9】 对于关系 R(姓名，课程)中的元组在“姓名”(X 属性)上的一个值“张军”，其在“课程”(Y 属性)上的像集为{物理，数学}，可表示张军修读的所有课程，如图 3-5 所示。

关系R

属性组X	属性组Y
姓名	课程
张军	物理
王红	数学
张军	数学

若x=张军
则Y_X

课程
物理
数学

Y_X表示了张军修读的所有课程

图 3-5 像集概念示例

$R\div S$ 得到一个新的关系，其属性由 R 中那些不出现在 S 中的属性所组成，其元组由满足如下条件的 x 构成：

(1) x 是关系 R 中元组在属性组 X 上的分量值。

(2) 像集 Y_X 包含关系 S 的所有元组。

根据除运算的定义，要想知道关系除法运算的结果，可分如下 4 步进行：

(1) 获取关系 R 中元组在 X 上的分量值 x。

(2) 获取各 x 的像集。

(3) 检查各个分量的像集是否包含 S。

(4) 将满足条件的 x 放入结果集合。

【例 3-10】 给定关系 R 和 S，如图 3-6 所示。先找到关系 R 中元组在属性组 X 上可能的分量值 x(有 3 个)，再计算其像集 Y_{x1}、Y_{x2}、Y_{x3}，判断每个像集是否包含关系 S 中的所有元组(Y_{x2}、Y_{x3}满足条件)，将满足条件的 x 放入结果关系 T 中(即 $x2$、$x3$)。T 就是 $R\div S$ 的结果。

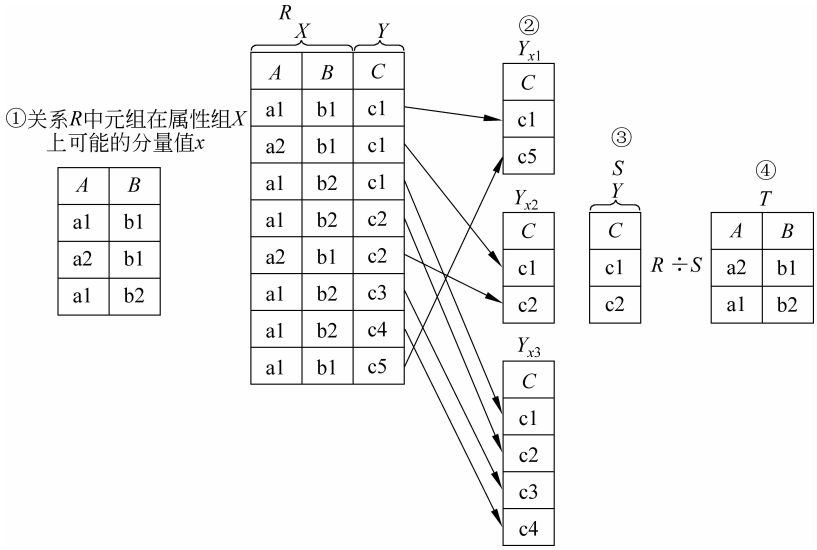

图 3-6 除法运算示例

【例 3-11】 给定选课关系 SC 和课程关系 C，如图 3-7 所示。查询选修所有课程的学生的学号。

关系SC

学号	课程号	成绩
1	C1	83
1	C5	83
5	C5	90
5	C1	92

关系C

课程号	课程名
C1	C语言
C5	数据库

图 3-7 例 3-11 关系实例

对如下三种表达，哪一种是正确的？

(1) SC÷C。

(2) SC÷$\pi_{课程号}(C)$。

(3) $\pi_{学号,课程号}(SC)\div\pi_{课程号}(C)$。

说明：

(1) 由于关系 C 中具有不包含在关系 SC 中的属性“课程名”，因而，除运算的前提条件不满足，无法进行除运算。所以，解法(1)是错误的解法。

在进行除运算前，应对除关系 C 进行投影，去掉不包含在被除关系 SC 中的属性“课程名”，即计算 $\pi_{课程号}(C)$，再做除法运算。由于投影之后所得的关系只包含属性“课程号”，而“课程号”包含在被除关系 SC 中，所以满足除运算的条件，能够进行除运算。

(2) 被除关系 SC 中不包含在除关系 C 中的属性是“学号”和“成绩”，所以除法运算的结果中包含“学号”和“成绩”两个属性。除运算的结果是那些选修了课程表中全部课程(C1 和 C5)且成绩相同的学生的学号和成绩，运算结果为{(1,83)}。显然，解法(2)不满足查询需求。

(3) 要完成题目所要求的查询“选修所有课程的学生的学号”，在进行除运算前，应根据操作的要求准确地确定像集属性和结果属性。对除关系和被除关系进行投影，去掉不需要的属性，再做除法运算，即执行 $\pi_{学号,课程号}(SC)\div\pi_{课程号}(C)$运算，运算结果为{1,5}。所以，解法(3)是正确的解法。

除运算不是基本运算，它可以由基本运算进行如下计算，推导出来：

$$R\div S=\pi_x(R)-\pi_x((\pi_x(R)\times S)-R)$$

前面介绍的 8 种关系代数运算，其中并、差、广义笛卡儿积、投影和选择 5 种运算为基本的运算；而其他 3 种运算，即交、连接和除，均可以用前 5 种基本运算来表达。

5. 重命名(rename)运算

为了有效地管理由关系代数生成的结果关系的属性名字，通常会引进一个重命名操作，运算符用小写希腊字母 ρ(rho)表示，则重命名运算表示为

$$\rho_{S(A_1,A_2,\cdots,A_n)}(R)$$

重命名后的关系与关系 R 有完全相同的元组，只不过关系的名字变成了 S。另外，关系 S 的各个属性分别命名为 $A_1,A_2,\cdots,A_n$，按从左到右的顺序排列。如果只是想把关系的名字改变为 S，并不改变其中的属性名字，可简单地使用 $\rho_S(R)$即可。

重命名运算可解决含有相同属性的两个关系的笛卡儿乘积或连接操作的属性命名问题，也可用来实现关系的自身连接操作(参见例 3-12 中的查询(8))，还可用来给一个代数表达式的结果命名为一个新关系。

许多文献还介绍一些扩展的关系代数运算，比如广义投影、聚集运算等，本教材不对此作介绍，这些操作在后续 SQL 语言的学习中会很容易理解和实现。

3.2.3 用关系代数表达式实现关系查询

如果只能在单个或者两个关系上进行一个操作，那么关系代数就不会那么有用。然而，如同其他的所有代数一样，关系代数允许任意复杂的表达式，其操作符可以用于任何关系之上，这个关系既可以是某个给定的关系，也可以是操作得到的结果关系。于是，可以用关系代数表达式来表达对数据库中关系的数据查询，表达式的结果就是这个查询的答案。

1. 关系代数表达式

关系代数中基本的表达式是如下二者之一：

(1) 数据库中的一个关系。

(2) 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{(张山，男，19)，(王武，男，20)，(李斯，女，20)}。

设 E_1 和 E_2 是关系代数基本表达式，则进行以下基本运算的结果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。

以上关系代数表达式进行有限次代数运算构成新的关系代数表达式。

由关系代数的基本运算足以表达任何关系代数查询。但若局限于基本运算，某些常用查询表达出来会很冗长。所以，关系代数表达式中也使用以下组合运算，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。

- $E_1 \cap E_2$；
- $E_1 \bowtie E_2$，$E_1 \bowtie_{A\theta B} E_2$，$E_1 \bowtie_F E_2$；
- $E_1 \div E_2$。

2. 实现关系查询

用关系代数表达式表达关系查询一般遵循如下求解过程：

(1) 确定查询目标，即结果关系中的属性。

(2) 明确查询条件。

(3) 选择从条件到目标的查找路径，并据此确定操作对象。即明确在操作过程中需要使用到哪些关系，这些关系又是如何被连接成一个关系的。

(4) 关系的合并。即根据步骤(3)的分析结果进行关系的连接。

(5) 元组的选择。即根据步骤(2)的分析结果(查询条件)进行元组的选择。

(6) 属性的指定。即根据步骤(1)的分析结果执行投影操作。

【例 3-12】 有一个描述学生及其选修的课程的关系数据库,它由三个关系组成,其关系模式为

学生关系:S(学号、姓名、性别、出生时间、专业)

课程关系:C(课程号、课程名、先修课程号)

选课关系:SC(学号、课程号、成绩)

用关系代数表达式表示如下查询:

(1) 查询 1990 年以后出生的学生姓名。

$$\pi_{姓名}(\sigma_{出生时间>'1990\text{-}12\text{-}31'}(S))$$

(2) 查询选修了课程号为 C2 的学生学号。

$$\pi_{学号}(\sigma_{课程号='C2'}(SC))$$

(3) 查询选修了课程名为“操作系统”,成绩为 90 的所有学生姓名。

$$\pi_{姓名}(\sigma_{课程名='操作系统'\wedge 成绩=90}(S \bowtie SC \bowtie C))$$

(4) 查询至少选修学号为 S5 的学生所选修的一门课程的学生的姓名。

$$\pi_{姓名}(\pi_{学号}(\pi_{课程号}(\sigma_{学号='S5'}(SC)) \bowtie SC) \bowtie S)$$

(5) 查询选修所有课程的学生的学号。

$$\pi_{学号,课程号}(SC) \div \pi_{课程号}(C)$$

(6) 查询不选修任何课程的学生的学号。

$$\pi_{学号}(S) - \pi_{学号}(SC)$$

(7) 检索选修了“张山”同学所选修的所有课程的学生姓名。

$$\pi_{姓名}(S \bowtie (\pi_{学号,课程号}(SC) \div \pi_{课程号}(\pi_{学号}(\sigma_{姓名='张山'}(S)) \bowtie SC)))$$

(8) 检索至少选修两门课程的学生学号。

$$\pi_{学号}(\sigma_{C\# \neq 课程号}(\rho_{SG}(S\#, C\#, G)(SC) \bowtie_{S\#=学号} SC))$$

3.3 关系演算

将数理逻辑中的谓词演算推广到关系运算中,用谓词演算来表达关系的操作,就得到了关系演算。关系演算是用查询的结果应满足的谓词条件来表达查询要求的。

如何表达元组满足的谓词条件呢?我们需要说明关系的另外一种表示方式。

3.3.1 关系演算中关系的表示

关系是一个集合,集合主要有两种表示方法:列举法和描述法。列举法是列举出集合中的元素,这种方法比较适用于有限集合。描述法用集合中的元素满足什么样的特性来表示集合,比如$\{x \mid x>3, x\in \mathrm{N}\}$表示的是所有大于 3 的整数集合。我们可以用集合描述法建立谓词与关系间的联系,用描述期望的结果形式地表达查询,而不是像关系代数那样用操作符计算结果。

在关系演算中,关系用谓词(predicate)表示,关系 R 可以看成是满足一定谓词条件的元组或属性域的集合,可表示为

$$\{u \mid R(u)\}$$

其中 u 可为元组变量或域变量，$R(u)$是一个谓词。

谓词实质上就是一个返回逻辑值的函数名。如果 R 是一个包含 n 个固定顺序的属性的关系，那么可以用 R 作为对应这个关系的谓词名；如果$(a_1,a_2,\cdots,a_n)$是 R 的元组，则 $R(a_1,a_2,\cdots,a_n)$的值为 TRUE，否则为 FALSE。

我们不去讲解前面提到的 QUEL 元组演算语言和 QBE 域演算语言那些具体的关系演算语言，而是讲解抽象的关系演算语言，目的就是为了更好地理解下一章所要学习的目前广泛使用的 SQL 查询语言。

3.3.2 元组关系演算

在$\{u|R(u)\}$中，当 u 为元组时，我们称所进行的关系演算为元组关系演算，用元组关系演算如下表达查询：

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

“|”左边的部分称为查询目标，包含一个元组变量 t，它的取值范围就是查询的结果。“|”右边的部分称为查询条件，$\varphi(t)$称为结果元组应满足的元组演算公式。

$\{t|\varphi(t)\}$表示满足元组演算公式 $\varphi(t)$的所有元组 t 的集合。

1. 元组演算公式

在元组关系演算表达式中，元组演算公式由原子公式组成。

1) 原子公式

原子公式有下面三种形式：

(1) $R(t)$：其中 R 是关系名称，t 是元组变量。$R(t)$表示 t 是 R 中的元组。关系 R 就可以表示为$\{t|R(t)\}$。

(2) $t[i]\theta u[j]$：其中 t 和 u 是元组变量，θ 是比较运算符。$t[i]\theta u[j]$表示“元组 t 的第 i 个分量与元组 u 的第 j 个分量满足比较关系 θ”，例如 $t[2]<u[3]$。

(3) $t[i]\theta C$：其中 C 是常量。$t[i]\theta C$ 表示“元组 t 的第 i 个分量与常量 C 满足比较关系 θ”，例如 $t[2]=3$。

2) 元组演算公式

元组演算公式的递归定义如下：

(1) 原子公式是公式。

(2) 设 φ_1 和 φ_2 是公式，则 $\neg\varphi_1$，$\varphi_1\wedge\varphi_2$，$\varphi_1\vee\varphi_2$，$\varphi_1\rightarrow\varphi_2$ 也是公式。

(3) 设 $\varphi(t)$是公式，t 是 $\varphi(t)$中的元组变量，则$(\exists t)\varphi(t)$，$(\forall t)\varphi(t)$也是公式。

(4) 有限次使用上述规则得到的式子都是公式。

其中，$\exists$是存在量词符号，$(\exists t)\varphi(t)$表示“若有一个 t 使 φ 为真，则$(\exists t)\varphi(t)$为真，否则为假”。$\forall$是全称量词符号，$(\forall t)\varphi(t)$表示“如果对所有 t 都使 φ 为真，则$(\forall t)\varphi(t)$为真，否则为假”。

在元组演算公式中各种运算符的优先次序如下，从上到下优先级从高到低。

(1) 算术比较符：$<$，$>$，$\leqslant$，$\geqslant$，$\neq$，$=$。