关系数据库设计

Posted on 2021-11-14
Words count in article: 4.2k | Reading time ≈ 14

关系数据库设计

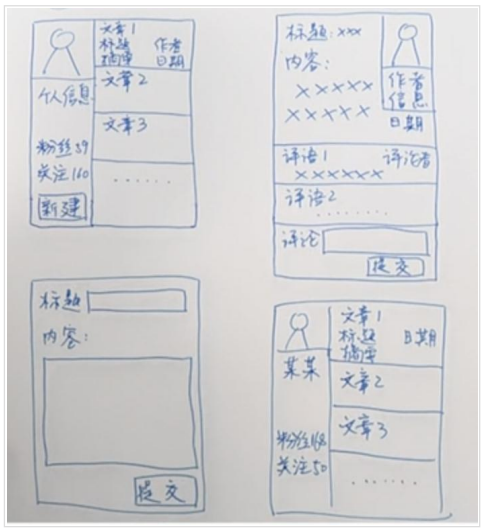
关系数据库设计

怎么做好数据库的设计?

1. 需求分析, 解决存什么
2. 概念设计，用何种模式
3. 数据库结构设计，解决怎么存

需求分析

下图是一个博客系统的4个界面



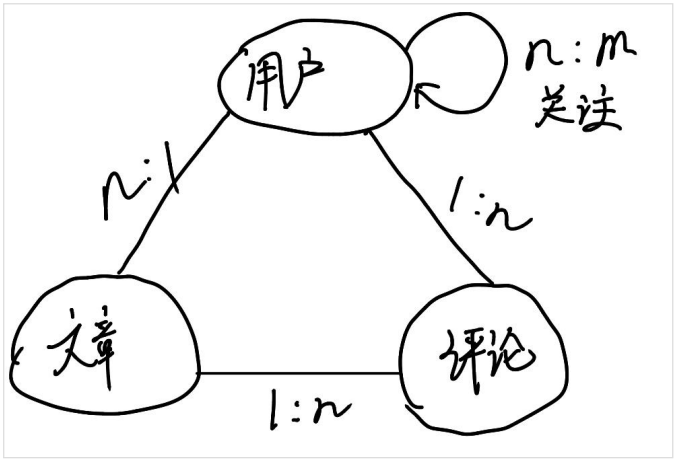
这个界面的功能如上图所示:

- 第一个登录界面，用户能看见自己的个人信息以及关注的人写的文章
- 第二个界面，是用户点击特定一篇文章后显示的文章具体信息
- 第三个界面是用户点击新建按钮之后显式的，用于新创建一个界面
- 第四个界面是其他博主的主页，我们可以看到他写了那些文章，它关注的人之类的信息。

概念设计

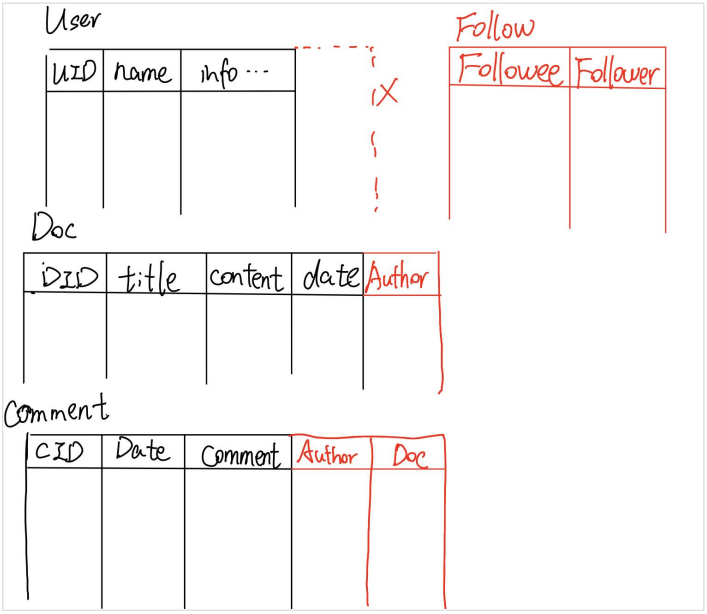
现在我们解决了需求分析，现在我们来做概念设计

对于这个 博客系统，我们需要创建三个对象：用户、文章、评论，其关系如下:



结构设计

我们要用关系来表示这三个对象，用表格来呈现，每一列代表对象的一个属性，如下图所示：



首先，黑色部分代表的是用表来展示对象的基本属性的部分。比如对于用户，有名字、信息等；比如对文档，有标题和内容等

但是怎么用表格来表示对象与对象之间的关系呢？

首先，对于文章和作者、评论和作者这类多对1关系，可以直接在后面加上一列。比如说，对于Doc表格，我们可以在后面加上一列Author，来记录UID，这样就能记录这篇文章的作者了

对于评论，我们可以在表后添加两列，分别记录代表作者的UID和代表文章的DID。用来表示这条评论所在的文章及其作者

但是对于用户和用户之间的关注关系，因为是多对多的，我们发现没有办法通过添加一列的方式来呈现，因此我们可以另外新建一张表，里面有Followee和Follower两列，分别记录被关注者和关注者的UID即可。

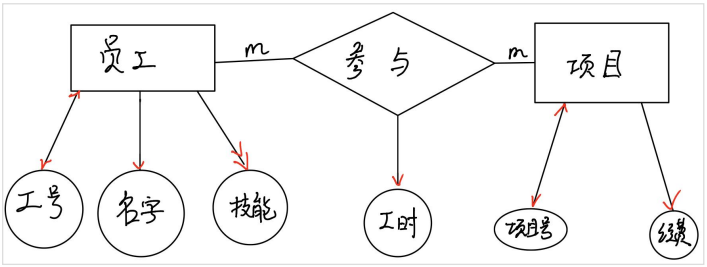
这是我们比较朴素的设计思路，接下来我们介绍一种更系统的设计方法——ER图

ER图

ER图的表现形式有点类似于前面面向对象的模型的概念设计。但是在表达对象与对象之间的关系有一些更细致的方式。

ER图中的E 代表 Entity(实体)，R代表Relationship(联系)，ER图也就是用来刻画实体与实体之间关系的示意图

比如说员工参与项目这个模型我们可以画出其ER图



形状的意义

其中，方块代表实体；圆圈代表实体中的属性；而菱型则代表连接对象之间的关系。

我们从这张ER图中可以很清楚的看到：员工实体有**工号、名字、技能**这三个属性；项目有**项目号、经费**；此外，参与这个关系也有一个**工时**属性，代表每个人需要花多少时间在这个项目上。

箭头的意义

现在，我们要了解不同类型的属性

- **唯一属性**，就是可以唯一识别一个实体的属性。比如员工里的工号，项目中的项目号。通常我们可以把唯一属性看做是这个实体中的ID。
- **单值属性**，名字和经费这类属性可以被称为单值属性，因为一个员工只能对应一个名字，一个项目只能对应一笔经费
- **多值属性**，员工的技能并不是单值属性，一个员工可以由多个技能。

不同的属性反映在不同的箭头上，唯一属性用双向箭头表示，单值属性用单箭头表示，双值属性用两个同一朝向的箭头表示

菱形两变m和1的意义

在上面这个ER图中，我们看到员工和项目之间的参与关系是一个多对多的关系，一个员工可以参加多个项目，同样一个项目也需要多个员工共同完成.因此参与关系的两变是m和m

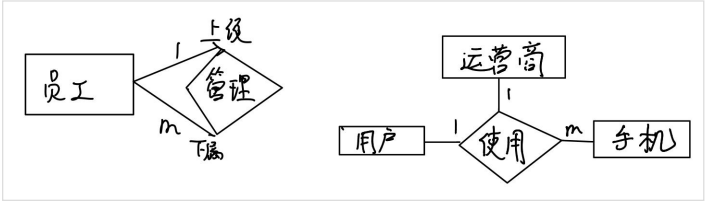
还有两种关系：一对多/多对一，一对一

比如员工和部门之间就是多对一的关系，一个员工只能属于一个部门，而部门可以有多个员工

比如经理和部门之间的就是一对一关系，一个员工只能管理一个部门

不同关系的表现形式

上面我们所介绍的都是两个实体之间的二元联系，其实联系可以有更多种不同的方式，比如：



比如说，同样是员工，上级对下级是一个一对多的关系。我们可以用左边这张图来表示

对于用户、运营商、手机这样一个三边关系，很难说是一对多还是一对一的关系，我们直接用语言描述即可：一个用户可以有多台手机，一个手机只能属于一位用户，而且一个手机只能对应一个运营商

延伸

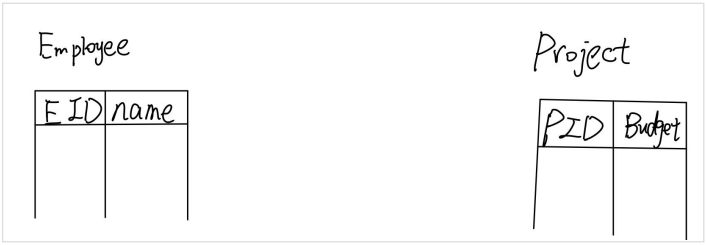
ER图还有其他更细枝末节的刻画，比如说商品和食品、电子产品之间其实是父类和子类的关系。

还有弱实体的概念，比如说一个房间的存在依附于一座楼房的存在，一座城市依附于一个国家的存在

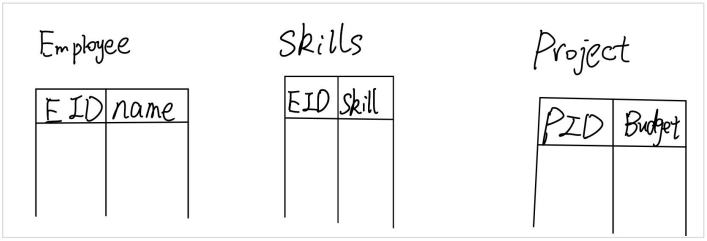
从ER图到关系模型设计

我们之前设计出来了ER图，现在我们要介绍一种固定的方式来将设计出来的ER图转换成一张或者几张关系表。我们还是以上面员工和项目的ER图为例

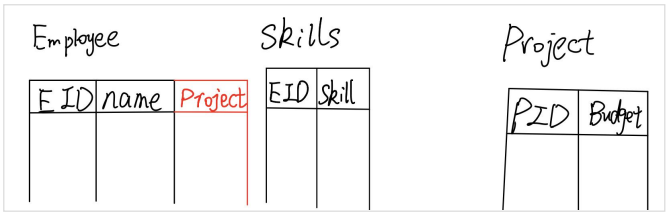
- 首先要将实体类中的唯一属性和单值属性选出来创建一张表



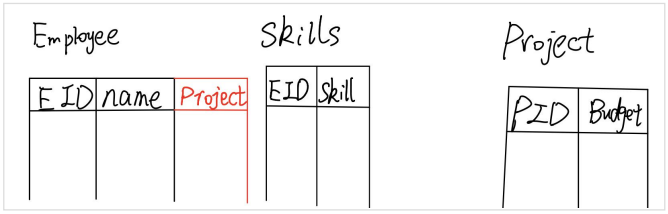
- 对于多值属性，要单独拿出来刻画，比如说对于技能，要新建一张skills表来存放。



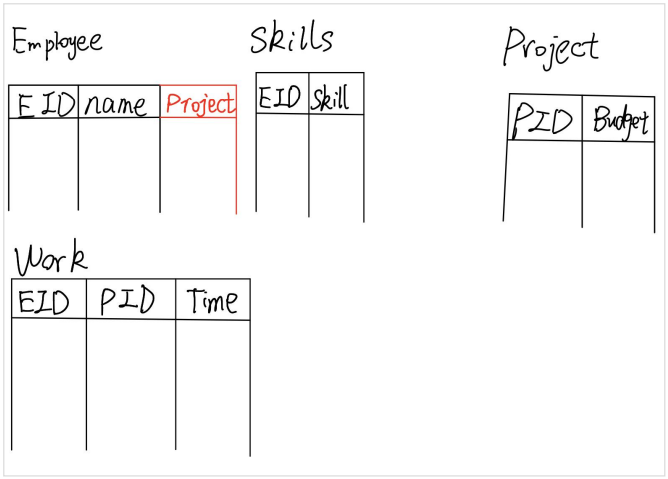
- 最后我们来处理联系，对于一对一或者一对多联系，我们可以将其归并到一个实体当中。但是对于多对多联系，我们需要新建一张表来存放这个联系
  - 我们假设员工和项目是一对一的联系,我们可以任意选择一个实体关系表，在后面加上一列。比如：



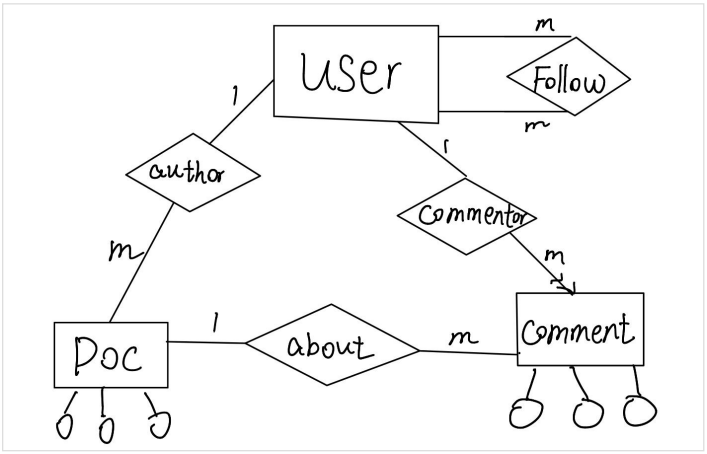
  - 我们假设员工和项目是多对一的联系，我们在多的那一个实体关系表后加一列，即可，这里还是选择加在员工表后



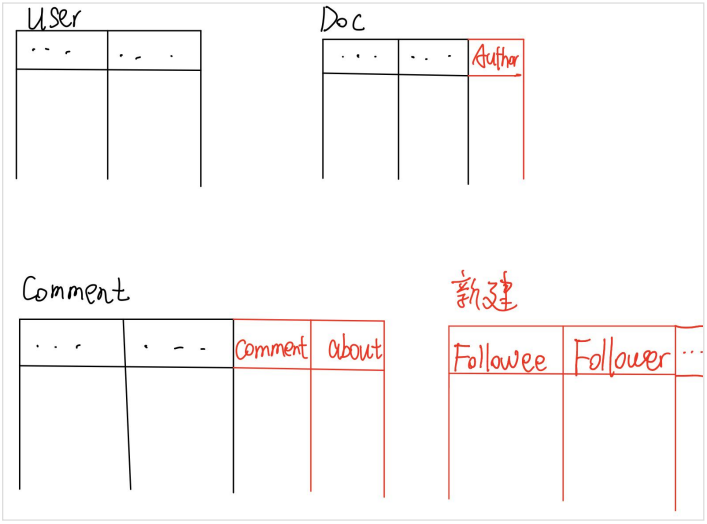
  - 事实上，员工和项目是多对多的联系，因此我们要新建第四张表，这张表有三列，前两列分别存放员工的ID和项目的ID，第三列存放工时



ER图在博客系统中的应用



我们这里忽略了每一个实体的内部属性，把目光聚焦到关系上. 首先在这个类似于三角形的关系中，一共有三个一对多关系，因此在Doc表后加一列，在Comment表后加一列。此外，我们还要处理”关注”这个关系，新建一张表即可。



我们看到，从ER图到关系表，是一个机械化的过程。只要按部就班来即可

例题

一般联系要么可以附着在一张现有的表上(1 to1,1 to m)，如果单提取出来新建一张表，说明这是m to m的关系，那么这一行是由关系两边的实体的ID共同确定的，即A

关系数据库中的冗余

我们在数据库中随处可见冗余，如下表所示：

这里第二行和第三行出现了两个May，然后Price，P_name 也出现了重复的情况

UID	U_name	PID	P_name	Price	Quant	Date
U001	Jason	P003	Pencil	10	1	3/10/2020
U002	May	P002	Soap	5	2	4/10/2020
U003	May	P003	Pencil	10	2	8/10/2020
U004	Bob	P002	Soap	5	1	10/10/2020

这是因为这张表是由 UID和PID确定的，但是U_name是单独由UID确定的，P_name和Price则是有PID单独确定的。在这张表里主键是UID+PID，因此在一张表里面当UID或PID重复出现多次的话，会造成数据的冗余。

冗余的副作用很多，最直观也最易理解的就是当我们修改一个值的时候需要将整个表扫描一遍然后修改所有冗余值。如何规避掉这些冗余，需要在我们设计数据库的时候下功夫。

现在我介绍一下**函数依赖**的含义：

比如u_name 是依赖于UID的，p_name,price是依赖于PID的。在这张表中有这两个函数依赖

```
1  UID -> u_name
2  PID -> p_name,price
```

当函数依赖的两个决定属性(UID和PID)不等于这个表的主键的时候，就会出现冗余的情况

数据库设计的规范化

为了规避冗余，我们需要把我们的数据库设计的规范化，只要我们按照ER图的设计理念，构造出来的数据表一般都是规范的。就像上面这张表，以规范化的方式我们要将其拆分为三张小表，如下：

User

UID	U_name
U001	Jason
U002	May
U003	Bob

Product

PID	Pname	Price
P002	soap	5
P003	pencil	10

Order

UID	PID	Quant	Date
U001	P003	1	3/10/2020
U002	P002	2	4/10/2020
U002	P003	2	8/10/2020
U003	P002	1	10/10/2020

我们要记住宽表拆分的原则：

让同时被使用到的属性(即出现在同一个SQL中的属性)尽可能位于拆分后的一张表中。这样子就不用再做连接，增加开销了

冗余带来的好处

上面我们介绍了两种模式(大表和拆分后的三张表)现在我们来讨论哪一种模式更好。

比如说对于查询

```
SELECT u_name,p_name
FROM XXX
WHERE date = ? AND price*quant >1000
```

虽然大表有冗余，但是能很高效的来处理这段查询。反观拆分的三张表格，我们需要对其做连接后才能完成查询，代价比较大。

那么在规范化和非规范化之间应该怎么选择呢？我们要看看那些冗余的属性被修改的几率是否很高。

如果修改的几率很大，那么对于大表来说每次修改的开销就很大，就不划算；反之则可以选择大表来存储数据

但是对于这张表来说，我们在商城注册了之后，一般就不回去更改名字；且商品一旦创建之后，也不太会修改它的名字。价格这列见仁见智了。因此总体来说对于这套数据库系统，非规范化导致的数据冗余带来的负面作用是比较小的，但正面效果是比较好的。

所以说，非规范化和规范化都有其使用场景。

如何利用冗余

现在我们有两张表：`Product(PID,P_name,Price)` 和 `Order(BID,UID,PID,Quant,Date)`

然后我们用这个查询

```
SELECT Sum(Quant)*Price
FROM Product,Order
WHERE Product.PID = Order.PID
AND Product.PID = ?
```

这个查询非常普遍，就是要我们去计算某一件特定商品的总销售额。如果我们采用标准化的关系表，那么会非常慢，因为Product和Order会连接成一张大表，我们需要扫描这张表中所有的内容，然后再统计计算才能得到结果。因此这时候我们可以用冗余来帮助我们简化查询，方法也很简单就是给Product加一个TotalSales属性

`Product(PID,P_name,Price,TotalSales)`

这样，我们就可以直接获得TotalSales了，而我们所需要付出的代价，仅仅是每次下订单的时候额外更新一下Product表格即可

最后我们对非规范化和冗余做一个延伸和总结。冗余经常会给我们的程序开发带来一些好处。最具有代表性的就是缓存机制。比如web server会专门开一片区域当缓存来存放数据库中经常被调用的那些值。 但是缓存是冗余的值，没了缓存还是可以正常运行，但是有缓存会大大提升程序的运行速度。

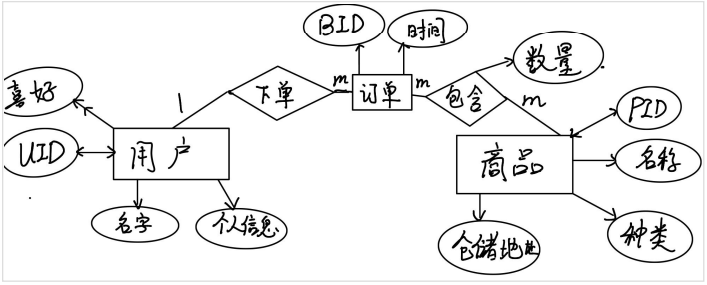
还有比如数据库内部的索引，这也是冗余的一种表现。

网上购物场景实例

- 网站陈列了各种各样的商品，提供商品的详细信息，包括商品名称、种类、价格、仓储地址等。
- 用户可以登录到网站，浏览并选择商品。
- 用户提交购买请求，包括：购买商品的种类和数量、购买客户的姓名和地址等。
- 系统审核用户的购买请求，完成购买，整个过程包括调用外部系统实现付款和送货，以及记录用户的购买历史。

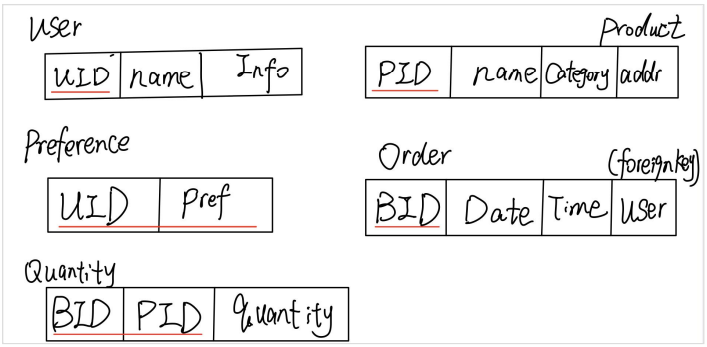
创建ER图

我们对于用户和商品之间的购买关系，有两种处理方式，第一种是用一个简单关系带描述(菱形)，第二种则是通过用户下订单、订单中包含商品的方式来呈现，如下图所示。



那么应该采用哪种方式呢?

选择标准就是这种关系能否由两个实体的ID去确定。比如UID和PID能唯一确定一个购买关系吗？答案是不能的，因为一个用户可以重复下单同一件商品，仅用UID和PID是反映出来的



例题

1. 第 1 题：假设有一个员工表 Employee(ID, Name, Address, Phone#)（其中ID为主键）。最初电话号码Phone#是一个单值属性，即每个人只有一个电话。后来发现，有少量员工有两个电话号码，且都需要记录。你会如何对数据库的结构设计进行调整?

   A：将Phone#的数据类型变成字符串，将两个电话号码用分号隔开，一起存在同一个字段中。

   B：将主键变为 (ID, Phone#)，将有两个电话号码的人在表中存两份。

   C：将模式修改为 Employee(ID, Name, Address)，Phone(ID, Phone#)。

   D：将模式修改为 Employee(ID, Name, Address, Phone#, Phone#Alt)，其中Phone#Alt用于记录另一个电话号码。 ✓

因为只有少量员工有两个电话号码，那么我们没有必要新开一张表去存储多值属性的关系，直接新建一列即可，这一列大多数都是空白的。不会让这张表变得非常宽

1. 第 4 题：假设有一张表：行程表（旅客号码，导游号码，打卡景点），记录了哪些旅客被哪些导游带领去了哪些地方。我们发现了很多如下形式的规律：不同的旅客在同一位导游的带领下，如果去了景点A，那么必定也会去景点B。请问数据库中是否存在冗余信息，可以如何处理?

   A：没有冗余

   B：有冗余，但无法通过结构设计去除 ✗

   C：有冗余，适当改变结构设计后可去除 ✓

   D：以上说法均不对

2. 某关系表R的外键是指：

A. 其他关系表的键，可以是R的任意属性

B. 该关系表R除主键之外的另一个键

C. 其它关系表的键，同时必须作为R的主属性(即R的键包含的属性)

D. 其他关系表的键，同时必须作为R的非主属性(即不被R的键包含)

在了解这个问题之前，我们首先要学习 主码、候选码、主属性、非主属性 的定义

1. 候选码的定义：如果关系中的某一**属性组**的值能唯一地标识一个元组，则称该属性组为候选码；
2. 主码的定义：如果一个关系有多个候选码，则选定其中一个为主码；
3. 主属性定义：候选码的诸属性称为主属性；
4. 非主属性定义：不包含在任何候选码中的属性称为非主属性；

因为我们可以通过两张表的主键来定义一张表格，这两个主键都是这张表的外键。因此，外键在表中也是用来标识一个元组的，是主属性。所以选择C

1. 请针对以下需求设计ER图，并构建相应的关系模式：

   一个关于电影、制作人员和演员的网站（类似一个简易的IMDB网站）。用户可以浏览每一部电影的简介和相关信息（出品年份、电影类型、时长、评级），以及其导演、编剧和演员的列表。用户还可以浏览每一位导演、编剧或演员的信息（姓名、性别、年龄、简介），以及他们参与过哪些电影作品。导演、编剧或演员只是职位，一个人可以身兼数职。

   每一位演员在其参演的电影中都扮演一定的角色。用户在浏览电影时，除了能看到演员信息，还能看到每位演员扮演了什么角色。用户在浏览演员时，除了能看到他（或她）参演的电影，还能看到他（或她）在每一部电影中扮演的角色。（注意：一位演员可以在一部电影中扮演多个角色。同一个角色也可能由多名演员扮演，比如，年少时由一位演员扮演，年老时由另一位演员扮演。）

   用户登录后还可以针对每一部电影、每一位导演、编剧或演员进行评价和打分，供别人参考。

----------本文结束，感谢您的阅读----------