# The Relational Model

**关系数据库理论基础总结**

1. **关系模型与关系的定义**
   * 在关系数据库中，数据以\*\*关系（relation）\*\*的形式组织。
   * 一个关系可以用**表格**表示：
     + **行（row/tuple）**：描述一个实体（如患者、人员、预约、员工等）的信息。
     + **列（column/attribute）**：表示该实体的一个属性（如姓名、电话、薪资等）。
   * **关系操作（relational operators）是封闭的**：对一个关系进行操作，结果仍然是一个关系。
2. **模式（Schema）与实例（Instance）**
   * 一组属性（表的列集合）定义了一个**关系模式（schema）**。
   * 实例（instance）就是在某个时间点，具体的行集合（即具体的数据）。
   * 每个属性都属于一个**有限的取值域（domain）**，例如薪资是数字，姓名是字符串。
   * 一个关系是所有可能属性取值的**笛卡尔积的子集**。即从属性值的所有可能组合里，关系只选择其中一部分作为实际存储的元组。
3. **关系的基本特征**
   * **行无序**：元组的排列顺序没有意义。
   * **列无序**：属性顺序没有意义，只要能识别出哪个是哪个属性即可。
   * **属性值原子性**：每个属性值必须是不可再分的基本值，不能是一个集合或另一个元组。
   * **不允许重复元组**：关系是集合，集合中没有重复元素。
4. **键（Keys）**
   * **超键（Superkey）**：能唯一标识元组的一组属性。
     + 如果一个属性或属性组合能唯一标识元组，就称为超键。
     + 如果集合 K 是超键，那么任何包含 K 的超集也是超键。
   * **候选键（Candidate Key）**：最小超键。即去掉其中任何一个属性后就不能再唯一标识元组。
   * **主键（Primary Key）**：从候选键中选定一个作为主要标识。
     + 例如员工表中的员工号（employee number）就是一个天然的主键。
   * **外键（Foreign Key）**：某个关系的主键出现在另一个关系中，这个属性在第二个关系中就叫做外键。
     + 被引用的关系称为**被参照关系（referenced relation）**，外键所在的关系称为**参照关系（referencing relation）**。
     + 例如：
       - 员工表的主键是员工号。
       - 项目-员工对应表中也包含员工号，用来表示某员工参与了哪个项目。
       - 这里员工号在项目-员工表中就是外键。
5. **关系的关联**
   * 外键使得一个表能够**引用另一个表的数据**，从而表达不同实体之间的联系。
   * 例如，项目-员工表通过“员工号”和“项目号”两个外键，表示“哪个员工参与了哪个项目”的多对多关系。

# Relational Algebra

## 关系代数简介（Relational Algebra Intro）

1. **关系代数的本质**
   * 关系代数是 **关系数据库的理论基础**，起源于集合论。
   * 它提供了一种抽象的数学模型，用来推理和操作数据。
   * 数据表示为 **关系（relation）**，即由一组 **元组（tuple）** 组成的集合。
   * 每个元组由固定数量的 **属性（attribute）** 构成。
2. **表格的直观类比**
   * 可以将关系直观地看作一张电子表格：
     + **行（row）** = 元组，表示某个对象的信息（如一个人、一笔交易）。
     + **列（column）** = 属性，表示该对象的某个方面（如姓名、金额、日期）。
   * 整张表 = 一个关系，包含所有元组。
3. **模式（schema）与操作**
   * 每个关系有一个固定的属性集合，称为其 **模式（schema）**。
   * 关系代数定义了一组 **基本操作**，可以将一个关系转换为另一个关系。
   * 基本操作是严谨的数学模型，保证数据更新与查询的正确性。
4. **关系代数的基本操作**
   * **选择（Select）**：挑选行（按条件过滤）。
   * **投影（Project）**：挑选列（保留部分属性）。
   * **并（Union）**
   * **交（Intersection）**
   * **差（Set Difference）**
   * **笛卡尔积（Cartesian Product）**
   * **重命名（Rename）**

➡️ 所有复杂操作（例如多表连接、多条件查询）都可以由这些基本操作组合而成。

1. **意义**
   * 有了这样一个数学基础，数据库系统能够：
     + **验证正确性**
     + **优化性能**
     + **开发新操作**
   * 保证数据库操作的一致性和完整性。

## 关系代数的核心概念（Relational Algebra Concepts）

1. **两个核心要素**
   * **关系（Relation）**：数据以及其结构。
   * **操作（Operations）**：对关系进行操作（查询、变换、添加、删除）。
2. **关系的定义**
   * 数学上：关系 = **元组的集合**。
   * 每个元组都有一组固定的属性。
   * 示例：银行交易表
     + 每行表示一笔交易：交易号、账户号、类型（存款/取款）、金额、日期。
     + 每列是一个属性。
3. **属性的域（Domain）**
   * 每个属性都有一个取值范围（类似编程语言的类型）。
   * 例如：
     + 金额 → 数字（浮点数）
     + 类型 → {“deposit”, “withdrawal”}
     + 日期 → Date 类型
4. **关系的特性**
   * 所有元组必须有相同的属性数和结构。
   * 关系是集合 → **不能有重复元组**。
   * **关系模式（Relation Schema）** 定义了：
     + 属性名
     + 属性类型/域
     + 哪些属性是键（Keys）
5. **键（Keys）在关系模式中的作用**
   * **主键（Primary Key）**：唯一标识元组（如交易号）。
   * **外键（Foreign Key）**：引用其他关系的主键（如账户号）。
   * 外键建立了表与表之间的联系。
6. **总结**
   * 关系代数 = 严格的数学模型。
   * 确保数据库的 **完整性、一致性**。
   * 为后续 **查询优化**、**复杂操作** 打下基础。

📌 **一句话总结**：  
关系代数把数据库看作 **集合论** 下的数学对象，用 **元组集合 + 一组基本操作** 来严谨地描述和操作数据。它不仅是 SQL 等查询语言的理论基础，还保证了数据库操作的正确性与一致性。

# Relational Algebra Joins

**1. 什么是连接**

* **连接（Join）** 是一类非常重要的关系代数操作，用来 **把两个不同关系中相关的元组组合在一起**。
* 它的核心思想是：根据某种条件，把表 A 和表 B 中相关的行合并，生成一个新的关系。

👉 直观例子：

* 表 1：客户表（CustomerID, Name）
* 表 2：订单表（OrderID, CustomerID, Product）
* 通过 CustomerID 连接两个表，就能得到包含客户姓名和订单产品的新表。

**2. 内连接（Inner Join）**

* **定义**：只保留左右两张表中 **匹配条件成立** 的行。
* 如果某一行在一张表中有，但在另一张表没有匹配，则不会出现在结果中。

**种类**：

1. **θ-连接（Theta Join）**
   * 条件可以是任意比较运算符（=, >, <, ≥, ≤, ≠）。
   * 例如：员工工资 > 部门预算的 1/10。
2. **等值连接（Equi-Join）**
   * 条件仅限于相等（=）。
   * 最常见：员工表的 DepartmentID = 部门表的 DepartmentID。
3. **自然连接（Natural Join）**
   * 特殊的等值连接。
   * 自动识别 **同名属性** 并要求它们相等。
   * 例如：员工表和部门表都含有 DepartmentID，做自然连接时会自动基于它来匹配。

**3. 外连接（Outer Join）**

* 内连接只保留匹配的行，而外连接则 **保留不匹配的行，用空值（NULL）填补缺失部分**。
* 分为三类：

1. **左外连接（Left Outer Join）**
   * 保留左表所有元组，即使在右表中没有匹配。
   * 不匹配时，右表的字段用 NULL 填充。
   * 例子：列出所有员工及其部门，即使有员工没有部门。
2. **右外连接（Right Outer Join）**
   * 保留右表所有元组，即使在左表中没有匹配。
   * 不匹配时，左表字段用 NULL 填充。
   * 例子：列出所有部门，即使有部门没有员工。
3. **全外连接（Full Outer Join）**
   * 左右两边都保留。
   * 结果中包含左表和右表所有元组，缺失部分填 NULL。
   * 例子：包含所有员工和所有部门，即使某些员工没有部门或某些部门没有员工。

**4. 为什么需要连接**

* 连接允许我们把一个“大而复杂”的表拆分成多个小表，从而：
  + **减少冗余**（避免信息重复出现）。
  + **避免不一致**（如部门预算在一个行更新了而另一行没更新）。
  + **结构更清晰**（员工信息和部门信息分表存储）。
  + **存储更高效**。

👉 然后我们只需要在查询时使用连接，就能重新组合出所需的信息。  
这就是为什么 **关系型数据库如此强大**：它允许 **规范化存储**（拆分）+ **灵活查询**（连接）。