西北工業大學

数据库系统基础

Fundamentals of Database Systems

软件学院 **钱锋** 编

2024 年 4 月 16 日

数据库系统基础

钱锋[1,2]

2024 年 4 月 16 日

[1]Email: strik0r.qf@gmail.com
[2]西北工业大学软件学院, School of Software, Northwestern Polytechnical University, 西安 710072

目录

第 1 章　数据库简介

1.1　数据库与数据库管理系统

1.1.1　数据库的四个基本概念

数据 (data) 是描述客观事物的符号记录.

数据库 (database) 是长期存储在计算机内部的、有组织的、可共享的大量数据的集合, 数据库中的数据按照一定的数据模型组织、描述和储存, 具有较小的冗余度、较高的数据独立性和易拓展性, 并可为各种用户所共享.

数据库管理系统 (database management system, DBMS) 是一个用于定义、构造和操作数据库和自爱不同的用户与应用之间共享数据库的通用软件系统 (general-purpose software system). 它允许用户创建和维护数据库.

数据库系统 (database system) 是由数据库、DBMS、应用程序 (application program) 和数据库管理员 (database administer, DBA) 组成的存储、管理、处理和维护数据的系统.

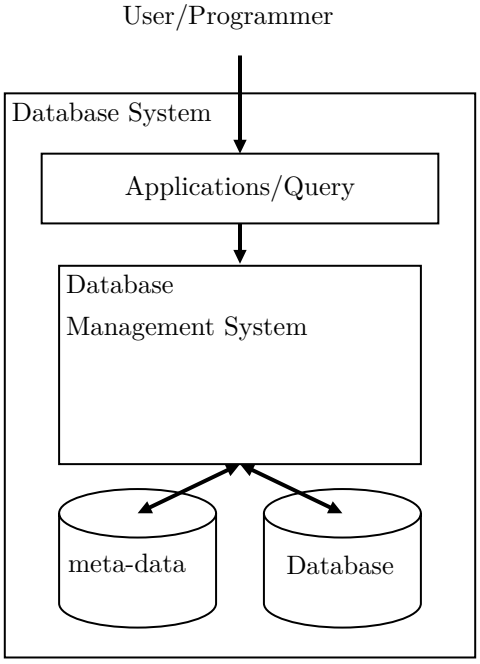


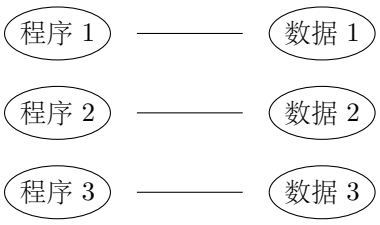
图 1.1: 一种简化的数据库系统环境

1.1.2　数据管理技术的发展

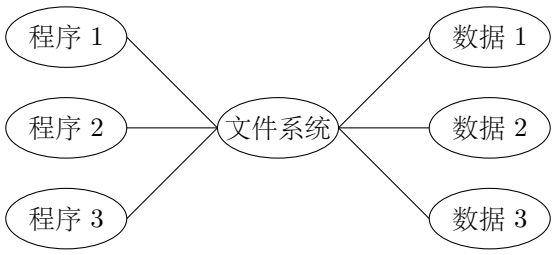
虽然数据库是当前数据管理最有效的方式之一, 在数据库技术出现之前, 数据管理还经历了两个阶段, 分别是手工管理阶段和文件系统管理阶段.

表 1.1: 不同数据管理技术发展阶段的特点

发展阶段	年代背景	应用背景	硬件条件	软件条件	数据管理
手工管理	1950s 之前	科学计算	打孔卡片、磁带; 没有磁盘	没有操作系统和专门的数据管理软件; 数据交互按照批处理方式进行	数据与程序相互依赖; 数据没有共享
文件管理	1950s-60s	科学计算; 数据管理	磁盘、磁鼓	操作系统下产生文件系统; 数据实时在线处理	数据文件相互独立; 数据共享困难, 冗余
数据库	1960s 之后	大规模数据管理	大容量磁盘	产生了专门的数据管理软件——数据库管理系统, 以满足不同的场景应用需求	实现数据的共享、透明



(a) 手工管理阶段的程序和数据基本是一一对应的, 没有数据共享



(b) 在文件系统阶段, 数据文件相互独立但是共享困难

(c) 数据库系统阶段实现了焦点分离

图 1.2: 数据库技术的不同阶段的特点

1.2　数据模型　数据库系统的模式架构

1.2.1　数据模型分类

数据模型分为**概念数据模型** (conceptual data model)、**物理数据模型** (physical data model) 和介于二者之间的**实现数据模型** (implementational data model).

- 概念数据模型按照用户的观点来为数据和信息进行建模.
- 物理数据模型描述了在计算机存储介质上存储数据的细节.

概念数据模型使用实体、属性和关系等概念来进行建模. 我们会在第 2 章中介绍 **实体—关系模型** (entity-relationship model, ER model) 和这些有关的概念.

实现数据模型包括广泛使用的**关系数据模型** (relational data model) 以及已经过时的网状模型 (network model) 和层次模型 (hierachical model), 还有一些更高级的实现数据模型的新家族成员, 例如**对象数据模型** (object data model).

1.2.2　数据库系统的模式架构

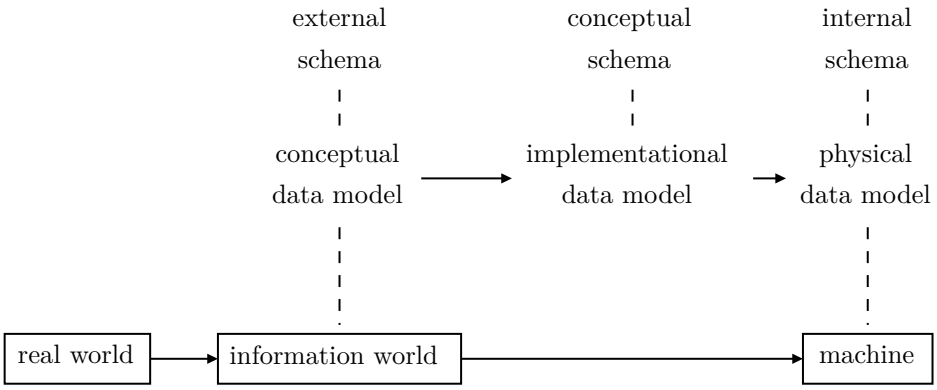


图 1.3: 数据库系统的模式架构

数据库系统的**内模式** (internal schema) 属于数据库系统的**内层** (internal level), 它描述了数据库的物理存储结构, 使用物理数据模型.

概念模式 (conceptual schema) 属于数据库系统的**概念层** (conceptual level), 它专注于描述实体、数据类型、关系、用户操作以及约束, 使用实现数据模型.

外模式 (external schema) s 呼吁数据库系统的**外层** (external level) 或**视图层** (view level), 它针对特定用户组来描述感兴趣的数据库的一部分, 并隐藏数据库的其他部分.

在不同的层次之间存在着**映射** (mapping), 外层—概念层映射实现外层与概念层的请求和结果之间的转换, 概念层—内层映射则实现概念层与内层的请求和结果之间的转换.

1.2.3　数据独立性

数据独立性 (data independence) 是指, 当改变数据库系统某一层上的模式时, 无需改变其上层的模式. 因此数据独立性又分为**逻辑数据独立性** (logical data independence) 和**物理数据独立性** (physical

data independence). 当某一层的模式改变时, 数据独立性保证了更高层上的模式可以保持不变, 只需要更改两个层次之间的映射即可. 这为数据库的设计、使用和维护带来了极大的便利.

第 2 章　实体—联系模型

例 2.0.1. CBA 赛事包括球队 (team)、球员 (player) 和比赛 (game) 的信息. 每个球队有唯一的编号 (tNo)、名称 (tName) 和所属城市 (tCity). 每个球队有多名球员, 每名球员有唯一的编号 (pNo)、姓名 (pName)、位置 (pPosition)、体重 (pWeight) 和年龄 (pAge). 每场比赛有唯一的编号 (gNo)、日期 (gDate)、地点 (gLocation) 和主客场球队. 每场比赛有多个球员参与, 每个球员在比赛中有特定的表现数据, 例如得分 (pgPoints)、助攻 (pgAssists) 和篮板 (pgRebounds). 根据以上信息构造的满足数据需求的实体—联系 (ER) 模型如图 2.1 所示.

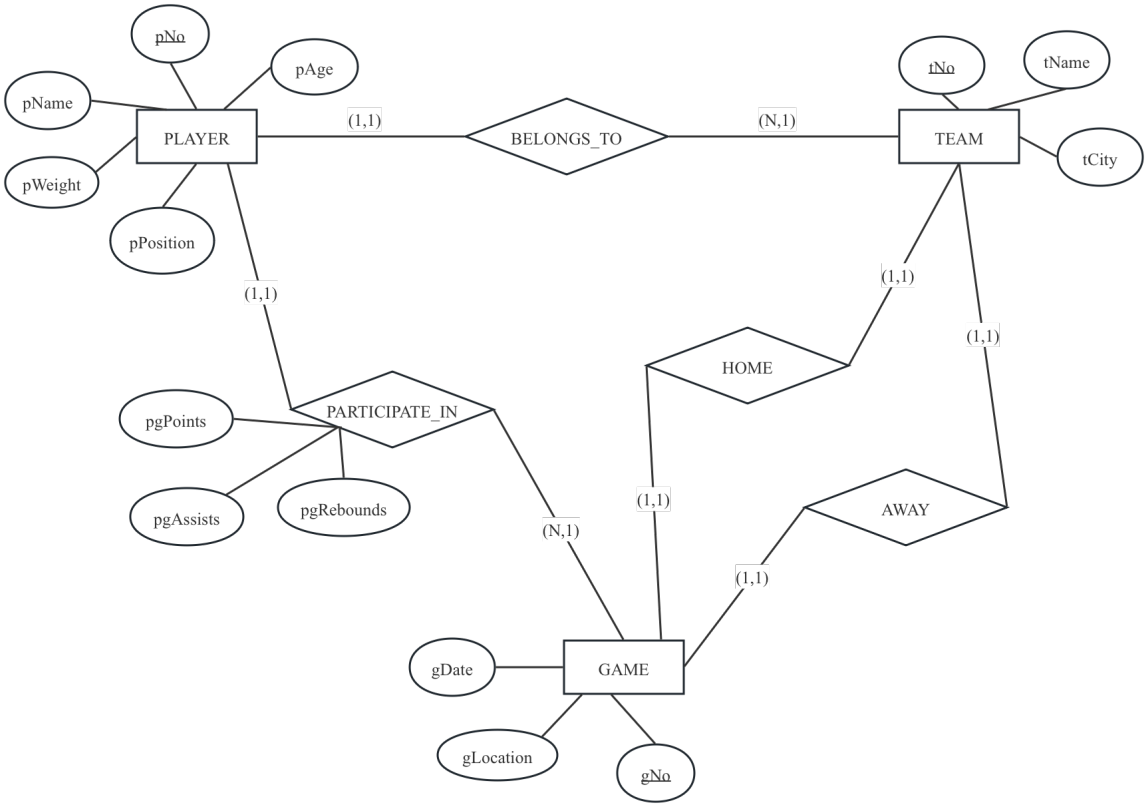


图 2.1: CBA 赛事的 E-R 建模

上图中需要特别说明的是, 一场比赛的主客场球队是根据主场 (home) 和客场 (away) 关系来确定的.

例 2.0.2. 某医院有多个科室 (department), 科室包含有相关编号 (dNo) 和名称 (dName). 每个科室有多

名医生 (doctor), 每名医生有各自的工号 (eNo)、姓名 (eName)、出生日期 (eBirthDate)、工龄 (eYearsOfService)、民族 (eEthicity)、籍贯 (ePlaceOfOrigin) 等信息. 每名医生只能在一个科室中工作, 但可以参与多个医学项目 (medical project, 即 project), 每个项目可由多名医生参加. 参加相应医学项目时, 有相关的项目周期 (pDuration)、名称 (pName)、项目经费 (pBudget)、项目简介 (pInfo) 等信息. 根据以上信息构造的满足数据需求的实体—联系 (ER) 模型如图 2.2 所示.

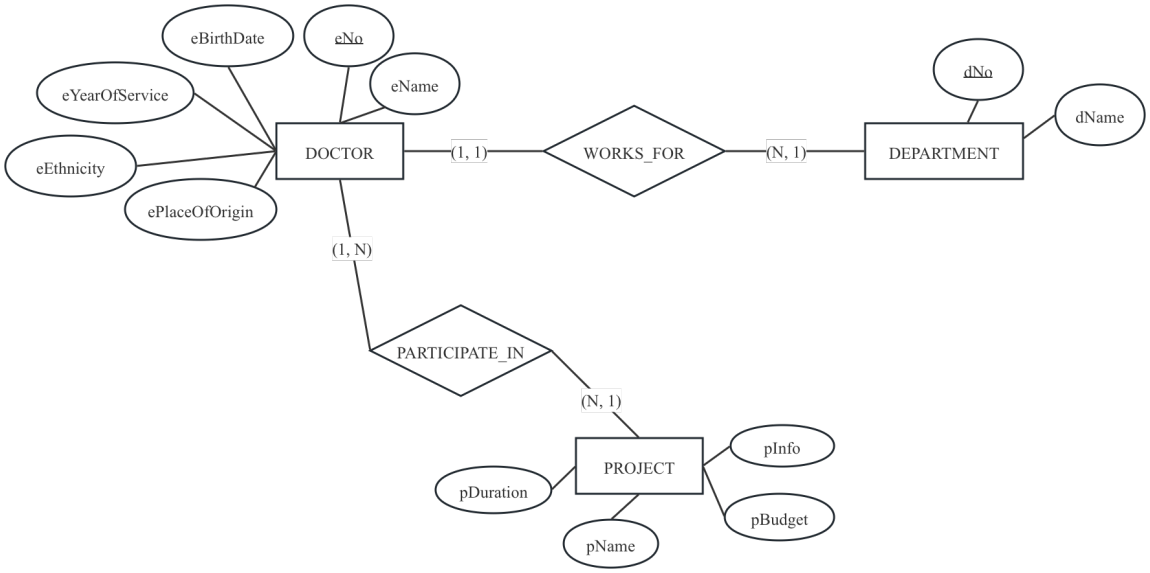


图 2.2: 医院内部的 E-R 建模

为了避免混淆, 我们在定义医生的有关属性时采用了 e 开头 (毕竟医生从某种角度上讲可以看作是医院的雇员, 即 employee).

第3章　关系数据模型和关系数据库约束

第 4 章　关系代数和关系演算

本章我们讨论关系模型的两种形式话语言, 分别是关系代数和关系演算.

关系代数 (relational algebra) 是形式话关系模型的基本操作集, 这些操作使用户能够将基本的检索请求指定为关系代数表达式.

关系演算 (relational calculus) 则提供了用于指定关系查询的更高级的声明性语言.

我们在本章中先介绍这些关系代数的运算规则, 然后再通过具体的例子的形式来解释这些运算的相关使用注意事项.

4.1　一元关系运算: 选择和投影

4.1.1　选择运算

分离公理告诉我们, 对于集合 A, 我们总能根据性质 P 来确定 A 的一个子集, 这个子集中的所有元素都是 A 中满足性质 P 的元素. 由此我们得到了关系的选择运算. 它从一个关系当中, "筛选"出符合某个条件的元组来, "过滤"掉那些不符合条件的元组.

定义 4.1.1. (选择运算). 设 $\mathcal{R}$ 是一个关系, $t \in \mathcal{R}$ 是关系 $\mathcal{R}$ 的一个元组, $p(t)$ 是在关系 $\mathcal{R}$ 的属性上指定的一个 Bool 表达式. 那么子集

$$\sigma_{p(t)}(\mathcal{R}) := \{t \in \mathcal{R} | p(t) = \text{True}\}$$

称为关系 $\mathcal{R}$ 在条件 $p(t)$ 下的**选择** (selection) 或者**过滤器** (filter), Bool 表达式 $p(t)$ 称为**选择条件** (selection condition).

注. $\mathcal{R}$ 是一个关系, 也可以是一个**关系代数表达式** (relational algebratic expression). 这是因为关系代数表达式是有关系运算序列构成的, 其结果也是一个关系.

注. 关系 $\mathcal{R}$ 的选择 $\sigma_{p(t)}(\mathcal{R})$ 的属性与 $\mathcal{R}$ 的属性相同, 这是因为选择运算并没有改变关系 $\mathcal{R}$ 中元组的分量.

注. 选择条件 $p(t)$ 一般是由许多的**子句** (clause) 组成的, 子句之间可以用 $\wedge, \vee, \neg$ 进行复合和连接.

注. 并不是所有的关系符号都可以用在选择条件的构造中, 例如, 如果被选择的属性的域是一个无序域的话, 那显然不可以利用 $\leqslant$ 关系来进行选择.

从选择运算的定义立即得到, 选择运算符 σ 是一个**一元** (unary) 运算符, 选择运算将单独应用于每个元组. 这是因为我们在进行关系的选择运算的时候, 往往是遍历关系 $\mathcal{R}$ 的所有元组 t, 分别检验它们是否满足选择条件 $p(t)$.

此外, 由于选择运算最终确定的是关系 $\mathcal{R}$ 的一个子集, 而关系 $\mathcal{R}$ 在数据库系统的范畴内是一个有限集合, 因此我们知道对于任意的选择条件 p, 都有

$$\sigma_p(\mathcal{R}) \leqslant |\mathcal{R}|.$$

它的意思是说, 对一个关系做选择运算, 不可能得到比原来更多的元组. 从常理上来说这是容易理解的, 从一个有限集合中排除掉一些不符合条件的元素后, 剩下的部分怎么会反而变多呢? 这又不是教育改革双减. 选择运算的这一性质, 让我们得以自然的引出下面的概念:

定义 4.1.2. (选中率). 设 $\sigma_p(\mathcal{R})$ 是基于选择条件 p 对关系 $\mathcal{R}$ 的一个选择. 我们称数 $|\sigma_p(\mathcal{R})|/|\mathcal{R}|$ 为条件 p 的**选中率** (selectivity), 即

$$\text{selectivity}(p, \mathcal{R}) = \frac{|\sigma_p(\mathcal{R})|}{|\mathcal{R}|},$$

它是两个有限集合 $\sigma_p(\mathcal{R})$, $\mathcal{R}$ 中元素数量的比值.

我们要进一步说明的是, 选择运算是**可交换** (commutative) 的, 这是因为我先做一次选择可以确定一个子集, 在这个选择的基础上再做一次选择, 本质上就是按照条件 p_1 和 p_2 的合取 $p_1 \wedge p_2$ 来进行选择. 因此根据命题的合取的交换性, 我们知道选择运算是可交换的, 而且若干个选择运算构成的关系代数表达式可以利用选择条件的合取合并成一次选择运算, 即

$$\sigma_{p_n}(\sigma_{p_{n-1}}(\cdots\sigma_{p_2}(\sigma_{p_1}(\mathcal{R})))) = \sigma_{\bigwedge_{k=1}^{n} p_k}(\mathcal{R}).$$

例 4.1.1. 假设 EMPLOYEE 是一个关系, 它的元组给定了某企业中的一个员工的信息, 它的属性是 No, Name, Sex 和 Salary. 那么月薪在 5000 元以上的选择就可以被表示为 $\sigma_{\text{Salary}>5000}(\text{EMPLOYEE})$. 性别为"武装直升机"(armed helicopter) 的选择就可以被表示为 $\sigma_{\text{Sex}='\text{armed\_helicpter}'}(\text{EMPLOYEE})$. 如果我们要查询月薪在 5000 元以上, 且性别为武装直升机的数据, 那么我们可以定义查询

$$\sigma_{(\text{Salary}>5000)\wedge(\text{Sex}='\text{armed\_helicopter}')}(\text{EMPLOYEE}).$$

4.1.2 投影运算

投影运算从关系中选择某些列, 并且会丢弃其他的列. 如果只对一个关系的某些属性感兴趣, 就可以利用投影运算, 将关系投影到这些属性上.

定义 4.1.3. (投影运算). 设 $\mathcal{R}$ 是一个关系, $[A_1, A_2, \cdots, A_n]$ 是关系 $\mathcal{R}$ 的属性, 那么

$$\pi_{A_i}(\mathcal{R}) = \{a_i | t \in \mathcal{R}\}.$$

称为关系 $\mathcal{R}$ 在属性 A_i 上的投影.

我们可以把关系的投影运算推广到有限个属性的情形, 如果属性列表 $A_1, A_2, \cdots, A_n$ 是关系 $\mathcal{R}$ 的 n 个属性. 那么

$$\pi_{[A_1,A_2,\cdots,A_n]} = \{[a_1, a_2, \cdots, a_n] | t \in \mathcal{R}\}$$

就是关系 $\mathcal{R}$ 在属性列表 $A_1, A_2, \cdots, A_n$ 上的投影, 其中每一个元组中各个属性的值出现的顺序与在属性列表中指定的顺序相同.

由于集合中的元素是互异的, 所以对关系 $\mathcal{R}$ 的非键属性进行投影, 就会有重复的元组, 而这些元组在集合中被视为同一个元素. 这就是投影运算的 **重复消除** (duplicate elimination) 现象.

注. 如果不消除重复元素, 那么得到的就不是一个集合, 而是包含重复元组的**包**, 这在形式化关系模型中是不允许的, 但是在 SQL 中是允许的. 在 SQL 中, 如果在查询的 `SELECT` 子句投影了一个属性列表后, 不使用 `DISTINCT` 来从查询中删除关键字, 那么查询结果将会包含重复的元组.

例 4.1.2. 某高校数据库系统中有以下关系:

表 4.1: COURSE

Course	Department
Mathematical Analysis	Math
Advanced Algebra	Math
Fundamentals of Computer Programming	Computer
Introduction to Software Engineering	Software

执行投影运算将关系 COURSE 投影到属性 Department 上, 就可以获得所有课程的开课部门组成的集合. 这就是说, 我们有查询结果

$$\pi_{\text{Department}}(\text{COURSE}) = \{[\text{Math}], [\text{Computer}], [\text{Software}]\}.$$

注意到数学分析 (Mathematical Analysis) 和高等代数 (Advanced Algebra) 两门课程都是数学与统计学院 (Math) 开设的, 但是在投影运算投影到开课部门 (Department) 属性后, 它们就成为一个元素了.

投影运算得到的元组数量总是少于关系 $\mathcal{R}$ 中的元组数量. 这是因为, 如先前所述, 当投影到一个关系 $\mathcal{R}$ 的非键属性时, 得到的是去除了重复元素之后的集合. 但是当我们投影到某个键属性上时, 得到的元组数量就会等于关系 $\mathcal{R}$ 的元组数量了.

习题 4.1.1. 举例说明投影运算不具有可交换性.

习题 4.1.2. 假设 list1 $:= [A_1, A_2, \cdots, A_n]$ 是关系 $\mathcal{R}$ 的一个属性列表, list2 $:= [A_{i1}, A_{i2}, \cdots, A_{ir}]$ 是它的一个子列表. 尝试说明为什么

$$\pi_{\text{list1}}\left(\pi_{\text{list2}}(\mathcal{R})\right)$$

是一个错误的关系代数表达式.

4.1.3 RENAME 运算

为了建立一个查询, 我们可能需要重复进行多次关系代数运算, 从而得到一个关系代数表达式. 这种多个运算及其嵌套得到的表达式称为**内联表达式** (in-line expression), 但有时, 一次应用一个运算, 并将中间结果用合适的名称来表示, 会使得整个过程更简洁, 这与程序代码追求可读性是一样的. 这就需要对中间关系和结果关系及其属性进行**重命名** (rename). 于是我们定义了如下运算:

定义 4.1.4. (重命名运算). 设 $\mathcal{R}$ 是一个关系, $[A_1, A_2, \cdots, A_n]$ 是关系 $\mathcal{R}$ 的属性. 那么 $\rho_{\mathcal{S}(B_1,B_2,\cdots,B_n)}(\mathcal{R})$ 是一个关系, 它表示

$$\{[b_1, b_2, \cdots, b_n] | \exists! t \in \mathcal{R}\,(a_1 = b_1, a_2 = b_2, \cdots, a_n = b_n)\}$$

称为关系 $\mathcal{R}$ 的一个**重命名** (rename), 其中 ρ 称为 RENAME 运算符, $\mathcal{S}$ 为新的关系, $[B_1, B_2, \cdots, B_n]$ 为新的属性, 新的属性与 $[A_1, A_2, \cdots, A_n]$ 在顺序上是相同的.

我们也可以使用二元运算符的赋值运算来进行关系的重命名, 我们认为执行语句

$$\mathcal{S}[B_1, B_2, \cdots, B_n] \longleftarrow \mathcal{R}[A_1, A_2, \cdots, A_n]$$

得到的关系 $\mathcal{S}$ 与关系 $\rho_{\mathcal{S}(B_1,B_2,\cdots,B_n)}(\mathcal{R})$ 是等价的.

4.2 集合论中的关系代数运算

4.2.1 并集、交集和差集

可以使用集合论运算的方法来处理两个元组的集合, 即两个关系. 包括并集 (union)、交集 (intersection) 和差集 (difference, 或称为 except). 但参与这三种运算的两个关系必须具有相同的**元组类型** (type of tuple). 于是我们首先定义两个关系的类型兼容性:

定义 4.2.1. 对于关系 $\mathcal{R}(A_1, A_2, \cdots, A_n)$ 和 $\mathcal{S}(B_1, B_2, \cdots, B_n)$, 如果同时满足下列条件:

1° $\deg\mathcal{R} = \deg\mathcal{S}$, 即它们具有相同的度 (degree, 即属性个数) n;

2° $\forall i \in [a, n] \cap \mathbb{Z}\,(\mathrm{dom}(A_i) = \mathrm{dom}(B_i))$, 即每个对应的属性都具有相同的域 (domain),

则称关系 $\mathcal{R}$ 和关系 $\mathcal{S}$ 是**类型兼容** (type comtatible) 或**并兼容** (union compatible) 的.

定义 4.2.2. 设 $\mathcal{R}, \mathcal{S}$ 是两个并兼容的关系, 定义

1° 关系的并: $\mathcal{R} \cup \mathcal{S} := \{t | (t \in \mathcal{R}) \vee (t \in \mathcal{S})\}$, 其中包括 $\mathcal{R}$ 或 $\mathcal{S}$ 或它们二者中的所有元组;

2° 关系的交: $\mathcal{R} \cap \mathcal{S} := \{t | (t \in \mathcal{R}) \wedge (t \in \mathcal{S})\}$, 其中包括既在 $\mathcal{R}$ 中又在 $\mathcal{S}$ 中的所有元组;

3° 关系的差: $\mathcal{R} - \mathcal{S} := \{t | (t \in \mathcal{R}) \wedge (t \notin \mathcal{S})\}$, 其中包括在 $\mathcal{R}$ 中但不在 $\mathcal{S}$ 中的所有元组.

注. 并运算和交运算都是可交换、可结合 (associative) 的.

4.2.2 Descartes 积运算

定义 4.2.3. (Descartes 积). 设 $\mathcal{R}(A_1, A_2, \cdots, A_m)$, $\mathcal{S}(B_1, B_2, \cdots, B_n)$ 是两个关系, $(a_1, a_2, \cdots, a_m)$ 和 $(b_1, b_2, \cdots, b_n)$ 是它们的元组. 那么

$$\mathcal{R} \times \mathcal{S} := \{(a_1, a_2, \cdots, a_m, b_1, b_1, \cdots, b_n) = (r, s) | (r =\in \mathcal{R}) \wedge (s =\in \mathcal{S})\}$$

称为关系 $\mathcal{R}$ 与关系 $\mathcal{S}$ 的 **Descartes 积** (Descartesian product).

4.3　关系代数查询　作业

已知关系模式

```
department(dNo, dName, officeRoom, homePage)
student(sNo, sName, sex, age, dNo)
course(cNo, cName, cPNo, credit, dNo) // cPNo 为先修课程
sc(sNo, cNo, score, recordDate)
```

用关系代数完成以下查询.

4.3.1　基本查询

例 4.3.1. 所有年龄小于 18 岁的男生姓名.

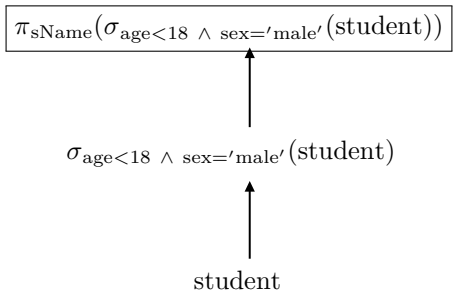


图 4.1: 年龄小于 18 岁的男生的姓名的查询树

评注. 我们可以使用如图 4.1 所示的查询树来表达关系代数表达式的建立过程, 在该查询中, 我们首先筛选出了所有年龄小于 18 且性别为男性的 student 元组, 然后将其投影到姓名属性上.

例 4.3.2. 查询所有学分大于 3 的课程名称.

解. 此后对于一些简单的查询, 我们不再给出查询树, 为了获得查询结果, 我们首先需要在 course 关系中筛选出学分大于 3 的元组来, 即 $\sigma_{\text{credit}>3}(\text{course})$, 然后我们将其投影到 cName 属性上, 因此关系代数表达式

$$\pi_{\text{cName}}\left(\sigma_{\text{credit}>3}(\text{course})\right)$$

即为所求.

例 4.3.3. 查询没有先修课程的课程名称.

解. 为了获取查询结果, 我们需要考虑 course 中没有先修课程的元组, 即

$$\sigma_{\text{cPNo=NULL}}(\text{course}),$$

然后将其投影到 cName 属性上, 因此

$$\pi_{\text{cName}}\left(\sigma_{\text{cPNo=NULL}}(\text{course})\right)$$

即为所求. 注意在 SQL 中表达一个值是否为 `NULL` 通常采用 `IS` 和 `IS NOT` 关系符, 但为了方便起见, 我们关系代数表达式中我们采用相等运算符 $=$.

4.3.2 连接查询

例 4.3.4. 查询“信息学院”所有学生的姓名.

解. 为了得到查询结果, 我们首先获取 department 表中信息学院 (School of Information Science, SIS) 所在的元组, 并将其投影到 dNo 属性上,

$$\pi_{\mathrm{dNo}}\left(\sigma_{\mathrm{dName}='\mathrm{SIS}'}(\mathrm{department})\right),$$

接下来将其与 student 表基于 dNo 属性进行自然连接,

$$\mathrm{student} \bowtie_{\mathrm{dNo}} \pi_{\mathrm{dNo}}\left(\sigma_{\mathrm{dName}='\mathrm{SIS}'}(\mathrm{department})\right),$$

所得到的就是信息学院中所有学生的信息. 这是因为不满足 dNo 等于学院的元组将会作为悬浮的元组被剔除, 将其投影到 sName 属性上,

$$\pi_{\mathrm{sName}}\left(\mathrm{student} \bowtie_{\mathrm{dNo}} \pi_{\mathrm{dNo}}\left(\sigma_{\mathrm{dName}='\mathrm{SIS}'}(\mathrm{department})\right)\right)$$

即为所求.

评注. 由于在 LaTeX 公式中输入汉字较为不便, 我们采用英文来作为名称给定关系代数表达式.

评注. 连接运算总是消耗时间的, 因此我们尽量减少参与连接运算的元组的个数以获得更高的执行速度, 但是由于本节中我们主要关注给出能够得到查询结果的有效的关系代数表达式, 因此本节中我们给出的关系代数表达式不一定是效率最优的.

例 4.3.5. 查询所有具有不及格记录的学生姓名.

解. 为了得到查询结果, 我们首先需要将学生的姓名“附加”到 sc 表中, 而附加这一字段需要参照 sNo 属性, 因此我们首先把 student 表投影到 sNo 和 sName 属性上,

$$\pi_{\mathrm{sName,sNo}}(\mathrm{student}),$$

接下来, 利用 sNo 属性, 将 $\pi_{\mathrm{sName,sNo}}(\mathrm{student})$ 与 sc 表做一个自然连接,

$$\mathrm{sc} \bowtie_{\mathrm{sNo}} \pi_{\mathrm{sName,sNo}}(\mathrm{student}),$$

所得到的表实际上就是在 sc 表中新增了 sName 字段的结果. 接下来, 我们筛选出课程考核不及格的元组,

$$\sigma_{\mathrm{scoer}<60}\left(\mathrm{sc} \bowtie_{\mathrm{sNo}} \pi_{\mathrm{sName,sNo}}(\mathrm{student})\right),$$

然后把它投影带 sName 属性上, 由于投影运算将会自然地消除重复的元组, 所以

$$\pi_{\mathrm{sName}}\left(\sigma_{\mathrm{scoer}<60}\left(\mathrm{sc} \bowtie_{\mathrm{sNo}} \pi_{\mathrm{sName,sNo}}(\mathrm{student})\right)\right)$$

即为所求.

例 4.3.6. 查询“信息学院”所有年龄小于 18 岁的女生的姓名.

解. 为了得到查询结果, 我们首先获取 department 表中信息学院 (School of Information Science, SIS) 所在的元组, 并将其投影到 dNo 属性上,

$$\pi_{\mathrm{dNo}}\left(\sigma_{\mathrm{dName}='\mathrm{SIS}'}(\mathrm{department})\right),$$

接下来我们从 student 表中筛选出年龄小于 18 岁且年龄为女的元组,

$$\sigma_{\text{age}<18\wedge\text{sex}='\text{Female}'}(\text{student}),$$

这是 student 表中所有的未成年女生组成的子表, 我们接下来将其与信息学院的元组进行自然连接,

$$\sigma_{\text{age}<18\wedge\text{sex}='\text{Female}'}(\text{student}) \bowtie_{\text{dNo}} \pi_{\text{dNo}}(\sigma_{\text{dName}='\text{SIS}'}(\text{department})),$$

由于 dNo 不对应于信息学院的 student 元组会被自然连接运算作为悬浮的元组剔除, 因此这就得到了信息学院中所有未成年女生的记录, 接下来将其投影到 sName 属性上,

$$\pi_{\text{sName}}(\sigma_{\text{age}<18\wedge\text{sex}='\text{Female}'}(\text{student}) \bowtie_{\text{dNo}} \pi_{\text{dNo}}(\sigma_{\text{dName}='\text{SIS}'}(\text{department})))$$

即为所求.

第 5 章　SQL 基础

SQL 是**结构化查询语言** (Structured Query Language) 的缩写, 是一种综合性的数据库语言. 它的前身是结构化查询语言 (Stuructured English QUEry Language, SEQUEL), 在美国国家标准学会 (American National Standards Institute, ANSI) 和国际标准化组织 (International Standards Organization, ISO) 的共同努力下实现了 SQL 的标准化.

关于 SQL 的历史我们不再赘述, 感兴趣的读者可以自行了解. 本章我们以 PostgresSQL 为例介绍一些基础的 SQL 语法, 需要注意的是, 各种 SQL 实现实际上都遵循着大体上相似的原则, 因此如果你的课程并不主要使用 PostgresSQL, 而是使用 Oracle 或者 MySQL 这样的其他的 DBMS 软件, 你依然可以参考本书来完成一些基本的 SQL 语法的学习, 在需要学习特定的 SQL 实现的特性的时候, 再转到更专业更深入的材料中学习.

5.1　SQL 中的数据定义和数据类型

5.1.1　SQL 中的 `CREATE TABLE` 命令

`CREATE TABLE` 命令用于指定一个新关系, 为此, 你需要指定新关系的名称、属性及其类型和一些约束条件 (如果需要的话).

例 5.1.1. 下述命令创建了一个“天气”关系 (方便起见, 我们以后都把关系称作**表**, 即 table), 它有五个属性 (我们以后把属性称为**列**或者字段, 即 column, 字段是在 Microsoft Access 中对关系的属性的称呼), 分别是城市 (city)、最高温度 (high temperature, temp_hi)、最低温度 (low temperature, temp_lo)、降水 (precipitation) 和日期 (date).

```
CREATE TABLE weather (
    city varchar(80),
    temp_lo int,
    temp_hi int,
    prcp real,
    date date
);
```

5.2　SQL 中的基本检索查询

```
department(dNo,dName,officeRoom,homepage)
student(sNo,sName,sex,age,dNo)
course(cNo,cName,cPNo,credit,dNo)
```

```
sc(sNo,cNo,score,recordDate)
```

5.2.1 基本 SQL 查询的 SELECT-FROM-WHERE 结构

SELECT 语句的基本形式也称为**映射** (mapping) 或 SELECT-FROM-WHERE 块 (select-from-where block). 它的形式为

```
SELECT <AttributeList>
FROM   <TableList>
WHERE  <Condition>
```

其中, <AttributeList> 是一个属性名称的列表, 查询将通过该列表来检索属性的值.

例 5.2.1. 查询所有年龄大于等于 20 岁的学生学号、姓名

```
SELECT sNo, sName
FROM student
WHERE age >= 20;
```

例 5.2.2. 查询所有姓钱的男生学号、姓名、出生年份.

```
SELECT sNo, sName, EXTRACT(YEAR FROM CURRENT_DATE) - age AS birthYear
FROM student
WHERE sName LIKE '钱%' AND sex = '男';
```

例 5.2.3. 查询所有学分大于 3 的课程名称.

```
SELECT cName
FROM course
WHERE credit > 3;
```

例 5.2.4. 查询所有没有被分配到任何学院的学生姓名.

```
SELECT sName
FROM student
WHERE dNo IS NULL;
```

例 5.2.5. 查询所有尚未设置主页的学院名称.

```
SELECT dName
FROM department
WHERE homepage IS NULL;
```

例 5.2.6. 查询各个学院的平均年龄;

```
SELECT dNo, AVG(age) AS Average_Age
FROM student
GROUP BY dNo;
```

例 5.2.7. 查询每个学生选修课程的平均分;

```
SELECT AVG(sc) AS Average_Score
FROM sc
GEOUNP BY sNo;
```

例 5.2.8. 查询各课程的平均分;

例 5.2.9. 查询各学院开设的课程门数;

例 5.2.10. 查询各门课程选修人数。

第 6 章　函数依赖和关系数据库规范化的基础知识

6.1　函数依赖

设 $R = \{A_1, A_2, \cdots, A_n\}$ 是 n 个属性组成的属性组, 我们也将其称为单个**泛关系模式** (universal relational schema). 本章讨论的目的, 是建立一种符合 R 的关系状态 $\mathcal{R}$.

定义 6.1.1. 设 X, Y 是 R 的两个子集, 这就是说, X 和 Y 是 R 中的两个子属性组, 如果对于关系模式 R 中任意时刻的一个可能的关系 $\mathcal{R}$, $\mathcal{R}$ 中任意的元组 t_1 和 t_2, 都满足从 $t_1[X] = t_2[X]$ 能够推出 $t_1[Y] = t_2[Y]$, 即

$$\forall t_1, t_2 \in \mathcal{R}\,(t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y])\,,$$

即 $\mathcal{R}$ 中不可能存在两个元组使得它们在 X 上的属性值相同而在 Y 上的属性值不同, 则称属性组 Y **函数依赖** (functional dependency) 于 X, 记作 $X \to Y$.

换言之, 属性组 X 中的一个值能够唯一确定属性组 Y 中的一个值. 这与我们在数学上接触过的函数的概念是一致的. 如果 Y 不函数依赖于 X, 则记作 $X \nrightarrow Y$. 函数依赖定义中的"任意"说明, 函数依赖不是指关系模式的某个或者某些关系实例满足约束条件, 而是指所有关系实例均要满足这一约束条件.

例 6.1.1. 对于存储于学校教务系统的一个学生来说, 有一关系 student(sNo, sName, sDepartment, sMajor), 在这一关系中, $\{\text{sNo}\} \to \{\text{sName}, \text{sDepartment}, \text{sMajor}\}$, 这就是说, 只要确定一个同学的学号, 就能够**唯一地确定** TA 的姓名、学院和专业, 例如, 如果一个学生 s 满足

$$s[\text{sNo}] = 2021114514,$$

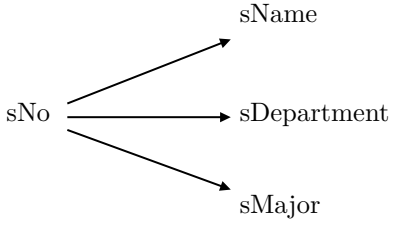
那么必然满足

$$s[\text{sName}] = \text{Strik0r},$$

$$s[\text{sDepartment}] = \text{Software},$$

$$s[\text{sMajor}] = \text{Software\_Engineering}.$$

你可以用图示的方法来明确的表达属性之间的函数依赖关系:



定义 6.1.2. 关系模式 $R=\{A_1,A_2,\cdots,A_n\}$ 的一个属性集合 $S\subset R$, 如果在任意的合法关系状态 $\mathcal{R}$ 中, 对于任意的两个元组 t_1,t_2 必有 $t_1[S]\neq t_2[S]$, 则称 S 是 R 的一个**超键**.

这就是说, 如果一个属性组是超键, 那么在合法的关系状态中, 元组的超键的值是唯一的, 即通过超键能够唯一地确定一个元组.

定义 6.1.3. 关系模式 $R=\{A_1,A_2,\cdots,A_n\}$ 的一个属性集合 $K\subset R$ 如果同时满足以下两个条件:

1° $\forall\mathcal{R}(R)\forall t_1t_2\,(t_1[K]\neq t_2[K])$, 即 K 是 R 的一个超键.

2° 对于 K 中任意的一个属性 A_i, 属性组 $K-\{A_i\}$ 都不再是 R 的超键.

则称 K 是 R 的一个**键**.

超键与键之间的关系, 就像上界与上确界之间的关系[1]. 上确界是最小上界, 同理, 键是不能再删除属性的超键.

如果一个关系模式具有多个键, 则把这些键称为候选键, 可以从中任意地选定一个作为**主键**, 其他的候选键则称为辅键. 在关系型数据库中, 每个关系模式都必须有一个主键.

定义 6.1.4. (主属性). 设 $R=\{A_1,A_2,\cdots,A_n\}$ 是一个关系模式, K 是 R 的一个候选键. 如果属性 $A_i\in K$, 即属性 A_i 是键属性组 K 的成员, 那么称 A_i 为 R 的一个**主属性** (prime attribute). 如果一个属性不是任何候选键的成员, 则称它为**非主属性** (nonprime attribute).

6.2 基于主键的范式

表 6.1: 基于主键的范式

范式	内容	校正方法
第一范式 1NF	关系中不应该包含多值属性或嵌套关系	为每个多值属性或嵌套关系组建新关系
第二范式 2NF	对于其主键包含多个属性的关系, 不应该存在非主属性对主属性的部分依赖	分解原来的关系, 为每个部分键及其相关属性建立一个新关系, 确保在一个关系中保留原始的主键以及完全函数依赖于它的任何属性
第三范式 3NF	关系中不应该有一个非键属性对另一个非键属性的函数依赖, 即不应该有一个非键属性传递依赖于主键	分解原来的关系并建立一个关系, 其中包括的非键属性可以函数确定其他的非键属性.

6.3 Boyce-Codd 范式

例 6.3.1. 现有关系模式: $R(A,B,C,D,E)$, 存在函数依赖 $ABC\to DE, AC\to D, D\to E$. 问:

[1] 如果读者想要了解有关有界数集的界与确界的更多内容, 可以参考笔者编写的《数学分析》.

1) 关系 R 属于第几范式?

解. $R \in$ 1NF, 这是因为

$$(ABC)^+ = \{A, B, C, D, E\},$$

因此 ABC 是 R 的键, 不妨选取其为主键, 则 A, B, C 为主属性, D, E 为非主属性, 不难发现由于 $AD \to D$, 因此存在 $ABC \xrightarrow{\text{Partial}} D$, 即非主属性 D 对键 ABC 的部分依赖.

2) 若关系 R 不属于 BCNF, 请将其逐步分解为 BCNF. 要求写出每一级范式的分解过程.

- $R_{21} = (\{A, C, D, E\}, \{AC \to D, D \to E\})$, $R_{22} = (\{A, B, C\}, \{ABC \to ABC\})$.
- 注意到在 R_{21} 中存在非主属性 E 对主属性 AC 的传递依赖, 即 $AC \xrightarrow{\text{Transitive}} E$, 于是 $R_{21} \notin$ 3NF.

  $R_{31} = (\{A, C, D\}, \{AC \to D\})$, $R_{32} = (\{D, E\}, \{D \to E\})$, $R_{33} = R_{22}$.
- 注意到此时 R_{31}, R_{32}, R_{33} 中同时也不存在主属性对主键的部分依赖, 因此它实际上已经满足 Boyce-Codd 范式, 即 $R_{31}, R_{32}, R_{33} \in$ BCNF.

例 6.3.2. 现有关系模式: $R(A, B, C, D, E, F)$, 存在函数依赖 $B \to C, A \to D, AB \to E, D \to F$. 问:

1) 关系 R 属于第几范式?

解. 由于

$$(AB)^+ = \{A, B, C, D, E, F\},$$

因此 AB 为关系 R 的键 (容易验证 $B^+ = \{B, C\}$, $A^+ = \{A, D, F\}$, 它们都不是超键), 选择其作为主键. 由于 $B \to C$, 因此存在非主属性 C 对主键 AB 的部分依赖 $AB \xrightarrow{\text{Partial}} C$. 这就是说, $R \in$ 1NF.

2) 若关系 R 不属于 BCNF, 请将其逐步分解为 BCNF. 要求写出每一级范式的分解过程.

- 为了消除非主属性对主键的部分依赖, 我们令

$$R_{21} = (\{B, C\}, \{B \to C\}),$$

$$R_{22} = (\{A, D, F\}, \{A \to D, D \to F\}),$$

$$R_{23} = (\{A, B, E\}, \{AB \to E\}).$$

  则它们属于第二范式, 即 $R_{21}, R_{22}, R_{23} \in$ 2NF.
- 注意到在 R_{22} 中, 由于 $A \to D, D \to F$, 即存在非主属性 F 对键 A 的传递依赖 $A \xrightarrow{\text{Transitive}} F$, 因此 $R_{22} \notin$ 3NF, 为此, 令

$$R_{31} = R_{21},$$

$$R_{32} = (\{A, D\}, \{A \to D\}),$$

$$R_{33} = (\{D, F\}, \{D \to F\}),$$

$$R_{34} = R_{23},$$

  则它们属于第三范式, 即 $R_{31}, \cdots, R_{34} \in$ 3NF.
- $R_{31}, \cdots, R_{34} \in$ BCNF, 这是因为在它们中不存在主属性对主键的部分依赖.

第 7 章　数据库设计的规范化问题

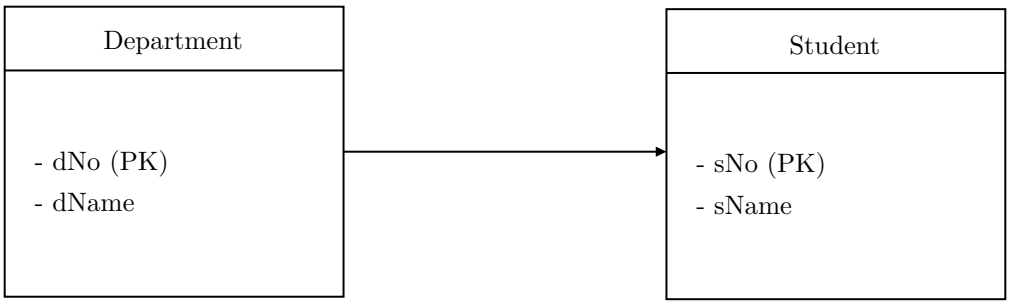


图 7.1

第8章　事務処理

参考文献

[1] [美] Rames Elmasri, [美] Shamkant B. Navathe. 数据库系统基础: 第 7 版 (Fundamentals of Database Systems, 7e)[M]. 陈宗斌等译. 北京: 清华大学出版社, 2020.

公诚勇毅　永矢毋忘
中华灿烂　工大无疆

2023 年　西北工业大学　中国西安