数据库

创建数据库的三范式

> 范式：范式从本质上讲是一种理论体系、理论框架。在该体系框架之内的该范式的理论、法则、定律都被人们普遍接受。
>
> 通俗的理解就是规则，规定

一、数据库三范式

1nf:第一范式主要是保证数据表中的每一个字段的值必须具有原子性，也就是数据表中的每个字段的值是不可再拆分的最小数据单元

> 简单来说就是：拆列

案例：

```
杨柳数    伤亡人数    课程名（语文，数学）
```

在上述字段中杨柳数可以被拆分为杨树和柳树，伤亡人数可以拆分为受伤人数和死亡人数，课程名有语文数学英语等课程。

非原子性的字段还有哪些？

2nf：在满足第一范式的基础上，要求表中的所有非主键列都完全依赖于主键，不存在部分依赖。

> 简单来说就是： 建主键

表要有个主键作为唯一标识，其它字段跟主键还要有关系

案例: 学生表

```
学生编号    学生姓名    班主任姓名    班主任工资
```

首先我们找到主键学生编号作为主键

但是表内的班主任相关字段跟学生有关系吗？有。班主任姓名跟学生关系较大，但是班主任工资跟学生的关系好像并不是很大。

所以这里我们的班主任工资字段就应该在学生表中删去。

```
学生编号    学生姓名    班主任姓名
```

> 补充：
>
> 哪些属性可以做主键
>
> 主键一定是int吗

注意：主键一般不参与业务，为什么？

3nf： 在满足第二范式的基础上，要求表中的所有非主键列和主键之间不存在传递依赖。

> 简单来说就是：拆表

案例：

```
学生编号    学生姓名  班主任编号  班主任姓名
```

我们这里有张学生表，根据上面的第二范式我们找到主键。

我们以学生编号作为主键，班主任编号，班主任姓名和学生有关系吗？有的，但是有直接关系吗？没有

所以，班主任的两个字段与学生表主键学生编号存在传递依赖。我们就需要消除传递依赖，就是**拆表**。

```
学生表：学生编号    学生姓名
班主任表：班主任编号  班主任姓名
```

> 我们这里拆表完成后，只看表找不到学生和班主任的对应关系。我们应该怎么做才能对应起来？

二、4nf 5nf

这些内容需要自己去查，去理解。

三、有些时候会违反第三范式

实际业务中有时会违反第三范式，比如我们在上面第三范式中

```
学生编号    学生姓名  班主任编号  班主任姓名
```

学生表有可能就是这样，并不进行拆表。这就涉及到了我们业务中的情况

联表查询消耗性能。

你们可以考虑一下之前的12306，除去验证买车票要用多长时间。这就是性能问题。

表关系

1、一对一

案例：

1. 用户和用户详情：用户表中保存用户的基础信息，如用户名、密码，而用户详情表保存用户的详细信息，如地址、电话、性别等。

   作业

2. 员工和工资卡：员工表记录每个员工的信息，工资卡表记录每个员工对应的工资卡信息。

    1. 员工表（**Employees**）：这个表中存储每个员工的基本信息。具体的字段可能包括：
        - EmployeeID：员工的唯一标识符，通常是一个自增的整数。这是表的主键。
        - FirstName：员工的名字。
        - LastName：员工的姓氏。
        - Email：员工的电子邮箱地址。
        - Phone：员工的电话号码。
        - HireDate：员工的入职日期。
        - Position：员工的职位，如“工程师”、“销售”、“经理”等。
        - Department：员工所属的部门。

   这些字段都是针对每个员工的基本信息。但是，如果我们还想存储每个员工的工资卡信息，那么可以创建另一个表。

    1. 用户和用户详情：
        - 用户表（**Users**）：主要字段可能包括UserID（主键）、UserName、Password等。
        - 用户详情表（**UserDetails**）：主要字段可能包括UserDetailID（主键）、UserID（外键，与用户表的UserID字段关联）、Address、Phone、Gender等。
    2. 学生和学生档案：
        - 学生表（**Students**）：主要字段可能包括StudentID（主键）、Name、Class、Age等。
        - 学生档案表（**StudentRecords**）：主要字段可能包括RecordID（主键）、StudentID（外键，与学生表的StudentID字段关联）、Grades、Awards、DisciplinaryActions等。
    3. 产品和产品详情：
        - 产品表（**Products**）：主要字段可能包括ProductID（主键）、ProductName、Price等。
        - 产品详情表（**ProductDetails**）：主要字段可能包括ProductDetailID（主键）、ProductID（外键，与产品表的ProductID字段关联）、

Manufacturer、Specification、Description等。

4. 医生和排班信息：
    - 医生表（**Doctors**）：主要字段可能包括DoctorID（主键）、Name、Specialty等。
    - 排班信息表（**Schedules**）：主要字段可能包括ScheduleID（主键）、DoctorID（外键，与医生表的DoctorID字段关联）、WorkDays、WorkHours等。
5. 作者和作者简介：
    - 作者表（**Authors**）：主要字段可能包括AuthorID（主键）、Name、Nationality等。
    - 作者简介表（**AuthorBios**）：主要字段可能包括BioID（主键）、AuthorID（外键，与作者表的AuthorID字段关联）、Biography、Awards、PublishedWorks等。

一对多

案例：

- 部门和员工

对于部门来说，一个部门可以有多个员工，这就是一对多。

用人可能就要问了，一个部门也可能只有一个员工啊？这不是一对一吗？

我们这里的一对多只考虑普适情况，特殊情况单独考虑。或者你可以这样理解，这个部门有没有可能再招新员工。

或者也有人有这样的疑惑：一个员工只能有一个部门，那他们的关系不是一对一吗？

首先，员工只能有一个部门，这个员工 和部门的关系是一对一，但是员工和部门是多对一。

我们在看表关系的时候是对整个表中的数据分析，而不是单条数据。

作业：

- 班主任学生
- 用户角色

多对多

案例：

- **学生和课程**：一个学生可以选修多门课程，一门课程也可以被多个学生选修。

我们在设计多对多关系的时候，通常需要通过引入一个第三个表（也叫做关联表或者桥接表）来实现。这个关联表至少包含两个字段，分别引用两个表的主键。这样，每个记录在一个表中都有多个与之对应的记录在另一个表中，反之亦然。

这样就巧妙的把多对多转化成了多个一对多。

作业：

多对多（Many-to-Many）的表关系指的是两个表之间的数据记录存在多对多的对应关系。以下是10个具有多对多表关系的业务场景：

1. **医生和患者**：一个医生可以治疗多个患者，一个患者也可以被多个医生治疗。
2. **作者和图书**：一个作者可以写多本图书，一本图书也可以由多个作者合写。
3. **用户和角色**：在许多系统中，一个用户可以拥有多个角色，一个角色也可以被多个用户拥有。
4. **商品和订单**：一个订单可以包含多种商品，一个商品也可以被包含在多个订单中。
5. **学生和社团**：一个学生可以参加多个社团，一个社团也可以有多个学生成员。
6. **演员和电影**：一个演员可以参演多部电影，一部电影也会有多个演员参演。
7. **音乐家和乐器**：一个音乐家可能会演奏多种乐器，一个乐器也可以被多个音乐家演奏。
8. **供应商和零售商**：一个供应商可以向多个零售商供货，一个零售商也可以从多个供应商那里购货。
9. **教师和课程**：一个大学老师可能会教授多门课程，一门课程也可能会由多位教师共同教授。

> 补充：
>
> 多对多怎么维护：建中间表，把多对多转换成一对多或者一对一
>
> 一对多在哪方维护: 维护就是建立关系，一般指建立外键。在多的一方法维护
>
> 一对一是特殊的多对一

表关系在java中怎么映射

1、一对一

用户和用户详情：

```
public class User {
    int userID;
    String userName;
    String password;
    UserDetails userDetails; // 这里在user中引入了userDetails
}
```

或者

```
public class UserDetails {
    int userDetailID;
    int userID;
    String address;
    String phone;
    String gender;
    //User user; // 在userDetails中存入user信息
}
```

当我们要把数据保存到数据库中，或者从数据库中查询数据的时候，很多同学不知道user数据怎么保存。

只保存用户的时候，可以只存User的userId,userName,password就可以了。

只保存UserDetails的时候，可以只保存UserDetails的数据，不存userID或者user。

当我要更新UserDetails时，可以把userID存到user里面也可以直接存userID。

查询的时候需要查询关联把用户详情存储到UserDetails里面。

user对象和userId只保留一个就行。

2、一对多：

部门和员工

```
public class Dept {
    int deptno;
    String dname;
    String loc;
    List<Emp> emp;
```

```
}
```

或者

```
public class Emp {

    int empno;
    String ename;
    String job;
    int mgr;
    BigDecimal sal;
    BigDecimal comm;
    Date hirdate;
    //int deptno;
    //注意这个地方，我们在Emp里面存一个Dept并不是说明一对一。
    // 而是说明我这Emp只有一个部门，所以一个Emp只需要存储一个Dept。
    //看表关系主要是看第一个表里面几条记录对应另一个表里面几条记录。
    //所以一个员工只对一个部门不能说明员工对部门是一对一的。
    //Dept dept;
}
```

对应增删改查的案例是根据部门进行分页查询