数据库 PJ——图书馆管理系统

实验报告

常朝坤
16307130138

目录

一、基本信息

- 项目类别：图书馆管理系统
- 技术栈：MySQL + Django + bootstrap
- 功能列表

编号	功能	备注
0	登陆	游客模式只能查询，不可借阅等
1	借书	不同用户有不同的借阅限额
2	还书	还书时需要通知第一个预约此书的用户
3	续约	只能连续约两次
4	预约	预约时给出书籍预约状态参考
5	添加书籍	只有超级用户可以添加书籍
6	违约记录	违约分类，多次违约罚款
7	通知系统	通知领取预约的书籍以及建购的书籍(建购即时通知逻辑尚未建立)
8	个人信息	借阅记录的查询以及通知查询
9	书评与打分	匿名评论与打分（可修改评论）

二、数据库 Table 设计

- Table 列表 *(斜体表示暂未用到的表)*

编号	表名	说明	用途
1	Stock	库存表	记录图书信息及库存量
2	Book	图书表,	存储单本书信息
3	Loan	借阅相关记录表	使用 flag 标记记录类别
4	Break	违约表	记录违约的借还记录
5	Fine	罚款表	记录导致罚款的违约记录
6	Employee	职工表	职工信息，职工有分类
7	User	用户表	存储注册的用户信息
8	Reader	读者表	读者信息
9	Review	书评表	存储所有的书评信息

10	Suggestion	建购表	向图书馆建议购买某本书
11	Notification	通知表	由操作产生的通知，比如领取预约书籍的通知
12	*Purchase*	购买表	图书馆购买新书的管理
13	*Reserve*	预约表	预约记录

- Stock 表与 Book 表

考虑到图书馆中的书可能有不止一本，为减少数据库的冗余，将图书信息存在两个表中

➢ Stcok 表存放与书籍相关的信息，包括作者、简介、数量及剩余数量等

```
class LibraryStock(models.Model):
    isbn = models.CharField(max_length=20)
    total = models.PositiveIntegerField()
    remain = models.PositiveIntegerField()
    reserving = models.PositiveIntegerField()
    name = models.CharField(max_length=100)
    author = models.CharField(max_length=50)
    version = models.PositiveIntegerField()
    publisher = models.CharField(max_length=200)
    publish_date = models.DateField(default='2018/5/21')
    price = models.PositiveIntegerField()
    category = models.CharField(max_length=20)
    review_rate = models.DecimalField(max_digits=3, decimal_places=2,)
    review_number = models.PositiveIntegerField()
    introduction = models.TextField(blank=True, null=True)
    modify_date = models.DateTimeField(blank=True, null=True, auto_now=True)
```

注：

✓ 库存表存储了一个 ISBN 属性，其在表中时唯一的，但不作为主键使用，主键使用 id。原因是 ISBN 是 13 位的，相对来说整形的 id 查询复杂度更低。

➢ Book 表存放与单本书相关的信息（如位置、是否在架上等），并添加一个连接到 Stock 表的外键

```
class LibraryBook(models.Model):
    add_date = models.DateTimeField(auto_now_add=True)
    location = models.CharField(max_length=20)
    # sub_date = models.DateTimeField(blank=True, null=True, auto_now=True)
    flag = models.PositiveIntegerField()
    # flag: 0—再架上 1—借出
    # （2—被预约，预约数量是（flag-1）个）
    isbn = models.CharField(max_length=20)
    stock = models.ForeignKey('LibraryStock', models.CASCADE)
```

注：

✓ flag 变量标识此书是否被借出或被预约（用一个变量标记两种信息，这样设计是为了减小数据库的空间）。0 表示在架上，非 0 表示不在架上；1 表示正在被借，n（n>=2）表示正在被预约，且预约的人数位 n-1。

✓ 在 Book 表中，加入了一个 ISBN 号，这个属性看似冗余。但是实际上是在冗余性和

数据查询的复杂度上做出的权衡。大量的查询（主要是预约）操作涉及到这个表，但由于前面 Stock 和 Book 的特点导致查询的查询条件往往为 isbn 或 id，于是添加了冗余，以方便减少查询的复杂度。在 id 和 isbn 的选择中，不选择 id 的原因是 id 是系统内默认的主键，没有实际意义，在效果相同的情况下，选择有意义的量更符合现实意义。

- Loan 表

Loan 表中存储了所有图书与用户的关系记录，包括借阅记录，借阅完成记录，预约记录，以及续借的标记和违约的标记。不同的借阅记录使用 flag 进行标记区分

```
class LibraryLoan(models.Model):
    loan_date = models.DateTimeField(default=timezone.now())
    loan_time = models.PositiveIntegerField(default=30)
    flag = models.IntegerField(default=1)
    # flag 0: 已还记录 1: 未还记录;  >2: 预约记录
    # flag <0 已换且 为违约状态  -1: 逾期违约记录;  -2: 损坏违约记录;
    return_time = models.DateTimeField(blank=True, null=True)
    renew_times = models.PositiveIntegerField(default=0)
    book = models.ForeignKey(LibraryBook, models.DO_NOTHING)
    isbn = models.CharField(max_length=20)
    employee = models.ForeignKey(LibraryEmployee, models.DO_NOTHING)
    user = models.ForeignKey(User, models.DO_NOTHING)
```

注：

- ✓ flag 标记记录类别。自然数表示正常状态：0 表示借还完成的记录，1 表示正在借阅中的记录，2 表示预约中的记录；负数表示违约状态，不同的负数标记违约类别。
- ✓ 由于将三大类记录合并到一个表中带来了一定的冗余，比如所有非完成的记录都不需要 return_time 属性，预约的记录也不需要 loan_time 属性。这种冗余也给程序的设计添加了很多复杂性（代码复杂性）。但是这也是一种权衡的结果，在本系统中，这种选择是更优的。首先这种设计大大减少了数据库的删增操作，比如处理预约记录只需要改动 flag 和 return_time 属性就可以了，如果分开就需要在借阅表中添加记录。这种程序级的优化可以减少数据库的大量更改。另外这也减少了一定的冗余，一般来说，成功预约的记录对管理员的意义并不大，当预约完成或被取消后，这条记录便没有了意义，单独存一张表对于底层数据库的物理读取也会产生压力。

User 表与 Reader 表

这个表格是 Django 的 Auth 表组中的一个，存贮了关于用户的大部分基本信息，包括账号密码邮件等等，偷个小懒使用了 Django 自带的表。但是这个表是通用表，和本程序相关的一些信息（如借阅次数、违约次数等）并没有，在不手动修改 User 表的情况下（这样会很麻烦，而且不安全），需要添加一个 Reader 表存储和本应用相关的用户信息。

- ➢ User 表

```
class AuthUser(models.Model):
    password = models.CharField(max_length=128)
    last_login = models.DateTimeField(blank=True, null=True)
    is_superuser = models.IntegerField()
    username = models.CharField(unique=True, max_length=150)
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=150)
    email = models.CharField(max_length=254)
    is_staff = models.IntegerField()
    is_active = models.IntegerField()
    date_joined = models.DateTimeField()
```

➢ Reader 表

```
class LibraryReader(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    limitation = models.PositiveIntegerField(default=10)

    borrowing_times = models.PositiveIntegerField(default=0)
    breaking_times = models.PositiveIntegerField(default=0)
    fining_times = models.PositiveIntegerField(default=0)
    borrow_number = models.PositiveIntegerField(default=0)
    reserve_number = models.PositiveIntegerField(default=0)
    suggestion_number = models.PositiveIntegerField(default=0)
    rank = models.PositiveIntegerField(default=10)
```

注：

- ✓ 加入外键 user 链接系统表 User。
- ✓ 存储和介于相关的信息，包括正在借阅、预约的数量，违约次数等，便于管理员的管理。

● Notification 表

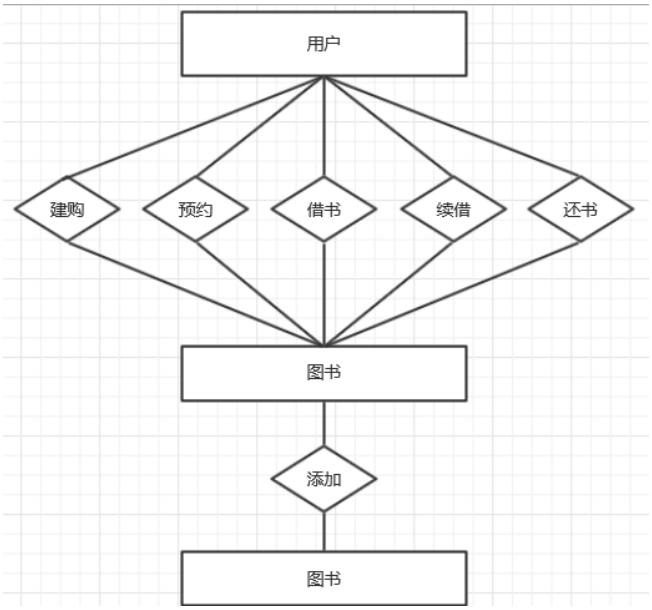
用于程序向用户的通知管理。比如预约的书被归还，通知记录会存储到这个表中。

```
class LibraryNotification(models.Model):
    user = models.ForeignKey(User, models.DO_NOTHING)
    message = models.TextField()
    type = models.PositiveIntegerField()
    date = models.DateTimeField(auto_now_add=True)
    flag = models.PositiveIntegerField(default=0)
    index = models.PositiveIntegerField()
    # type=1 : 预约通知:
    # flag: 0: 创建  1: 已通知  2: 已完成  3: 预期未完成  4: 预约被取消
```

注：

- ✓ Messge 存储通知的信息内容
- ✓ （type，flag）标记通知类别：1 表示预约类别；flag 标记子信息（如图）

三、ER 图设计



四、功能实现

- 用户管理
  - 登陆

```
url(r'^login/$', login, {'template_name': 'users/login.html'}, name='login'),
```

登陆部分后端使用了 Django 内置的登陆管理函数 login，而没有使用 session。原因在于这样方便后端对于登陆权限的控制。

  - 注销

```
url(r'^logout/$', views.logout_view, name='logout'),
```

```
def logout_view(request):
    logout(request)
    return HttpResponseRedirect(reverse('library:index'))
```

注销和 login 相匹配，也采用了内置的管理函数 logout。(但是写了一个视图函数，在视图函数中调用 logout 函数。目的是体验一下不同的使用方法，这样可以为注销添加新的其他特性)

  - 注册

```
url(r'^register/$', views.register, name='register'),
```

```python
def register(request):
    if request.method != "POST":
        form = UserCreationForm()
    else:
        form = UserCreationForm(data=request.POST)

        if form.is_valid():
            new_user = form.save()
            authenticate_user = authenticate(username=new_user.username, password=request.POST['password1'])
            login(request, authenticate_user)
            # 同步Reader表
            user = User.objects.get(username=new_user.username)
            LibraryReader.objects.create(user_id=user.id)
            return HttpResponseRedirect(reverse('library:index'))

    context = {'form': form}
    return render(request, 'users/register.html', context)
```

注册管理是手动编写的管理函数，并且在其中完成了 Reader 表和 User 表的同步。在创建用户时，由于使用了 Auth 的 User 表，所以创建用户也使用了 User 的内置表单 UserCreationForm。

- 个人信息

```python
path('records/<int:user_id>', views.records, name='records')
```

```python
def records(request, user_id):
    all_records = LibraryLoan.objects.filter(user_id=user_id)
    returned = all_records.filter(flag__lte=0).values('id', 'book', 'loan_time', 'loan_date')  # 完成借阅
    borrowing = all_records.filter(flag=1).values('id', 'book', 'loan_time', 'loan_date')  # 正在借阅
    reserving = all_records.filter(flag=2).values('id', 'book', 'loan_time', 'loan_date')  # 正在预约
    broken = returned.filter(flag__lt=0).values('id', 'book', 'loan_time', 'loan_date')  # 违约借阅记录
    personal_info = User.objects.get(id=user_id)
    notifications = LibraryNotification.objects.filter(user_id=user_id).filter(flag__lt=2).order_by('flag')

    for record in returned:...
    for record in broken:...
    for record in borrowing:...
    for record in reserving:...

    context = {'returned': returned, 'broken': broken, 'borrowing': borrowing,
               'reserving': reserving, 'personal_info': personal_info, 'notifications': notifications}
    return render(request, 'users/user_records.html', context)
```

在Loan表中查询所有与此用户相关的信息,Notification表中的通知信息,以及用户信息,并分类返回给 Html 文件及进行渲染。

- 借阅管理
- 图书列表

```python
path('books', views.books, name='books'),
```

```python
from django.core.paginator import Paginator,EmptyPage,PageNotAnInteger
```

```python
def books(request):
    if request.method != 'POST':
        re_book = LibraryStock.objects.all()
    else:
        re_book = LibraryStock.objects.filter(name__contains=request.POST.get("name"))

    paginator = Paginator(re_book, 4)
    page = request.GET.get('page')
    try:
        books = paginator.page(page)  # 获取前端请求的页数
    except PageNotAnInteger:
        books = paginator.page(1)  # 如果前端输入的不是数字,就返回第一页
    except EmptyPage:
        books = paginator.page(paginator.num_pages)
    context = {'books': books, 'msg': "So Cool!"}

    return render(request, 'library/books.html', context)
```

✓ 此处使用了 Django 内置的分页管理函数进行了分页显示处理，一页显示四条记录。

✓ 加入了搜索功能，目前只支持按照书名进行模糊查询。

➢ 借书/还书/续借/预约/建购

```python
path('borrow_gd/<int:stock_id>', views.borrow_guidance, name="borrow_gd"),
path('borrow/<int:stock_id>/', views.borrow_book, name='borrow'),

path('renew_confirmation/<int:book_id>', views.renew_confirmation, name='renew_confirmation'),
path('renew/<int:book_id>', views.renew_book, name='renew'),

path('return/<int:book_id>', views.return_book, name='return'),

path('reserve_confirmation/<int:stock_id>', views.reserve_confirmation, name='reserve_confirmation'),
path('reserve/<int:book_id>', views.reserve_book, name='reserve'),
path('reserve_hd/<int:book_id>/<int:sign>', views.reserve_handle, name='reserve_handle'),

path('suggest', views.suggest_book, name='suggest'),
```

基本上每一个功能都加入了 confirmation 操作确认函数（_gd 也是），这样处理的目的是为了程序的可用性和安全性，便于用户再次确认操作信息，同时也对显示信息进行了分级处理，使得借阅时读者不需要浏览太多无关的信息。（不同操纵的确认函数稍有不同，这和功能相关，比如借阅时需要向根据提交的 stock_id 选择一个合适的 book_id 提供给借阅函数）

● 还书功能

```python
@login_required()
def return_book(request, book_id):
    borrow_logs = LibraryLoan.objects.filter(user_id=request.user.id).filter(flag=1)
    borrow_log = borrow_logs.get(book=book_id)

    # 修改 loan 表
    if borrow_log is None:
        msg = '没结果这本书，请先借阅'
        return HttpResponseRedirect(reverse('library:errormsg', args=[msg]))

    time_now = timezone.now()
    time_lag = time_now - borrow_log.loan_date
    print(time_lag)
    print(time_lag.days)
    if time_lag.days > borrow_log.loan_time:
        borrow_log.flag = -1
        # 违约记录一下
        break_log = LibraryBreak.objects.create(reson='逾期未还', break_date=time_now,
                                                punishment='上小本本', user_id=request.user.id)
        if LibraryReader.objects.get(user_id=request.user.id).breaking_times > 3:
            fine_money = (LibraryReader.breaking_times - 3) * 10
            break_log.punishment = '罚款' + str(fine_money)
            fine_log = LibraryFine.objects.create(fine=fine_money, break_id=break_log.id)
            fine_log.save()
        LibraryReader.objects.get(user_id=request.user.id).breaking_times += 1
        break_log.save()
    else:
        borrow_log.flag = 0
```

```python
    borrow_log.return_time = time_now

    # update Book table
    r_book = LibraryBook.objects.get(id=book_id)
    r_book.flag = 0
    # update Stock table
    r_stock = LibraryStock.objects.get(id=r_book.stock_id)
    if r_stock.reserving > 0:
        # 通知第一个预约的人可以去借预约的书了，存储预约通知表，在表里记录
        # 在loan表里寻找最早预约了这本书的人
        reserving_logs = LibraryLoan.objects.filter(flag=2).filter(book=book_id).order_by('loan_date')
        if reserving_logs.exists():
            reserving_log = reserving_logs[0]
            reserving_user = User.objects.get(id=reserving_log.user_id)
            notify_message = reserving_user.username + 'Hello!' + 'Your reserving book' + r_stock.name + '(with isbn:' +
                             r_stock.isbn + ')can be borrowed now, please go to your personal page to get it!'
            # 通知Ta来取书
            notify_log = LibraryNotification.objects.create(user_id=reserving_log.user_id, message=notify_message,
                                                            flag=0,
                                                            index=reserving_log.id, type=1)
            notify_log.save()
    else:
        r_stock.remain += 1

    borrow_log.save()
    r_book.save()
    r_stock.save()

    return HttpResponseRedirect(reverse('users:records', args=[request.user.id]))
```

PEP 8: line

- ✓ 还书时需要确认用户借了这本书，而且需要查询正在预约这本书的用户，找到最早预约这本书的用户，并建立通知，通知给该用户预约的书籍可以领取。
- ✓ 还书时也需要确认此次借阅是否违约，如果违约需要存储违约记录，如果违约次数过多，

需要加入罚款记录，罚款金额和违约次数相关。

- 建购功能

向图书馆管理员提供建购的 ISBN 号，建议图书馆购买次数，记录存储在 Suggestion 表中。

- 书评和打分功能

```
path('new_review/<int:stock_id>', views.new_review, name='new_review'),
path('edit_review/<int:review_id>', views.edit_review, name='edit_review'),
```

```
def new_review(request, stock_id):...


@login_required
def edit_review(request, review_id):
    review = LibraryReview.objects.get(id=review_id)
    stock = review.stock
    # 确认用户，保护edit
    if review.user != request.user:
        raise Http404

    if request.method != "POST":
        form = ReviewForm(instance=review)
    else:
        form = ReviewForm(instance=review, data=request.POST)
        if form.is_valid():
            form.save()
            return HttpResponseRedirect(reverse('library:bookinfo', args=[stock.id]))

    context = {'review': review, 'stock': stock, 'form': form}

    return render(request, 'library/edit_review.html', context)
```

✓ 登陆的用户可以在书籍详细信息中添加评论（匿名）。
✓ 用户可以看到所有评论，但只可以编辑自己评论

五、前端设计

- 使用了 Bootstrap 官方文档的 Cover 样例进行继承修改
- 采用了根据窗口大小自动调节的布局
- 使用了 Bootstrap.css 中的渲染类
- 使用了网上的一个简介的登陆界面
- 在图书列表加入了分页管理的控制逻辑
- 加入了提示界面

六、实验感想

大学目前位置做过最好玩的PJ，没有之一。虽然占分不高，但是热情很高，感觉学到很多东西，过程很有趣。

数据库PJ最重要的就是数据库设计，数据库设计不好，程序会很难写。

一个功能完善的程序真的时需要考虑好多好多方面