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 13 日

数据库系统基础

钱锋[1,2]

2024 年 4 月 13 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.

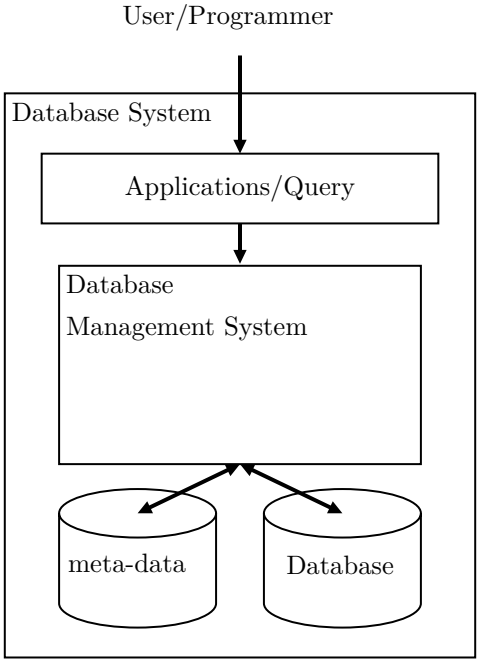


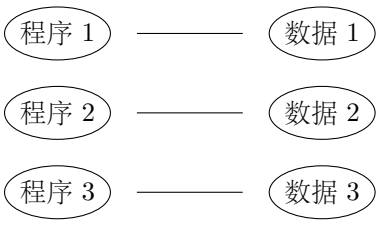
图 1.1: 一种简化的数据库系统环境

1.1.2　数据管理技术的发展

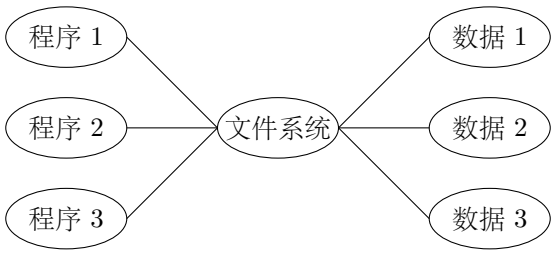
虽然数据库是当前数据管理最有效的方式之一, 在数据库技术出现之前, 数据管理还经历了两个阶段, 分别是手工管理阶段和文件系统管理阶段.

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

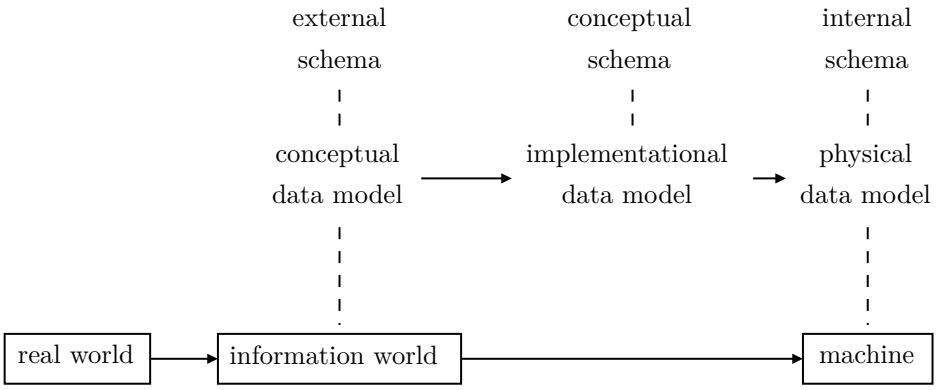


图 1.3: 数据库系统的模式架构

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—关系模型

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 [2] 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).

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.

5.1　SQL 中的基本检索查询

5.1.1　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.

第 6 章　函数依赖和关系数据库规范化的基础知识

第 7 章　数据库设计的规范化问题

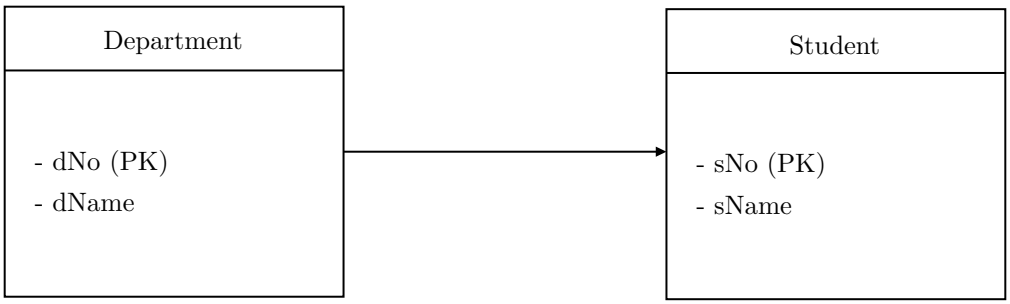


图 7.1

附录 A　集合论

A.1　公理化集合论

接下来我们介绍一个公理系统, 该公理系统描述了被称为集合的数学对象的性质, 在介绍这些公理后, 我们将展示这些公理的一些最简单的推论.

A.1.1　Zermelo-Fraenkel 公理系统

公理 1. (外延公理). $A=B \iff \forall x\left((x\in A) \iff (x\in B)\right)$, 即集合 A 与集合 B 相等, 当且仅当它们所具有的各元素是相同的.

公理 2. (分离公理). 集合 A 和性质 P 总能够确定一个集合 B, 其元素是且仅是 A 中具有性质 P 的元素, 即 $B=\{x\in A|P(x)\}$.

例 A.1.1. 我们可以用分离公理来理解差集运算, 设 A,B 是两个集合, $A\setminus B \xlongequal{\text{def}} \{x\in A|x\notin B\}$, 显然这里的 A 是一个集合, 而 $x\notin B$ 是一个性质, 它们总是能够确定这样的集合 $A\setminus B$.

例 A.1.2. 试证明$\forall A\,(\varnothing\in A)$, 即空集是任何集合的子集.

证明. 分离公理在一些数学结构中是很常用的, 我们需要从一个集合 A 中分离出具有性质 P 的元素来. 显然, 在这个问题中, 设 X 是一个集合, 那么我们能够分离出它的空子集 $\varnothing_X=\{x\in X|x\neq x\}$ 来. 设 Y 是另一个任意给定的集, 有 $\varnothing_Y=\{y\in Y|y\neq y\}$, 根据外延公理, $\varnothing_X=\varnothing_Y$. 由于我们的 X,Y 都是任意给定的集合, 这就是说, 空集是唯一的, 我们用 $\varnothing$ 来表示空集, 并且, 任何集合都有空子集. □

公理 3. (并集公理). 由集合族 $\mathscr{A}$ 的诸集合 A_i 的元素组成的集合 $\cup\mathscr{A}$ 是存在的. 即集合的并集是一个集合, 且

$$x\in\cup\mathscr{A} \iff \exists A\in\mathscr{A}\,(x\in A)\,.$$

例 A.1.3. 根据并集公理和分离公理, 我们可以给出集合族 $\mathscr{A}$ 的交集 $\cap\mathscr{A}$ 的定义, 并集概念的核心是并集中的元素属于集合族中的每一个集合, 因此 $\cap\mathscr{A} \xlongequal{\text{def}} \{x\in\cup\mathscr{A}|\forall A\in\mathscr{A}(x\in A)\}$.

公理 4. (配对公理). 对于集合 X 和集合 Y, 存在一个集合, 其元素是且仅是 X 和 Y, 将该集合记作 $\{X,Y\}$, 称为集合 X,Y 的**无序偶**.

公理 5. (子集之集公理). 对于任意给定的集合 A, 存在一个集合 $\mathscr{P}(A)$, 使得 $\mathscr{P}(A)$ 中的元素是且仅是 A 的子集, 即 $\mathscr{P}(A)=\{X|X\subset A\}$.

例 A.1.4. 现在我们来回答为什么要按照定义 **??** 的方式来定义二元关系. T.B.C.

公理 $1 \sim 5$ 限制了形成新集合的可能性. 为了表述下面的公理, 我们需要给定后继集和归纳集的概念.

定义 A.1.1. (后继集). 设 X 是一个集合, 集合 $X \cup \{X\}$ 称为 X 的**后继集**, 即在 X 中补充一个单元素集合 $\{X\}$. 集合 X 的后继集记作 X^+.

定义 A.1.2. (归纳集). 假设 X 是一个集合, 如果 X 包含空集以及自身任何一个元素的后继集, 则称 X 是一个归纳集.

公理 6. (无穷公理). *归纳集, 即包含空集以及自身任何一个元素的后继集的集合, 是存在的.*

例 A.1.5. 根据无穷公理和公理 $1 \sim 4$, 我们可以建立自然数集 $\mathbb{N}_0$ 的 Von Neumann 方案. 我们把自然数集 $\mathbb{N}_0$ 定义为各归纳集的交集, 即最小归纳集. $\mathbb{N}_0$ 的元素是集合

$$\varnothing, \varnothing^+, \left(\varnothing^+\right)^+,$$

其中, $\varnothing^+ = \varnothing \cup \{\varnothing\} = \{\varnothing\}$, 而 $\{\varnothing\}^+ = \{\varnothing\} \cup \{\{\varnothing\}\}$. 这些集合就是我们用符号 $0, 1, 2, \cdots$ 表示的并称之为自然数的数学对象的模型.

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.

[2] [俄罗斯] B.A.Zorich. 数学分析: 第 7 版. 第一卷 [M]. 北京: 高等教育出版社, 2019

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