数据库设计

# 概念：

关系型数据库建议在E-R模型的基础上，我们需要根据产品经理的设计策划，抽取出来模型与关系，制定出表结构，这是项目开始的第一步

在开发中有很多设计数据库的软件，常用的如power designer，db desinger等，这些软件可以直观的看到实体及实体间的关系

设计数据库，可能是由专门的数据库设计人员完成，也可能是由开发组成员完成，一般是项目经理带领组员来完成

我们初学者现阶段不需要独立完成数据库设计，但是要注意积累一些这方面的经验

# 三范式

经过研究和对使用中问题的总结，对于设计数据库提出了一些规范，这些规范被称为范式(Normal Form)

目前有迹可寻的共有8种范式，一般需要遵守3范式即可

◆ 第一范式（1NF）：强调的是列的原子性，即列不能够再分成其他几列。

考虑这样一个表：【联系人】（姓名，性别，电话） 如果在实际场景中，一个联系人有家庭电话和公司电话，那么这种表结构设计就没有达到 1NF。要符合 1NF 我们只需把列（电话）拆分，即：【联系人】（姓名，性别，家庭电话，公司电话）。1NF 很好辨别，但是 2NF 和 3NF 就容易搞混淆。

 第二范式（2NF）：首先是 1NF，另外包含两部分内容，一是表必须有一个主键；二是没有包含在主键中的列必须完全依赖于主键，而不能只依赖于主键的一部分。

考虑一个订单明细表：【OrderDetail】（OrderID，ProductID，UnitPrice，Discount，Quantity，ProductName）。

因为我们知道在一个订单中可以订购多种产品，所以单单一个 OrderID 是不足以成为主键的，主键应该是（OrderID，ProductID）。

显而易见 Discount（折扣），Quantity（数量）完全依赖（取决）于主键（OderID，ProductID），

而 UnitPrice，ProductName 只依赖于 ProductID。

所以 OrderDetail 表不符合 2NF。不符合 2NF 的设计容易产生冗余数据。

可以把【OrderDetail】表拆分为【OrderDetail】（OrderID，ProductID，Discount，Quantity）和【Product】（ProductID，UnitPrice，ProductName）来消除原订单表中UnitPrice，ProductName多次重复的情况。

第三范式（3NF）：首先是 2NF，另外非主键列必须直接依赖于主键，不能存在传递依赖。即不能存在：非主键列 A 依赖于非主键列 B，非主键列 B 依赖于主键的情况。

考虑一个订单表【Order】（OrderID，OrderDate，CustomerID，CustomerName，CustomerAddr，CustomerCity）

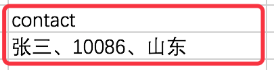
主键是（OrderID）。 其中 OrderDate，CustomerID，CustomerName，CustomerAddr，CustomerCity 等非主键列都完全依赖于主键（OrderID），所以符合 2NF。

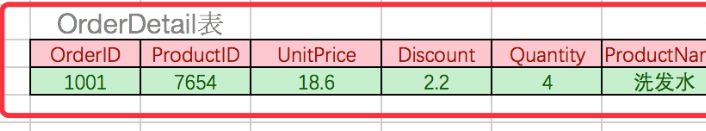
不过问题是 CustomerName，CustomerAddr，CustomerCity 直接依赖的是 CustomerID（非主键列），而不是直接依赖于主键，它是通过传递才依赖于主键，所以不符合 3NF。

通过拆分【Order】为【Order】（OrderID，OrderDate，CustomerID）和【Customer】（CustomerID，CustomerName，CustomerAddr，CustomerCity）从而达到 3NF。

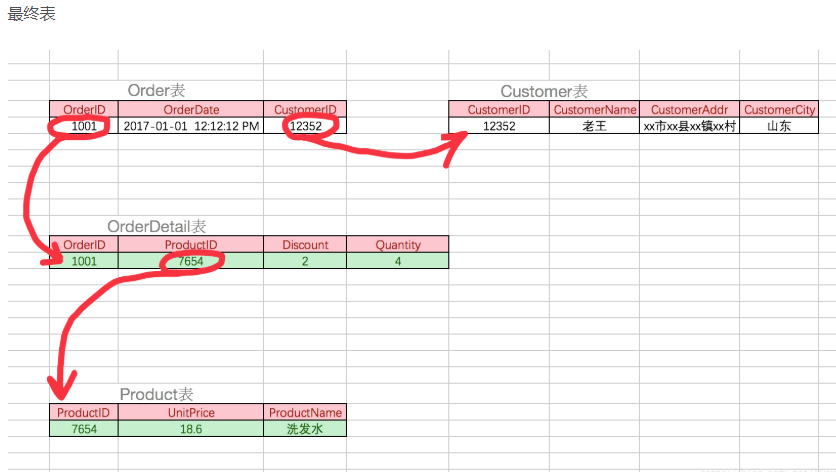
\*第二范式（2NF）和第三范式（3NF）的概念很容易混淆，区分它们的关键点在于，2NF：非主键列是否完全依赖于主键，还是依赖于主键的一部分；3NF：非主键列是直接依赖于主键，还是直接依赖于非主键列。

# 测试：









数据库设计一：

设计一套学生公选课定课系统，其中内容包含课程编辑、讲师资料编辑、学生资料编辑、学生选课、考试成绩登记、成绩公示等模块。其他要求，要求进行权限最小化划分。

在此，以学生资料编辑这个子模块进行设计。