实验 5 数据库程序设计

3170105392 袁轶珂

一、 实验目的

1、设计并实现一个简洁的图书管理系统，具有入库、查询、借书、还书、借书证管理等基本功能。

2、通过本次设计来加深对数据库的了解和使用，同时提高自身的系统编程能力。

二、 实验平台

开发语言：python 3.7

调用库：

	configparser	3.7.4	支持配置文件的设置
	Pyinstaller	3.4	支持单应用程序打包
	Pymysql	0.9.3	支持对 mySQL 数据库的调用
	Pyqt5	5.11.3	支持 GUI 设计

数据库平台：MySQL

三、 总体设计

3.1 系统功能设计

本系统实现了图书管理的基本功能如下表：

功能	描述
图书查询	根据书籍的各项信息进行图书的查找 在列表中显示所有符合查询范围的书籍

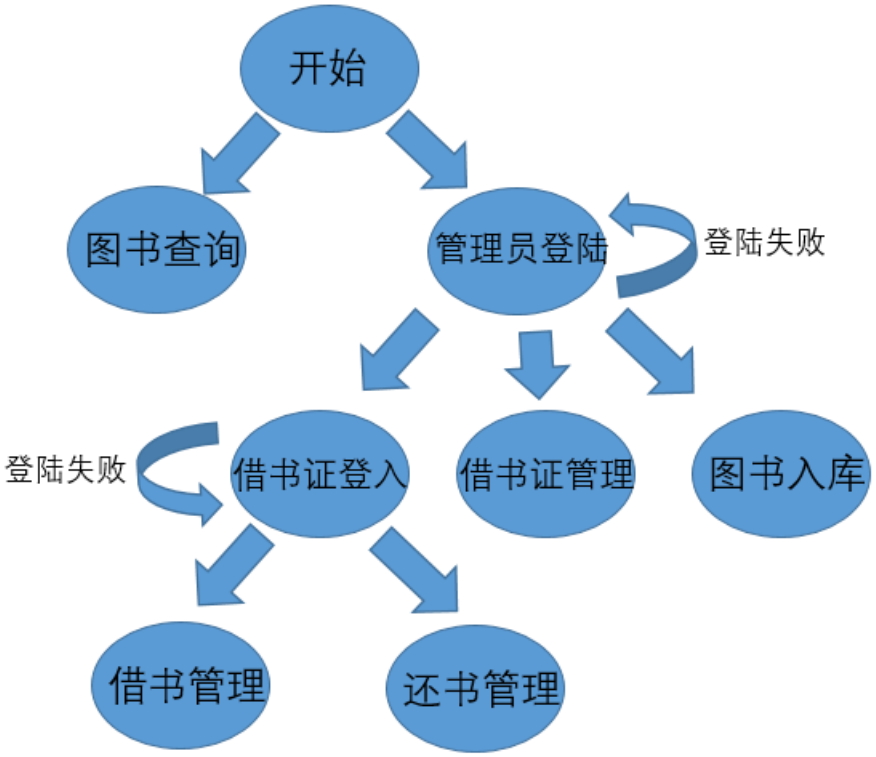
管理员登陆 /退出登录	输入管理员账号和密码进行登录，解锁借还书界面 退出登录后可重新输入账号密码登录
借书管理	输入借书证卡号自动显示对应持卡人并进入界面 界面显示持卡人所有借书记录 输入书号进行借书
还书管理	输入借书证卡号自动显示对应持卡人并进入界面 界面显示持卡人所有未还书籍记录 单击记录进行还书
书籍管理	共包含三个功能，书籍入库、在库增删和批量入库 书籍入库为单本入库，输入书籍相关信息进行入库 在库增删为单本库存信息变更，输入书号对库存和总量进行调整 批量入库通过导入文本文件实现，根据文本内容对多本书进行入库
借书证管理	共包含三个功能，添加借书证、删除借书证和查看借书证 添加借书证即输入新借书证相关信息即可创建 删除借书证时输入卡号显示对应持卡人姓名若确认则删除 查看借书证会将当前数据库内所有借书证信息显示在列表内

3.2 数据库架构

表	成员	主键
Books	书号、类别、书名、出版社 、出版时间、作者、价格、总藏 书量、库存、更新时间	书号

LibraryCard	借书卡号、姓名、单位、类别、更新时间	借书卡号
Users	用户名、密码、姓名、联系方式	用户名
LibraryRecords	借书卡号、书号、借书日期、还书日期、操作人	借书卡号、书号

3.3 流程图设计



图书查询模块不需要管理员身份即可进入，而借书证管理、图书入库模块需要在登录管理员的状态下才能进入。并且，在管理员登录模式下，再登录某一人的借书证就可以进行该持卡人的借还书管理。

3.4 开发设计

本次开发所用的语言为 python，开发平台为 pycharm。

Python 是一种面向对象的动态类型脚本语言，python 由于其简洁明了易于阅读的代码风格和强大的开源可扩展性，在近些年来越来越多的被用做大型项目的开发语言。Python 的库多而丰富，利用这些库就可以完成对数据库和图形界面的设计。

Pycharm 是由 JetBrains 打造的一款 Python IDE，具有十分优秀的编码协助和调试功能。

本次开发中所用到的数据库为 MySQL。

MySQL 是一个开源的关系型数据库管理系统，在 WEB 应用方面有着十分出色的表现，其所使用的 SQL 语言是访问数据库的标准化语言。在本项目中，我们利用 python 的库 pymysql 对数据库进行连接。

本次开发所用到的图形界面开发框架为 QT。QT 原本用于 C++语言的图形界面开发，后开发出了 PyQt 库用于 Python 编程语言，基本实现了 QT 的所有模块和功能，本项目就是使用该模块集对图形界面进行了设计。

四、 详细设计

1、 数据库初始化

为系统正常运转，我们需要编写一个预运行的 SQL 脚本对数据库进行初始化，包括表的建立和部分数据的填充以使得系统能够正常运行。表的结构在模块三总体设计的数据库架构部分已经进行过阐释，在此不再重复。

2、 数据库连接类

本项目中利用 pymysql 库对 mySQL 数据库进行访问。利用库中 connect 函

数对数据库进行访问：

```
self.conn = pymysql.connect(host=db_host, port=db_port,
user=db_user, passwd=db_pass, db=db_name)
```

函数的各参数由配置文件的读取得到，直接调用 configparser 库中的 get 函数即可。

在获得链接之后将链接的成员光标保存为连接类的公有成员。通过对光标的成员方法 execute 的调用即可实现数据库的访问，execute 函数的参数即 SQL 语句字符串。

为使调用时更加方便，我们对不同类型的 SQL 语句进行封装，将 SQL 语句的各部分作为列表参数，由写好的函数帮助生成 SQL 语句，下列函数有：

```
def select_sql(table, keys, conditions=''):
    """
    查询语句
    :param table:表名
    :param keys: 需要查询的属性
    :param conditions: 查询条件
    :return: 可执行查询语句
    """

def delete_sql(table, conditions):
    """
    删除语句
    :param table: 表名
    :param conditions: 删除条件
    :return: 可执行删除语句
    """

def update_sql(table, updating, conditions=''):
    """
    更新语句
    :param table: 表名
    :param updating: 需要更新的属性
    :param conditions: 属性筛选条件
```

```
    :return: 可执行更新语句
    """


def insert_sql(table, attrList, valueList):
    """
    插入语句
    :param table: 表名
    :param attrList: 属性列表
    :param valueList: 值列表
    :return: 可执行插入语句
    """
```

要接受数据库的返回值如 select 查询结果，需要调用光标的 fetch 方法，同样为使用方便，我们也封装一个连接类的成员函数 fetch_result，将返回值与其对应的参数值组成字典，多行返回结果则返回一个字典列表。

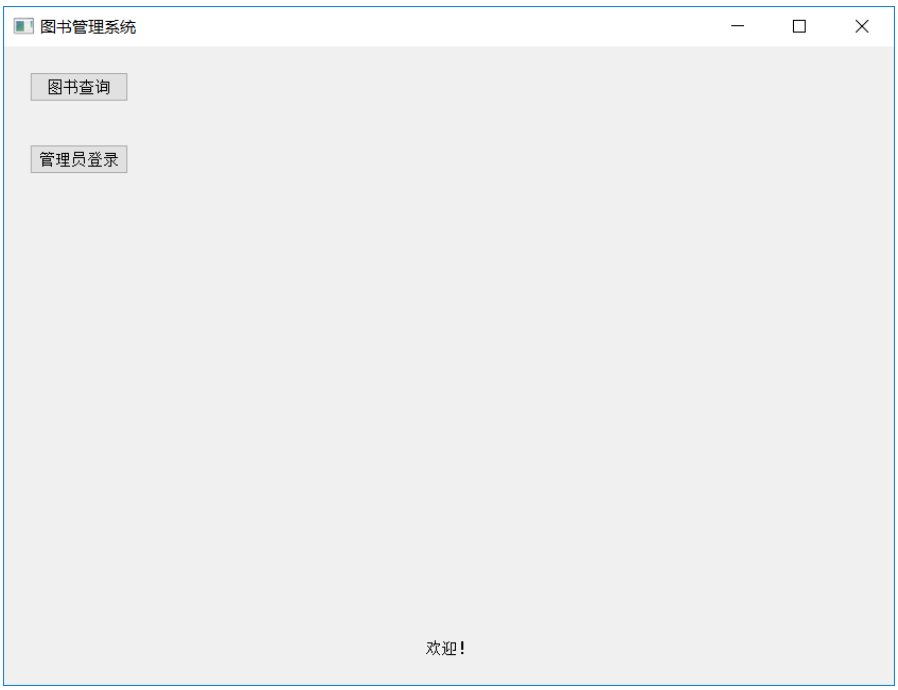
```
def fetch_result_list(self):
    """
    返回数据库结果(字典列表形式)
    :return: 字典列表
    """
```

3、 主界面设计

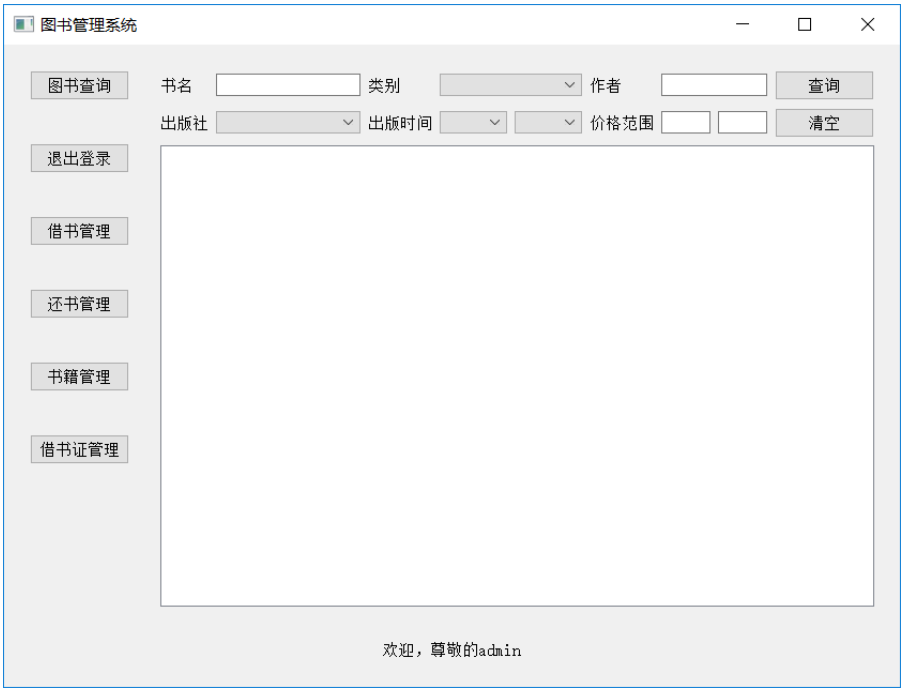
界面采用导航栏式的设置。左侧为导航栏，其中是当前所有可用功能的按钮，右侧为独立窗体，窗体中内容根据点击左侧按钮所选择的服务而改变，在初始化界面中为空。

在设计主界面时，我们考虑到在实际的图书管理系统应用中，图书查询功能是对所有人开放的，而如借还书和借书证管理功能是只对管理员开放而不对普通读者开放的。所以在设计主界面时，我们设定根据当前用户状态对导航栏和内容窗体的显示进行改变。即在管理员登陆之前导航栏只有图书查询一个功能按钮，当管理员登陆成功之后则会出现借还书和借书证管理等功能按钮。

登录前界面实现效果如下：



登录后界面实现效果如下:



这个效果的实现我们利用了 QT 的 stackedWidget 类，这个类相当于一个 Widget 数组，我们可以把同一窗体待切换的内容加载进栈，并通过索引来控制当前显示内容。核心代码如下:

```
self.stack = QStackedWidget()
self.stack.addWidget(self.stack_NULL_0)
```

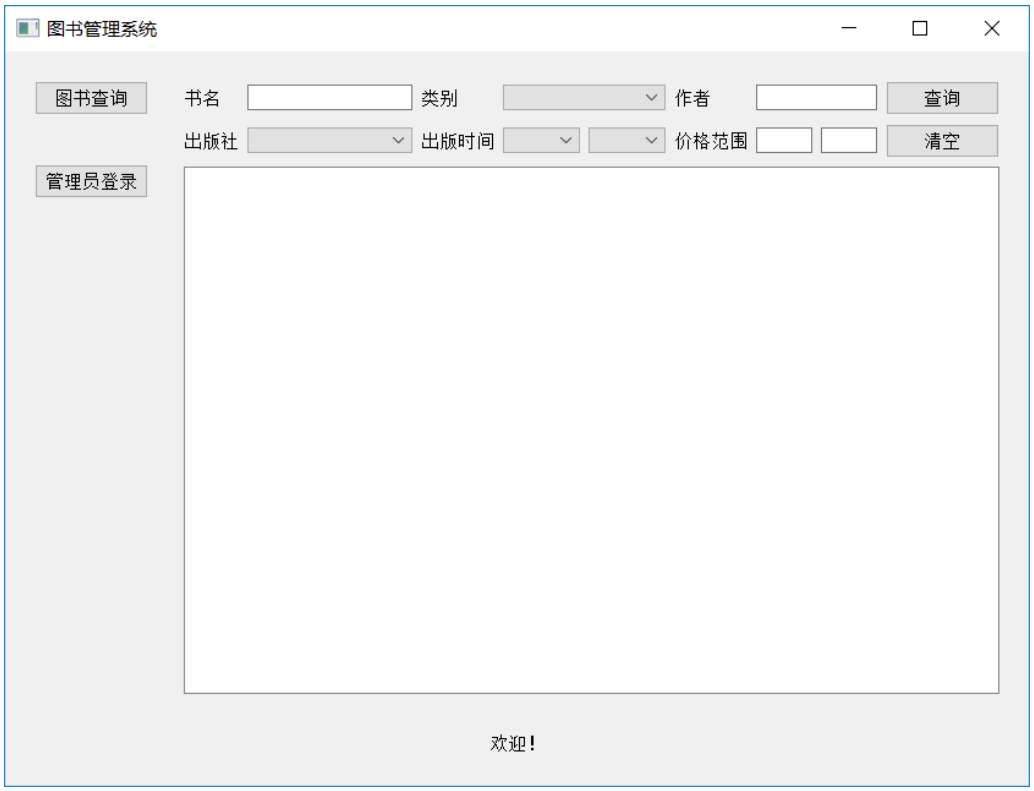
```
self.stack.addWidget(self.stack_query_1)
...
self.stack.setCurrentIndex(0)
```

stack_***_index 即自己编写的继承 QWidget 的各窗体。

在整个窗体的最下部分我们模仿 QMainWindow 类设置了一个状态栏用于显示现在的用户状态。在管理员登录之前显示“欢迎！”，在管理员登录之后会显示“欢迎，尊敬的***”，其中*部分会填充为当前登录的管理员姓名，具体实现会在管理员登陆界面设计中进行详细阐释。

4、 图书查询模块设计

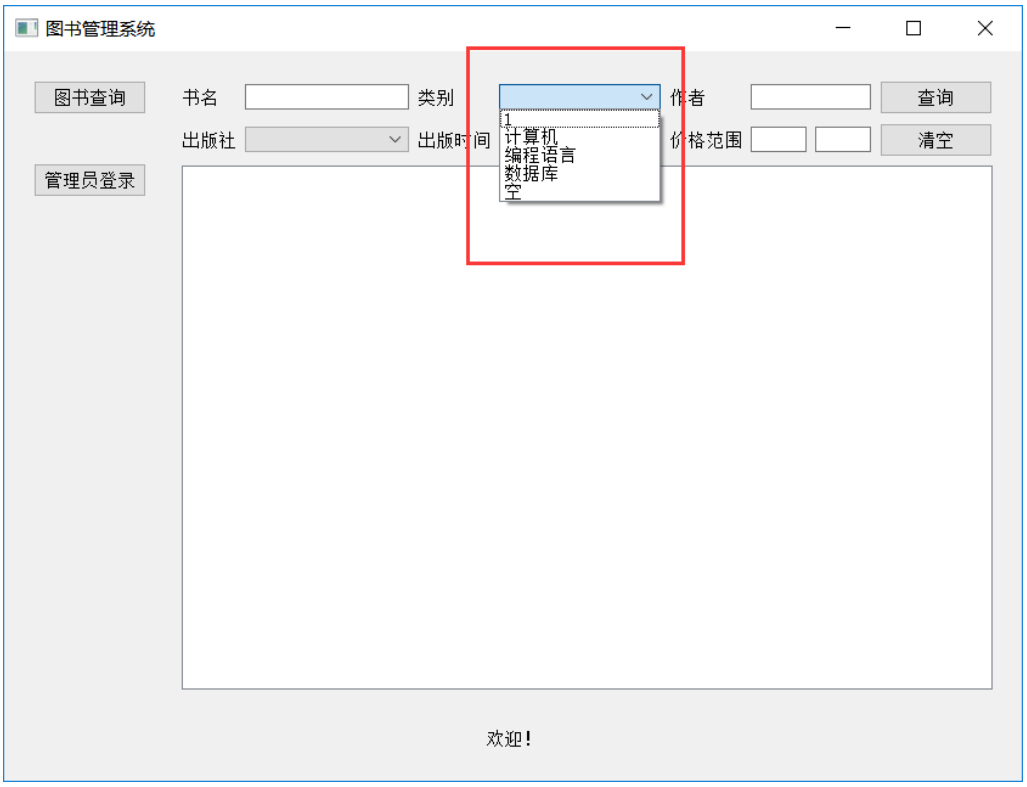
图书查询模块主要分两部分组成，上半部分为查询条件设置模块，下半部分为查询结果显示模块。查询条件设置利用 QT 的网格布局 QGridLayout 实现，而两个模块组成上下结构 QVBoxLayout 构成整个查询模块，实现的界面效果如下：



查询模块的逻辑条件做交运算，未输入的框默认为不限条件，查询结果必须

同时满足所有已输入框中的条件才会显示在下方的列表中。

结合实际使用情况来看，诸如类别、出版社等属性有相对固定的某些选择，不像书名属性变换较大，所以书名、作者等属性的限制条件我们设置为输入框（QLineEdit），而类别、出版社等属性的输入我们设置为复选框（QComboBox），用户只需要在所有的选项中进行选择即可，其实现效果如下：



实现复选框的逻辑需要结合对数据库的使用。我们设置当每次打开图书查询界面时，复选框会对其可选择项进行初始化。对数据库的中的 Books 表进行 select 查询，将所有不同的类别值存为列表加入复选框中，出版社和出版时间亦然，SQL 查询语句为：

```
SELECT BookType, Publisher, Year FROM Books
```

在用户完成条件的输入之后单击查询按钮将该条件对应的查询结果显示在下方的表中。逻辑实现利用了 QT 的信号槽机制，当按钮被单击时该按钮发送一个 clicked()信号，我们需要写一个函数放到槽中，将该信号和槽连接起来，这样

当信号被发送时我们就可以执行相应的查询函数。

查询函数的内容主要是构造有关查询的 SQL 语句，我们取得代表各个条件的控件值，并将其作为参数传入 SQL 语句构造函数，执行 SQL 语句并接受返回结果，其实现代码如下：

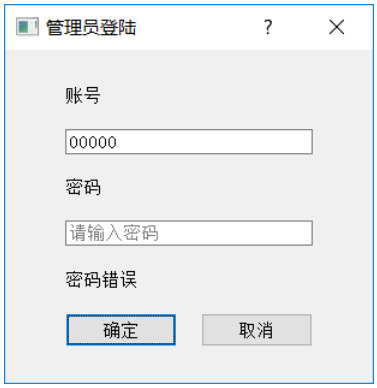
```
conditions = ''
if self.lineEdit_name.text() != '':
    conditions += "BookName = '" + self.lineEdit_name.text() + "' and "
if self.lineEdit_author.text() != '':
    conditions += "Author = '" + self.lineEdit_author.text() + "' and "
if self.lineEdit_pricelow.text() != '':
    conditions += 'Price >= ' + self.lineEdit_pricelow.text() + ' and '
if self.lineEdit_pricehigh.text() != '':
    conditions += 'Price <= ' + self.lineEdit_pricehigh.text() + ' and '
......
select = database.select_sql('Books', '*', conditions)
self.db.execute(select)
attrlist = ['书号', '类型', '书名', '出版社', '出版时间',
            '作者', '价格', '总藏书量', '库存', '最近更新']
reslist = self.db.fetch_result(attrlist)
```

利用返回的字典列表我们将值显示在屏幕上。此处利用了 QTableWidget 控件完成，我们对行数和列数进行设置后，利用一个二重循环以行列坐标的形式将查询结果添加到表单中：

```
self.table.setColumnCount(len(attrlist))
self.table.setRowCount(len(reslist))
self.table.setHorizontalHeaderLabels(attrlist)
for i in range(len(attrlist)):
    for j in range(len(reslist)):
        self.table.setItem(j, i,
QTableWidgetItem(reslist[j][attrlist[i]]))
```

5、 管理员登录界面设计

由于管理员所控制的系统功能多于游客所控制的系统功能，所以我们需要设置一个用于管理员登陆的确认界面以确认当前用户状态。其实现效果如下：

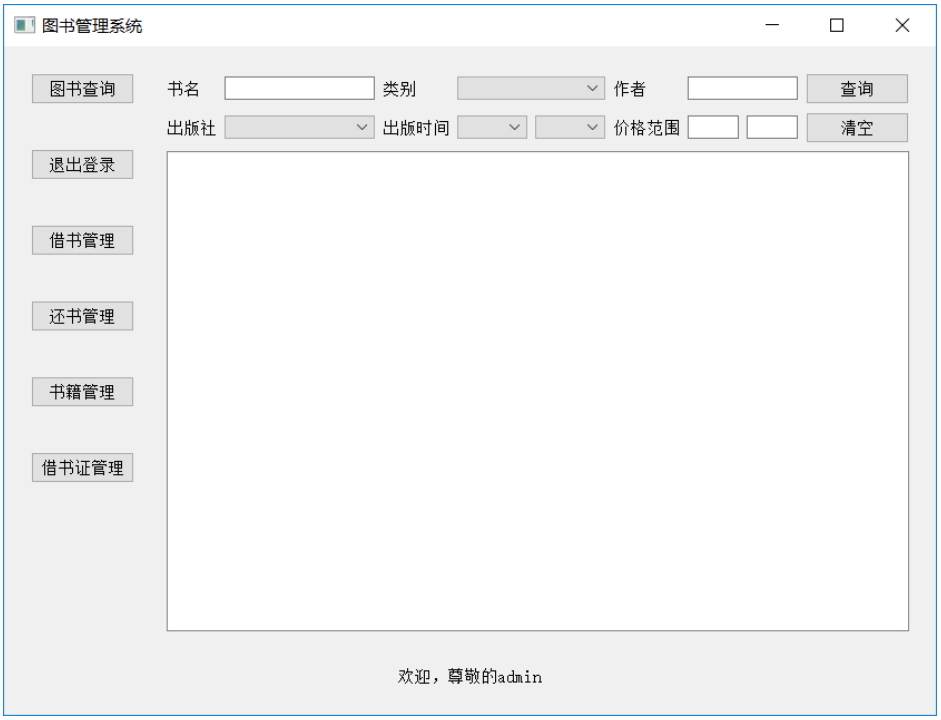


该界面通过继承 QT 的对话框控件类 QDialog 实现，同样采用垂直布局设计。使用两个 QLabel 和两个 QLineEdit 来实现账号密码的提示语和输入框。在确定和取消按钮与输入部分中间我们加了一个 QLabel 做提示框，当用户输入信息完成单击确定按钮之后，若输入信息有误，我们将会在该提示框中输出错误信息，引导用户重新输入正确信息。

对管理员信息的查证要访问数据库中的 Users 表，读取用户输入的用户名信息，以该信息作为条件筛选出对应的用户名、密码和管理员名称等信息。SQL 查询语句为：

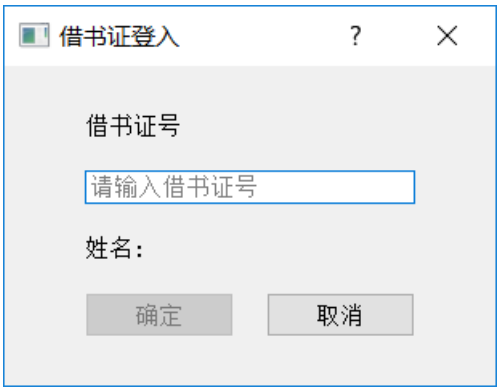
```
SELECT UserID, Password, Name FROM Users WHERE UserID = id
```

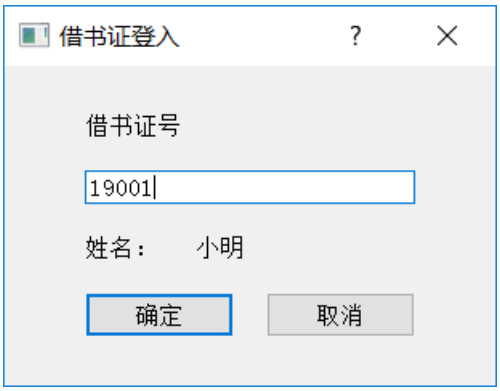
此处有两种情况，若用户名不在数据库中则返回空结果，提示框中显示用户名不存在；若返回记录，则对返回值中的密码进行匹配，若匹配成功则成功登陆，若匹配不成功则显示密码错误。登录失败后会自动清空输入错误的部分，若登陆成功则进入管理员模式下的主界面：



6、 借书模块设计

在实际应用图书管理系统时，借还书的操作一定是与某一借书证一一对应的，所以在设计本系统时，我们规定一定要登入某一借书证才能进入借书界面，当然在借书界面中也设置了相关切换用户的按钮，这样可以更好的处理一个人借多本书的情况，不需要反复录入借书证号即可流畅借书。所以在此处我们首先阐述借书证登陆界面的逻辑：（左侧为输入借书证号前界面，右侧为输入后界面）





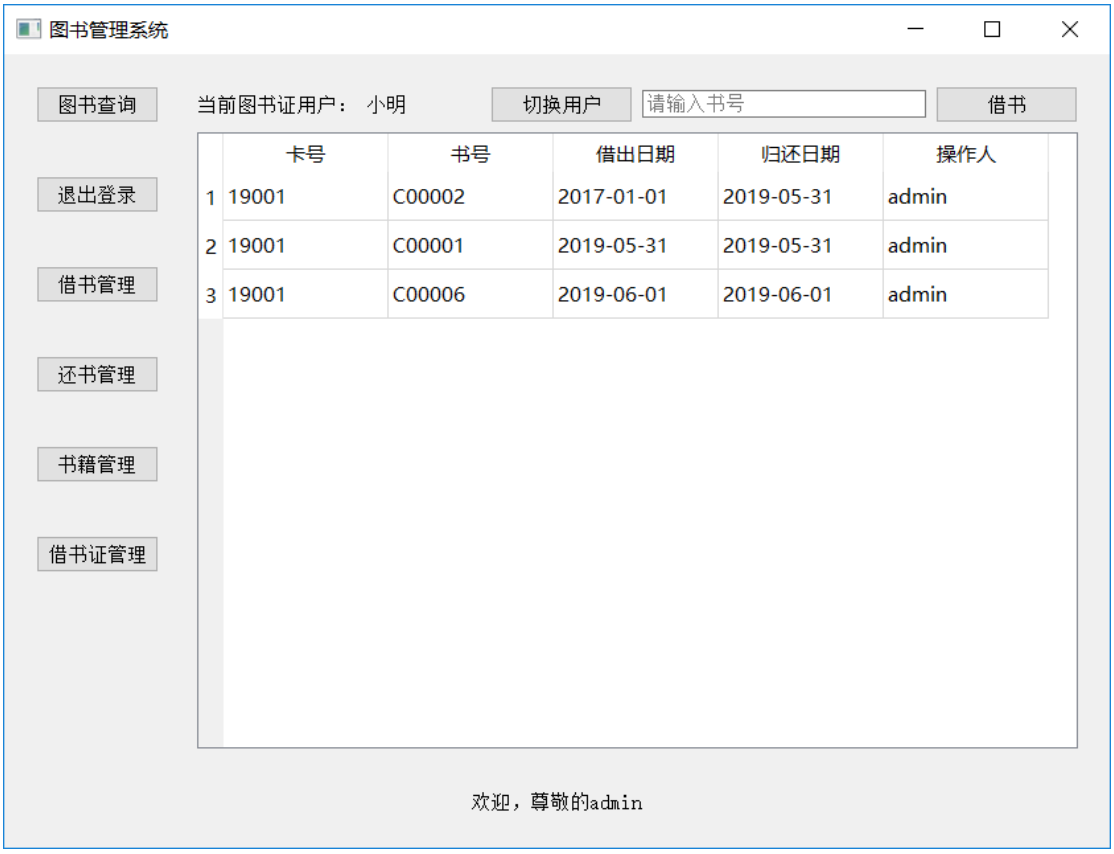
此处界面的逻辑以及布局与管理员登录界面基本一致，而考虑到实际应用中

的情况在此处添加了一个小细节：自动查询用户和按钮的可用状态变化。借书证的登陆与管理员登陆不同是没有密码的，只需要一个输入框即可。而为了方便管理员向该借书证持卡人确认身份，我们在此处设定输入借书证号会自数据库中查询出该借书证号对应的姓名并显示出来，当显示出姓名时表明该借书证号输入正确且存在于数据库中，此时确定按钮变为可按状态表示可以正确登录。

该功能的实现其实并不复杂，QLineEdit 控件中包含一个 textchanged 信号，当输入框中内容改变后就会发送该信号，我们利用该信号编写对应槽函数，函数功能为取得更新后的输入框控件值并以该值为条件进入数据库查询，若有查询结果则将其显示在姓名之后，SQL 查询语句如下：

```
SELECT CardNo, Name FROM LibraryCard WHERE CardNo = account
```

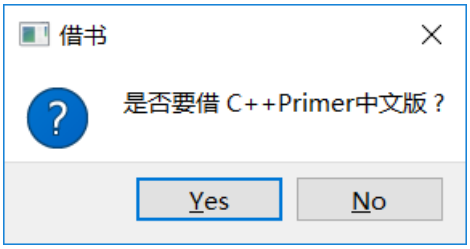
成功登陆后，主界面的右侧窗体会显示借书界面。切换用户按钮可以重新调用借书证登录对话框，在输入框中输入书号后单击借书按钮可以进行借书，下方表格显示的是当前登录用户的所有借书记录。



在图书馆选择借阅某本书时是肯定知道该书的准确书号的，所以这里我们没有采用多属性交叉比对的方式而是只使用书号来进行借书，由于书号是主码所以借书结果一定是准确的。当用户输入书号单击借书之后我们针对该书号进入数据库查询，SQL 语句如下：

```
SELECT BookName, Storage FROM Books WHERE BookNo = No
```

为防止错误输入书号等情况出现，我们在借书按钮之后增加了一个询问对话框，在该对话框中输出刚刚的查询结果以做二次确认，界面效果如下：



当用户单击确认之后我们要对书籍信息做一次比对，根据输入的书号查询该书相关信息，若是当前借书证已借过该书则不能重复借阅并且若库存不足也同样

不能借阅。在上一句查询语句中我们已经查询到了 Storage 信息，只需要再查询借阅记录即可，SQL 语句如下：

```
SELECT BookNo FROM LibraryCard WHERE CardNo = readerID
```

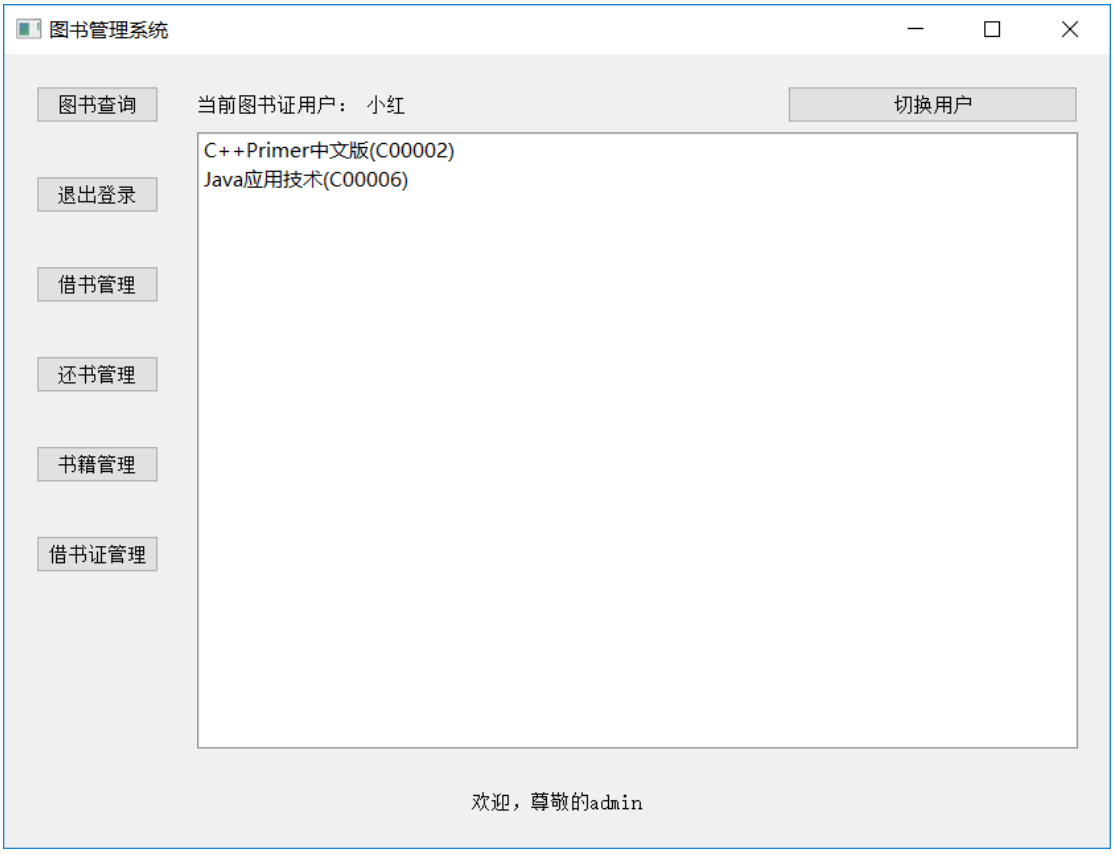
检查过可以借阅之后，我们就要对数据库中的数据进行更新，主要分为两部分，一部分是对书籍库存的更新，另一部分是对读者借书数据的更新。库存更新利用 Update 语句使得 Storage = Storage – 1，借书数据即 insert 一条新的数据在当前借书证下。其实现代码如下：

```
# storage - 1
conditions = "BookNo = '" + bookID + "'"
update = database.update_sql('Books', 'Storage = Storage - 1',
conditions)
self.db.execute(update)
# insert record
insertList = [self.readerID, bookID, str(datetime.date.today()),
self.adminName]
for i in range(len(insertList)):
    insertList[i] = "'" + insertList[i] + "'"
insert = database.insert_sql('LibraryRecords', ['CardNo', 'BookNo',
'LentDate', 'Operator'], insertList)
self.db.execute(insert)
self.db.commit()
self.initTable()
```

7、 还书模块设计

还书界面的进入方式与借书界面一致，都是需要先登录某个借书证用户，在界面中也同样设置了切换用户的按钮，在此不再重复阐述。

当我们登录某个借书证用户之后，我们就可以进入还书界面，该界面包含当前用户的名称，切换用户按钮和一个包含未还书记录的列表，单击该列表项即可执行还书，其界面实现效果如下：

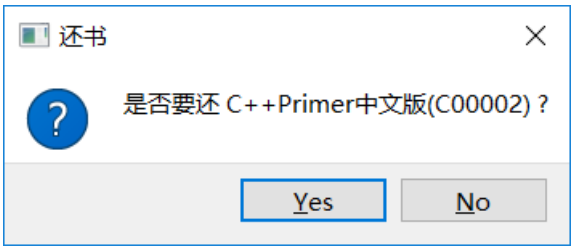


右侧窗体中的列表实际上是未还书列表，也是通过数据库查询结果实现。在借书界面中，插入新借书记录时归还时间一项未赋值，所以对已借但未还的书来说其区别就是归还时间属性是否为 None。在还书界面中，我们只将未还的书目插入列表中，列表的效果继承 QList 控件类实现，其具体实现代码如下：

```
def initList(self):
    self.borrowList.clear()
    conditions = "LibraryRecords.BookNo = Books.BookNo and
LibraryRecords.CardNo = '%s'" % self.readerID
    conditions += " and isNULL(LibraryRecords.ReturnDate)"
    select = database.select_sql('LibraryRecords, Books',
['Books.BookNo', 'BookName'], conditions)
    self.db.execute(select)
    reslist = self.db.fetch_result(['BookNo', 'BookName'])
    mlist = []
    for tup in reslist:
        temp = "%s(%s)" % (tup['BookName'], tup['BookNo'])
        mlist.append(temp)
```

```
    self.borrowList.addItems(mlist)
```

当单击列表中某一项后，槽函数将根据信号发送者判断要还哪一本书，并弹出相应对话框再次确认，对话框效果如下：



若确认要进行还书操作，则进入本系统的还书逻辑。与借书逻辑类似，还书逻辑对数据库的数据改动也分为两部分，更新该书所对应的借书记录，在其中添加还书日期，利用 update 语句实现；更新当前书目的库存，也是 update 语句使得 Storage = Storage + 1，具体实现代码如下：

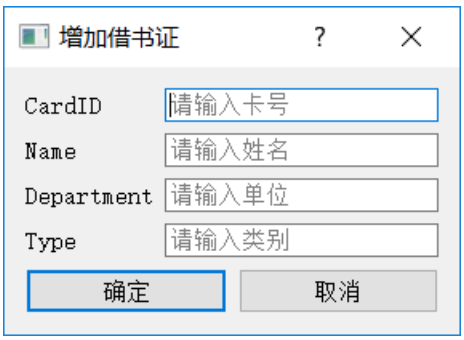
```
# update ReturnDate
date = str(datetime.date.today())
conditions = "CardNo = '%s' and BookNo = '%s'" % (self.readerID,
bookNo)
set = "ReturnDate = '%s', Operator = '%s'" % (date, self.admin)
update = database.update_sql('LibraryRecords', set, conditions)
self.db.execute(update)
# update storage
conditions = "BookNo = '%s'" % bookNo
update = database.update_sql('Books', "Storage = Storage + 1",
conditions)
self.db.execute(update)
self.db.commit()
QMessageBox.about(self, '还书', '还书成功！')
self.initList()
```

8、 借书证管理模块设计

本模块主要实现管理员对借书证的管理，包含查看、添加、删除三个功能。右侧窗体主要由三个按钮和一个表单组成，三个按钮分别对应借书证管理的三个

功能，而下方表单则是用于显示借书证的查询结果。界面设计如下：

接下来将按功能顺序进行一一介绍：

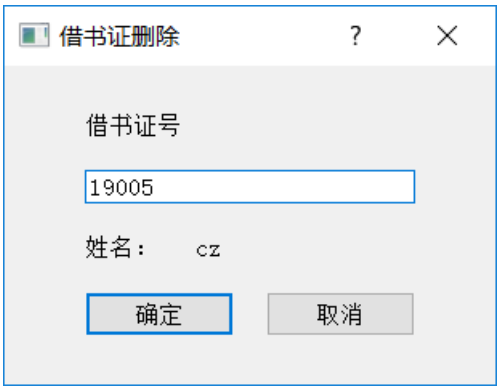


首先是增加借书证模块。由于主界面窗体控件有限，所以该功能与删除功能都利用对话框（QDialog）实现。对话框中包含一张借书证涉及到的所有属性的填写框，在用户单击确定之后我们对取得的数据进行完整性校验以保证其能正确的插入数据库中，校验内容包括四个属性必须不能为空和用户名作为主码不能与数据库中重复。

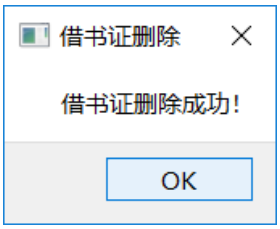
在通过完整性校验后基本可以确定能够正常插入数据库中，我们只需要调用

insert 语句并在之前属性的基础上补上今日的日期即可。其实现代码如下：

```
# insert
attrList = ['CardNo', 'Name', 'Department', 'CardType', 'UpdateTime']
insertList = [id, name, department, type, str(datetime.date.today())]
for i in range(len(insertList)):
    insertList[i] = "'" + insertList[i] + "'"
insert = database.insert_sql('LibraryCard', attrList, insertList)
self.db.execute(insert)
self.db.commit()
```



然后是借书证的删除模块。与借书证用户登录界面类似，该模块使用了相同的语句逻辑使得在输入借书证号后能出现相匹配的姓名做确认。（具体逻辑见 6、借书界面设计）在成功删除之后会出现相应的提示框如下：



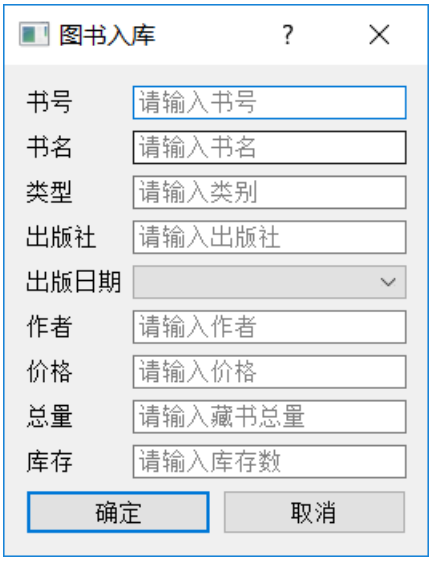
最后是借书证查看界面，该界面的实现逻辑也较为简单，通过 SQL 语句对数据库中的 LibraryCard 表进行查询，将查询结果字典列表逐项插入表单中即可，其 SQL 语句如下：

```
SELECT * FROM LibraryCard
```

9、 图书入库模块设计

本模块主要实现了对图书的相关处理，共包含单本入库、批量入库、在库书籍数量调整三个功能，下方的表单部分是当前所有书籍的全部属性信息。界面实现效果如下：

界面上的书籍入库按钮对应的功能是单本入库，可以将一本书籍的信息插入到数据库中。同样我们利用对话框（QDialog）来实现这一功能，与之前的对话框不同的是本次出版日期一项由于只能是日期格式所以采用了复选框（QComboBox）来进行实现。界面实现效果如下：



与增加借书证相同，在取得数据后我们同样需要进行完整性校验。校验内容

除各项属性不能为空之外，还需要对数据类型进行控制：

BookNo	书号为主码不能重复
Price	必须为整数
Total	必须为整数
Storage	必须为整数且小于 Total 值

满足各项要求之后我们就可以调用 insert 语句插入一条新的图书记录，其实现代码如下：（由于部分属性值为字符串，我们给这些值加上引号）

```
# insert
for i in range(len(insertList)):
    if i in [0, 1, 2, 3, 5, 9]:
        insertList[i] = "'" + insertList[i] + "'"
insert = database.insert_sql('Books', attrList, insertList)
self.db.execute(insert)
self.db.commit()
```

批量入库的功能与单本入库类似，要实现一次性对重复的许多属性进行读取和插入使用文件比较合适。所以单击批量入库按钮会调用 QT 的文件对话框（QFileDialog），我们限定读取文件类型为文本文件。读入文本文件的内容之后我们对字符串做分析将文件内容转变为元组为 key 属性为 value 的字典列表。

批量入库书籍信息文本文件的格式十分简单，只需要按照书号、类别、书名、出版社、出版时间、作者、价格、总量、库存的顺序将属性排列好，中间用逗号隔开不带空格，每一本书占一行即可。举例如下：

```
C00006,编程语言,Java 应用技术,机械工业出版社,2000,谭浩强,100,100,50
C00007,数据库,Hadoop 实战,人民邮电出版社,2004,柯尼汉,300,100,20
```

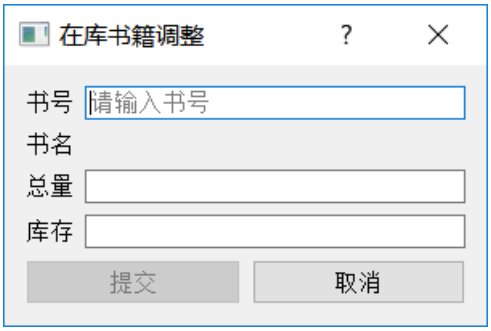
字符串分析的具体步骤如下。首先调用 readline 函数读入一本图书所代表的信息，然后利用 strip 方法去除结尾的换行符，再利用 split 方法以逗号分割开所

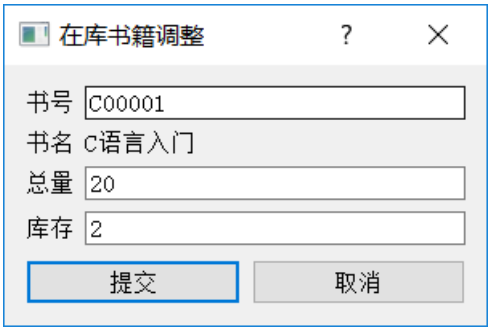
有字符串，将其作为列表传入 check 函数做完整性校验，校验内容与单本入库时一致。以上步骤不断循环直到读入整个文件为止。实现代码如下：

```
with f:
    data = f.readlines()
    for i in range(len(data)):
        insertList = (data[i].strip()).split(',')
        checked = self.check(i, insertList)
        if not checked:
            return
        insertList.append(str(datetime.date.today()))
        for i in range(len(insertList)):
            if i in [0, 1, 2, 3, 5, 9]:
                insertList[i] = "'" + insertList[i] + "'"
        insert = database.insert_sql('Books', attrList, insertList)
        self.db.execute(insert)
self.db.commit()
```

在通过 pymysql 库调用 mySQL 时，如果是对数据库进行修改的语句例如 insert、update 等，执行后必须提交（commit）才能实际发生效果。若不提交则在数据库光标关闭时这些改动就会丢失。在这里我们利用这一点来处理批量入库的异常中断。在不断读取文件的循环中，若完整性校验部分失败说明其中有一行书目是不符合要求的，程序直接退出该函数，前面的插入语句虽然已经执行但是未提交，就不会影响到下一次再读取同一个文件造成重复的插入。

最后是在库书籍调整功能，这个功能的应用范围是对图书管理系统数据库中已有书籍的库存或总量数据进行修改。该对话框的实现依旧采用借书证登录界面的逻辑对书号和其名称实现了实时显示功能。（见 6、借书模块设计）并且，在输入书号之后总量和库存的输入框中也会同样显示当前数据库中该书的信息，这个信息可以被修改，修改之后单击确定按钮就可以提交该修改。





提交的实现逻辑与单本入库的部分逻辑相同，取得当前书号作为 update 语句的条件以定位到该条图书记录，对总量和库存的数值做完整性校验，若符合则调用 update 语句对以上两值进行更新。实现代码如下：

```
id = self.lineEdit_id.text()
storage = self.lineEdit_storage.text()
total = self.lineEdit_total.text()
if not total.isdecimal():
    QMessageBox.about(self, '在库书籍调整', '图书总量必须为整数')
    return
if not storage.isdecimal():
    QMessageBox.about(self, '在库书籍调整', '图书库存必须为整数')
    return
if int(total) < int(storage):
    QMessageBox.about(self, '在库书籍调整', '图书总量必须大于库存')
    return
update = database.update_sql('Books', 'Storage = %s, Total = %s' %
(storage, total),
                             "BookNo = '%s'" % id)
self.db.execute(update)
self.db.commit()
```

以上三个功能在对话框结束（accept）之后都会调用 initTable 函数来更新下方的表单以显示最新的图书记录。