# 业务设计

## 逻辑设计

### 范式设计

#### 数据库设计的第一大范式

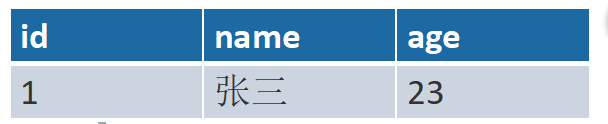
数据库表中的所有字段都只具有单一属性

单一属性的列是由基本数据类型所构成的

设计出来的表都是简单的二维表



name-age列具有两个属性，一个name,一个 age不符合第一范式，把它拆分成两列

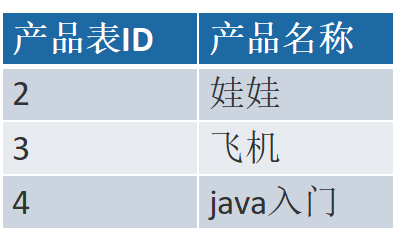


#### 数据库设计的第二大范式

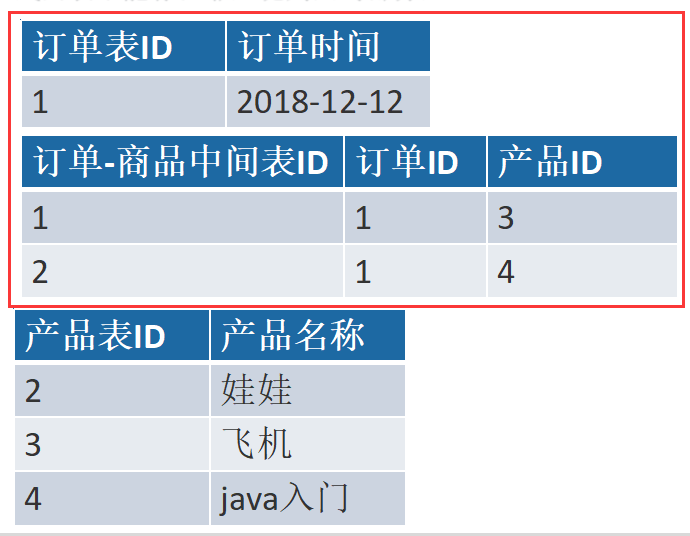
要求表中只具有一个业务主键，也就是说符合第二范式的表不能存在非主键列只对部分主键的依赖关系

有两张表：订单表，产品表





一个订单有多个产品，所以订单的主键为【订单ID】和【产品ID】组成的联合主键，这样2个组件不符合第二范式，而且产品ID和订单ID没有强关联，故，把订单表进行拆分为订单表与订单与商品的中间表



#### 数据库设计的第三大范式

指每一个非非主属性既不部分依赖于也不传递依赖于业务主键，也就是在第二范式的基础上相处了非主键对主键的传递依赖



其中

**客户编号 和订单编号管理 关联**

**客户姓名 和订单编号管理 关联**

**客户编号 和 客户姓名 关联**

如果客户编号发生改变，用户姓名也会改变，这样不符合第三大范式，应该**把客户姓名这一列删除**

#### 范式设计实战

按要求设计一个电子商务网站的数据库结构

1. 本网站只销售图书类产品
2. 需要具备以下功能

用户登陆 商品展示 供应商管理

用户管理 商品管理 订单销售

用户登陆及用户管理



**只有一个业务主键，一定是符合第二范式**

**没有属性和业务主键存在传递依赖的关系，符合第三范式**

商品信息



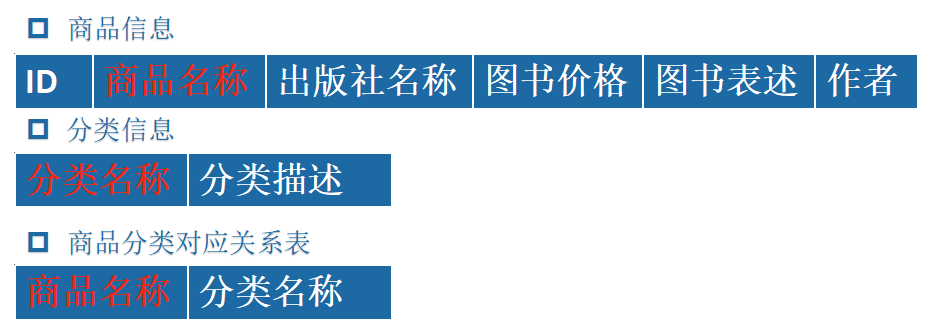
一个商品可以属于多个分类，故，商品名称和分类应该是组合主键，会有大量冗余，不符合第二范式。应该把分类信息单独存放



另外再建立一个中间表把分类信息和商品信息进行关联



最后的三张表如下



供应商管理功能



符合三大范式，不需要修改，但假如增加新的一列【银行支行】，这样随着银行账户的变化，银行支行也会编号，不符合第三大范式



在线销售功能



**有多个业务主键，不符合第二范式**

**订单商品单价。订单数量，订单金额 存在传递依赖关系，不符合第三范式**

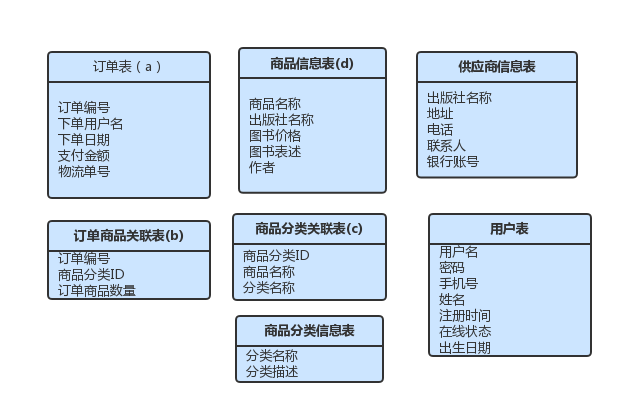
拆分的结果如下



这时候，【订单商品分类】与【订单商品名】有依赖关联，故合并如下

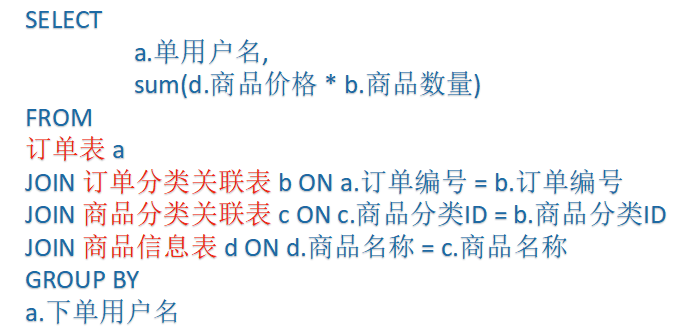


表汇总

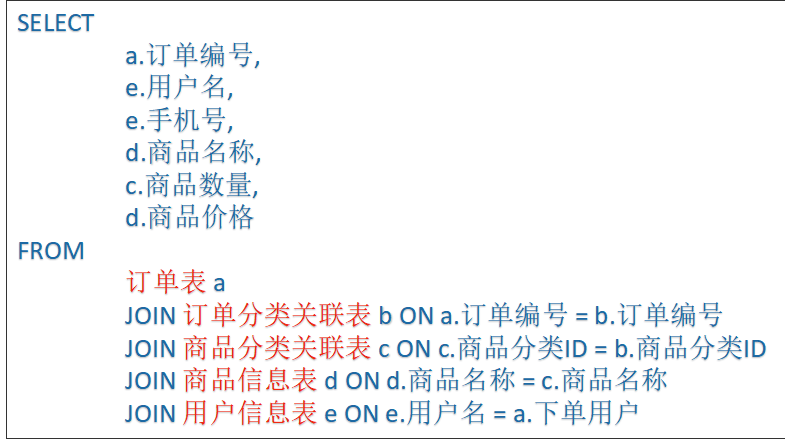


查询练习

编写SQL查询出每一个用户的订单总金额（用户名，订单总金额）



编写SQL查询出下单用户和订单详情（订单编号，用户名，手机号，商品名称，商品数量，商品价格）



问题：

大量的表关联非常影响查询的性能

完全符合范式化的设计有时并不能得到良好得SQL查询性能

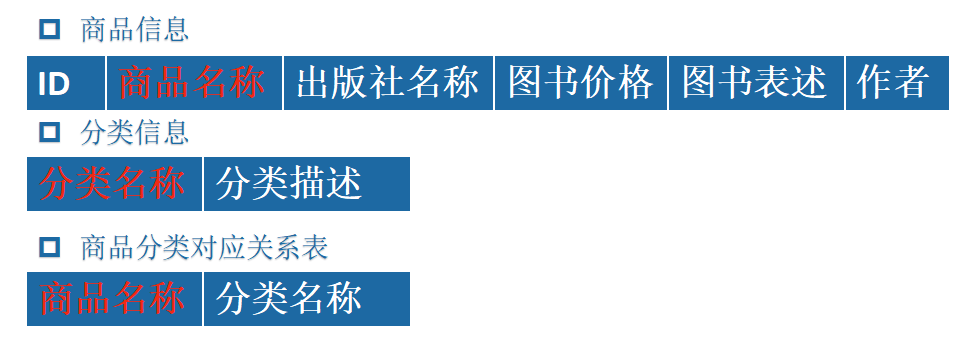
### 反范式设计

#### 什么叫反范式化设计

* 反范式化是针对范式化而言得，在前面介绍了数据库设计得范式
* 所谓得反范式化就是为了性能和读取效率得考虑而适当得对数据库设计范式得要求进行违反
* 允许存在少量得冗余，换句话来说反范式化就是使用空间来换取时间

商品信息反范式设计

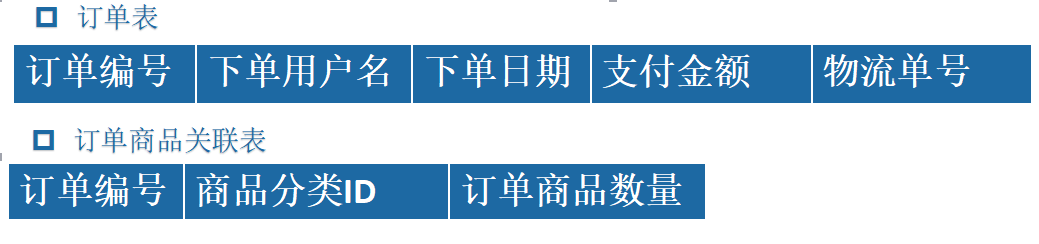
下面是范式设计的商品信息表



商品信息和分类信息经常一起查询，所以把分类信息也放到商品表里面，冗余存放

在线销售功能反范式

下面是在线手写功能的范式设计



**首先来看订单表**

1. 查询订单信息要关联查询到用户表，但用户表的电话是可能改变的，而且查询订单的时候经常查询到用户的电话
2. 查询订单经常会查询到订单金额，所以把订单金额也冗余进来

新设计的订单表如下



**再来看订单关联表**

1. 和商品信息反范式设计一样，查询订单的时候经常查询商品分类，所以把商品分类和订单名冗余进来

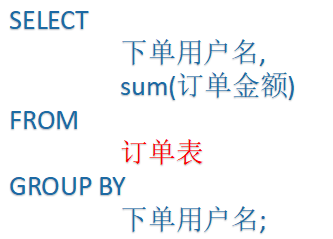
2.商品的单价可能会编号，如果关联查询查询只能查询到最新的商品价格，而查询不到下订单时候的价格，并且商品单价经常会查询。 所以把订单单价也冗余进来

新设计的商品关联表如下

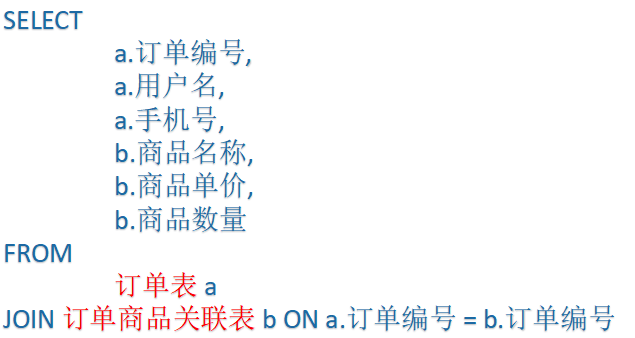


查询练习

**编写SQL查询出每一个用户的订单总金额**



**编写SQL查询出下单用户和订单详情**



### 总结

不能完全按照范式得要求进行设计

考虑以后如何使用表

#### 范式化设计优缺点

优点：

可以尽量得减少数据冗余

范式化的更新操作比反范式化更快

范式化的表通常比反范式化的表更小

缺点：

对于查询需要对多个表进行关联

更难进行索引优化

#### 反范式化设计优缺点

优点：

可以减少表的关联

可以更好的进行索引优化

缺点：

存在数据冗余及数据维护异常

对数据的修改需要更多的成本

## 物理设计

### 命名规范

数据库、表、字段的命名要遵守可读性原则

使用大小写来格式化的库对象名字以获得良好的可读性

例如：使用custAddress而不是custaddress来提高可读性。

数据库、表、字段的命名要遵守表意性原则

对象的名字应该能够描述它所表示的对象

例如：

对于表，表的名称应该能够体现表中存储的数据内容；对于存储过程

存储过程应该能够体现存储过程的功能。

数据库、表、字段的命名要遵守长名原则

尽可能少使用或者不使用缩写

### 存储引擎选择



### 数据类型选择

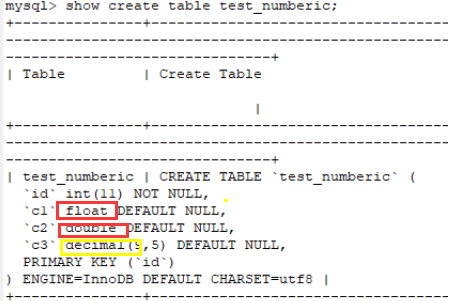
当一个列可以选择多种数据类型时

* 优先考虑数字类型
* 其次是日期、时间类型
* 最后是字符类型
* 对于相同级别的数据类型，应该优先选择占用空间小的数据类型

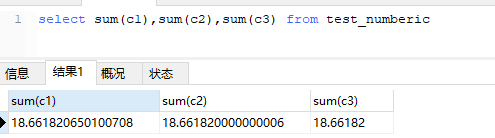
浮点类型



注意float 和double 是非精度类型，如果是和金额相关尽量用decimal



select sum(c1),sum(c2),sum(c3) from test\_numberic



日期类型

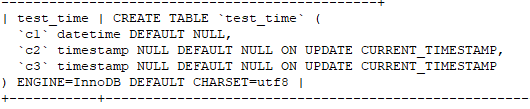
面试经常问道 timestamp 类型 与 datetime区别



datetime类型在5.6中字段长度是5个字节

datetime类型在5.5中字段长度是8个字节

**timestamp 和时区有关，而datetime无关**



insert into test\_time VALUES(NOW(),NOW(),NOW());

set time\_zone="-10:00"

