浙江大学

数据库系统（实验）

Student Name	刘佳润
Title	实验五：图书管理系统
Mentor	陈岭
Student ID	3180105640
Class	数媒 1801

PART 1：实验要求

1. 设计并实现一个精简的图书管理系统，要求具有图书入库、查询、借书、还书、借书证管理等功能。
2. 通过该图书馆系统的设计与实现，提高学生的系统编程能力，加深对数据库系统原理及应用的理解。

PART 2：系统需求

（1）基本数据对象

对象名称	包含属性
书	书号, 类别, 书名, 出版社, 年份, 作者, 价格, 总藏书量, 库存
借书证	卡号, 姓名, 单位, 类别 (教师 学生等)
管理员	管理员 ID, 密码, 姓名, 联系方式
借书记录	卡号, 借书证号 ,借期, 还期, 经手人 (管理员 ID)

（2）基本功能模块

模块名称	功能描述
管理员登陆	输入管理员 ID, 密码; 登入系统 或 返回 ID/密码 错误.
图书入库	1. 单本入库 2. 批量入库 (方便最后测试) 图书信息存放在文件中, 每条图书信息为一行. 一行中的内容如下 (书号, 类别, 书名, 出版社, 年份, 作者, 价格, 数量) Note: 其中 年份、数量是整数类型; 价格是两位小数类型; 其余为字符串类型 Sample: (book_no_1, Computer Science, Computer Architecture, xxx, 2004, xxx, 90.00, 2)

图书查询	要求可以对书的 类别, 书名, 出版社, 年份(年份区间), 作者, 价格(区间) 进行查询. 每条图书信息包括以下内容: (书号, 类别, 书名, 出版社, 年份, 作者, 价格, 总藏书量, 库存) 可选要求: 可以按用户指定属性对图书信息进行排序. (默认是书名)
借书	1.输入借书证卡号 显示该借书证所有已借书籍 (返回, 格式同查询模块) 2.输入书号 如果该书还有库存，则借书成功，同时库存数减一。 否则输出该书无库存，且输出最近归还的时间。
还书	1.输入借书证卡号 显示该借书证所有已借书籍 (返回, 格式同查询模块) 2.输入书号 如果该书在已借书籍列表内, 则还书成功, 同时库存加一. 否则输出出错信息.
借书证管理	增加或删除一个借书证.

除图书查询功能外，其余功能模块都应该由图书管理员操作。

PART 3：开发环境与开发技术说明

开发环境:

操作系统：Windows 10

主机处理器：Intel® Core™ i5-7300HQ CPU @ 2.50GHz　2.50GHz

系统类型：x64

计算机名：LAPTAP-A0VCDLT2

数据库管理系统：Microsoft SQL Server 2019 (Developer)

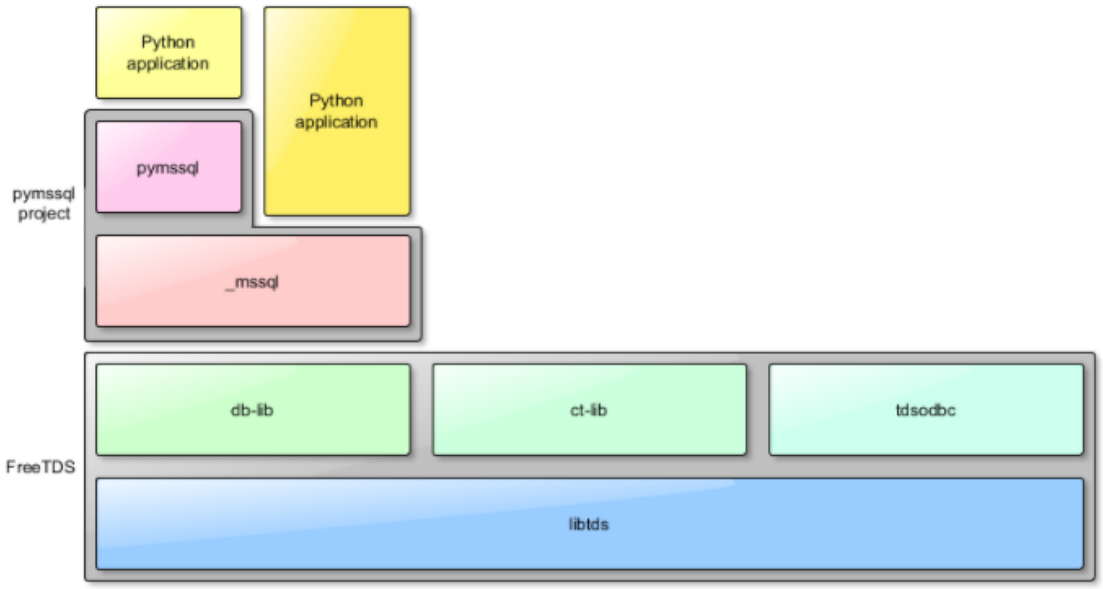
宿主语言选择：Python 3.6

编译环境选择：PyCharm

开发技术:

本实验所用到的开发技术主要是利用 Python 开源数据库连接库 pymssql 与图形开发界面 wxPython。

其中，pymssql 是一个 python 的数据库接口，基于 FreeTDS 构建，对_mssql 模块进行了封装，遵循 python 的 DBAPI 规范，而 FreeTDS 是一个 C 语言连接 sqlserver 的公共开源库。



pymssql 的工作原理可以基本概括如下：

使用 connect 创建连接对象；connect.cursor 创建游标对象，SQL 语句的执行在游标上执行；cursor.execute()方法执行 SQL 语句，cursor.fetch()方法获取查询结果；调用 close 方法关闭游标 cursor 和数据库连接。本工程具体是如何将 SQL 语句与 DBMS 对接的，具体方式会在下面说明。

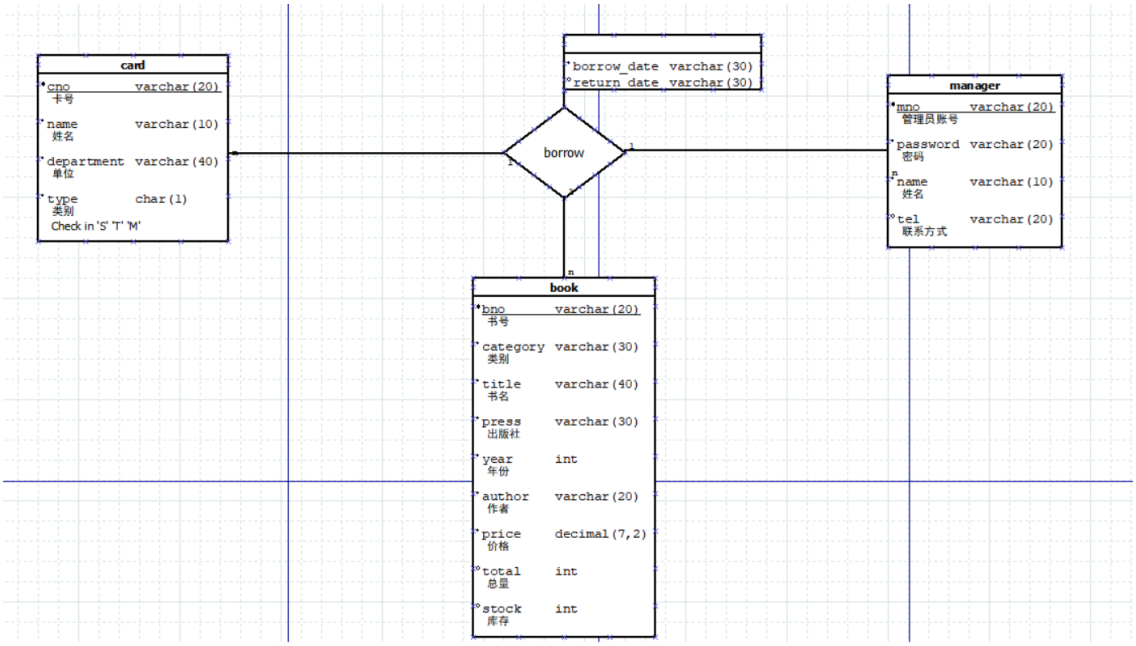
wxPython 是一个开源的 Python GUI 开发库，基于 wxWidgets。wxWidgets 使

用的是标准 C++，与现有各类工具库无缝连接，在不同平台上也是完全 Native GUI，是真正的跨平台。wxWidgets 的主体是由 C++构建的，但你并不是必需通过 C++才能使用它。wxWidgets 拥有许多其它语言的绑定（binding），比如 wxPerl，wxJava，wxBasic，wxJavaScript，wxRuby 等等，wxPython 就是 Python 语言的 wxWidgets 工具库。

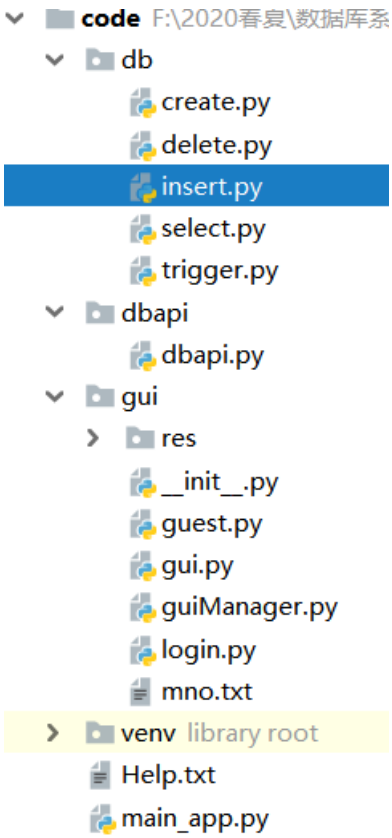
本实验是在 windows 平台开发的，所以同时安装了 pywin32 模块。pywin32 允许你像 VC 一样的使用 python 开发 win32 应用，更重要的是，我们可以用它直接操控 win32 程序，捕捉当前窗口、获取焦点等。

PART 4：系统整体设计

数据库结构设计:



工程结构设计:



db：数据库后端文件

create.py：建表 SQL 语句

delete.py：删除元组 SQL 语句

insert.py：插入与更新 SQL 语句

select.py：查找 SQL 语句

trigger.py：触发器 SQL 语句

dbapi：数据库与 Python 的接口函数，采用了 pymmsql 库

gui：图形界面文件

guest.py：游客界面

gui.py：管理员界面

guiManager.py：界面切换控制

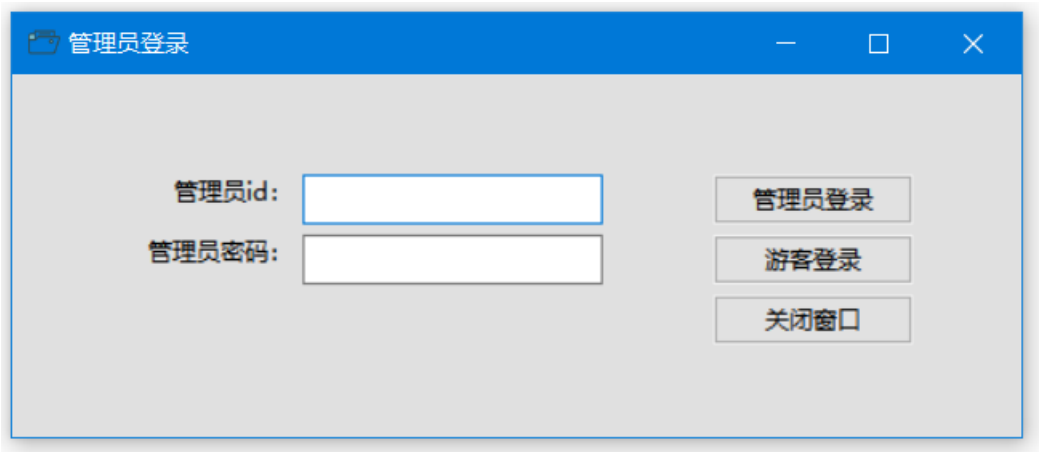
login.py：登录界面

main_db.py：数据库建立时的入口

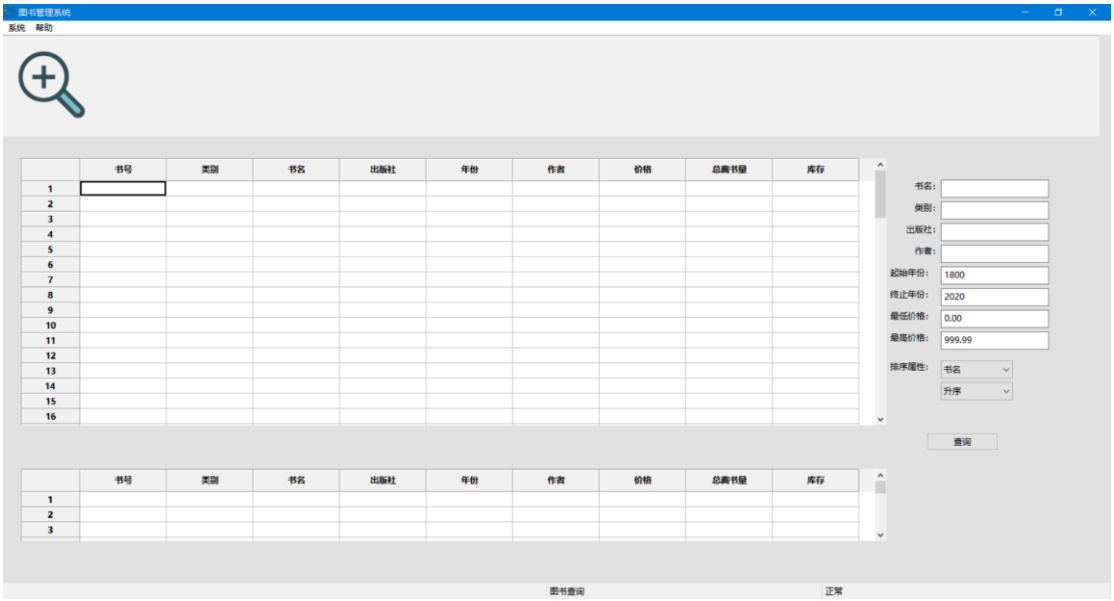
main_app.py：整个系统的入口

图形界面设计:

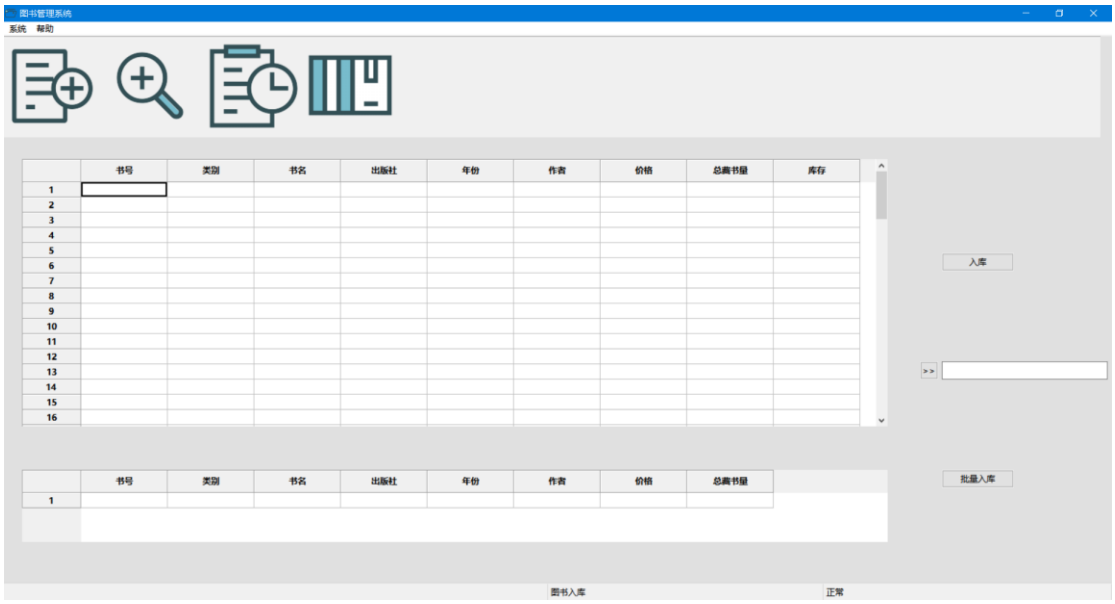
登录界面

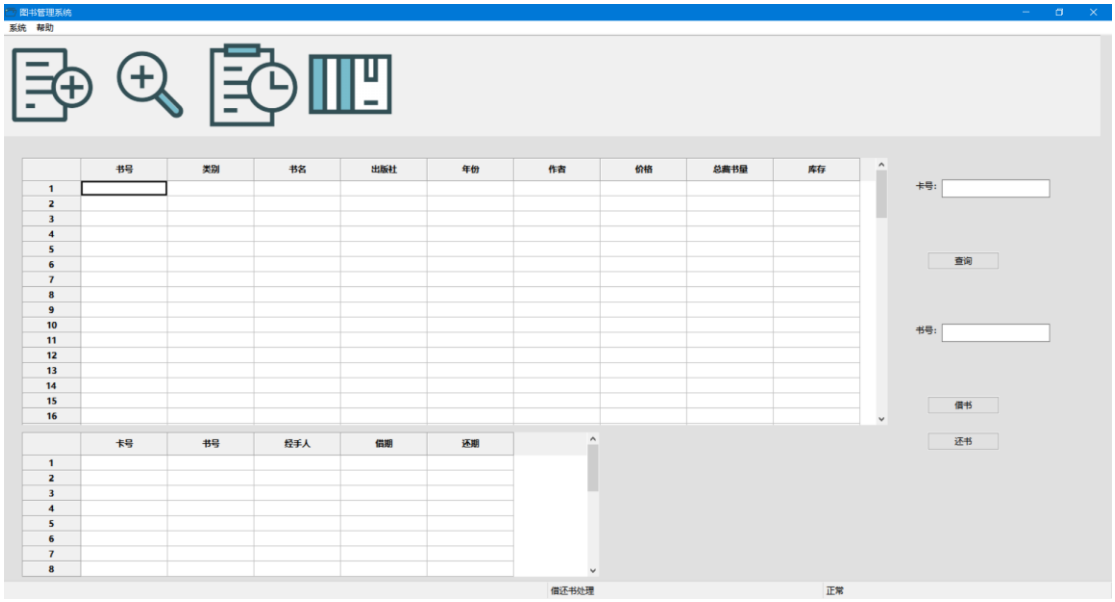


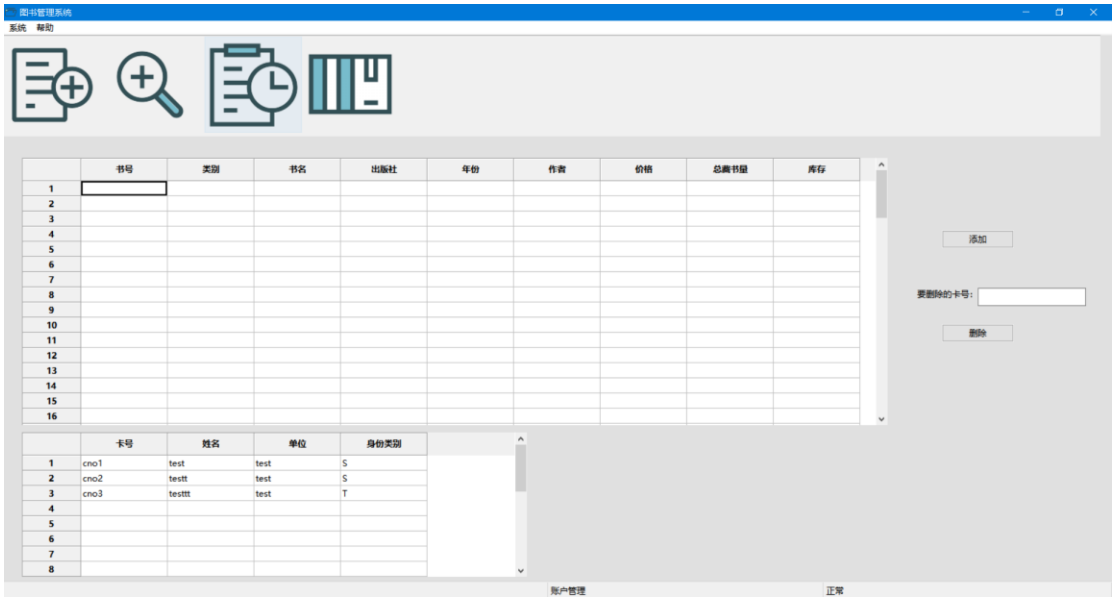
游客界面：只有查询功能



管理员界面：工具栏依次为图书入库、书籍查询、借还书和借书卡管理


图书管理系统
系统 帮助
书号 类别 书名 出版社 年份 作者 价格 总藏书量 库存
入库
>>
书号 类别 书名 出版社 年份 作者 价格 总藏书量
批量入库
图书入库
正常


图书管理系统
系统 帮助
书号 类别 书名 出版社 年份 作者 价格 总藏书量 库存
卡号:
查询
书号:
借书
还书
卡号 书号 经手人 借期 还期
借还书处理
正常


图书管理系统
系统 帮助
书号 类别 书名 出版社 年份 作者 价格 总藏书量 库存
添加
要删除的卡号:
删除
卡号 姓名 单位 身份类别
cno1 test test S
cno2 testt test S
cno3 testtt test T
账户管理
正常

PART 5：细节设计、功能演示与部分具体实现说明

总述：与数据库的连接

使用 pymssql 库——一个基于 Python 系统的可以与 SQL Server 连接。

本工程中，在 dbapi.py 文档中，建立了两个用于处理 SQL 语句的接口。后续对数据库的操作都基于这两个模式。其原理相仿，针对需要调用的是运行状态（成功执行与否）还是返回结果（查询结果）以及是否需要提交操作，有一些细微的差别：

```python
import pymssql

class sqlserverapi:
    def __init__(self,server='localhost',user='sa',password='[redacted]',dbname='bmg'):
        self._server = server
        self._user = user
        self._password = password
        self._dbname = dbname

    def ExcuteDMLSQL(self,sql):
        """
        执行DML操作
        :return:
        """
        try:
            conn = pymssql.connect(self._server, self._user, self._password, self._dbname)
            cursor = conn.cursor()
            cursor.execute(sql)
            conn.commit()
            conn.close()
            return 'SUCCESSED'
        except pymssql.Error as err:
            print(err)
            return 'FAILED'
```

```python
def ExcuteDMLSQLSelect(self,sql):
    """
    执行DML查询操作
    :return:
    """
    try:
        conn = pymssql.connect(self._server, self._user, self._password, self._dbname)
        cursor = conn.cursor()
        cursor.execute(sql)

        data = cursor.fetchall()

        return data
    except pymssql.Error as err:
        return err
```

本实验中所有的对表操作都基于这两个 dbapi 函数。只需要嵌入相应的 SQL 语句（字符串），即可连接到 SQL Server，对表进行增删改查。所以在后面的阐述与展示中，不再具体地贴出代码，而只是在一些比较重要的需要说明实现方法的地方做以说明。例如触发器的编写等等。

总述：图形界面的实现

图形界面的实现使用了 Python 中的 wx 库——一款图形界面开发库。wx 库中为各个控件设置的接口都十分便利。可以直接在布置好图形界面后，对在相应的控件上接上相应的事件函数即可，例如绘制新的表或者执行某个 SQL 语句等等。

多用户登录机制

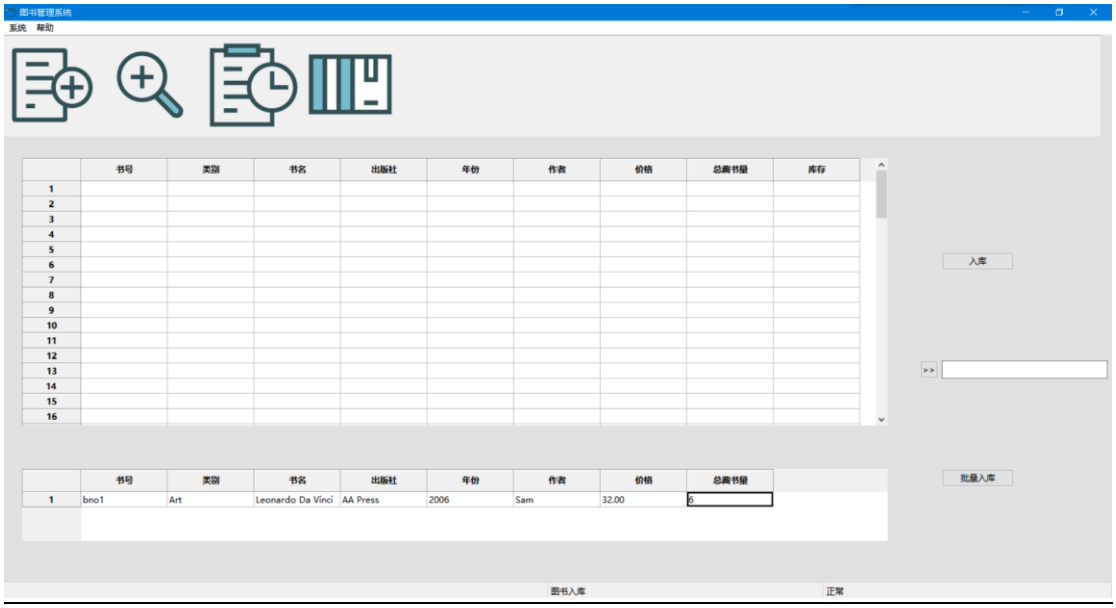
用户分为管理员与游客，其中管理员的信息全部储存在manager表中，该表的操作无法通过前端图形界面实现，只能在DBMS或工程中修改。登录界面中输入管理员账号与对应密码即可进入系统。或者选择游客登录，将会进入另外一个图形界面，只有查询功能。

入库与批量入库

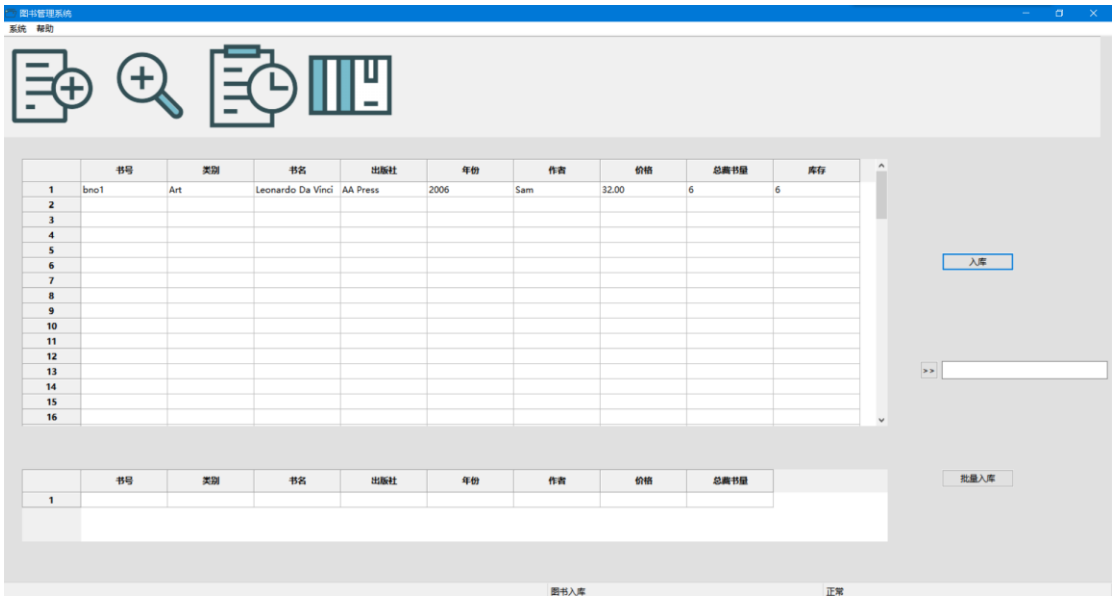
管理员在入库界面可以通过在副表中各栏输入完整的信息以单本入库，或是直接打开编辑好的格式正确的文本文档以批量入库。

单本入库操作：

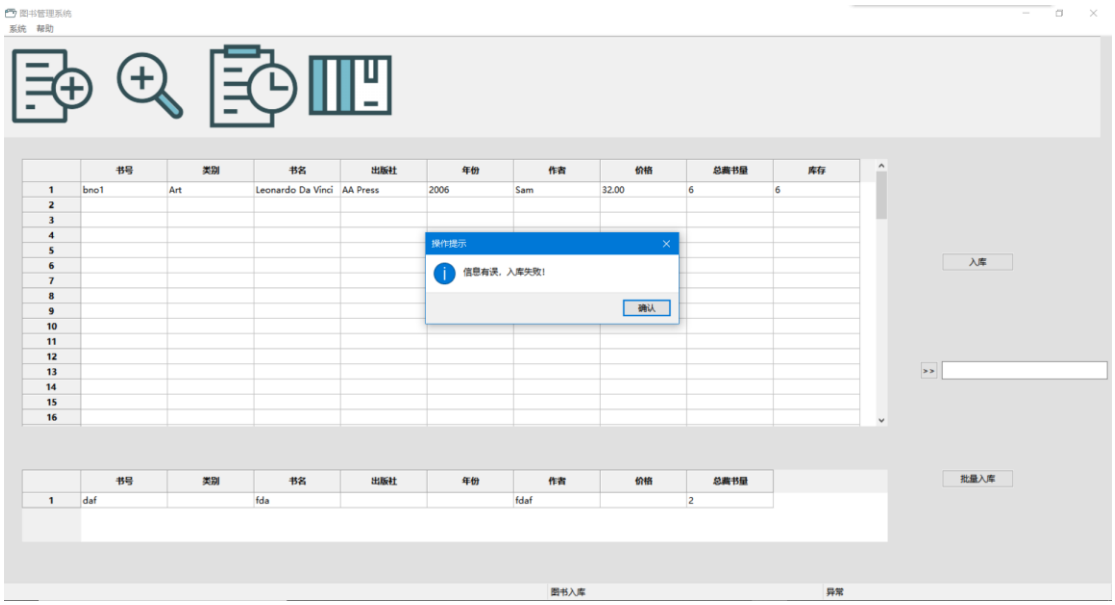
在副表中完整写入书籍信息



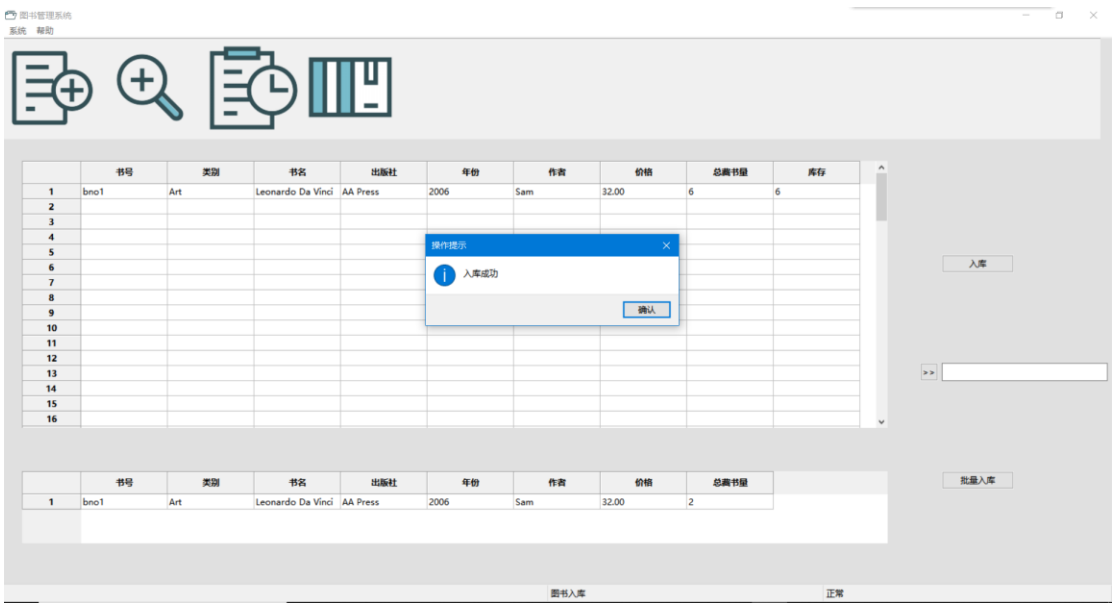
点击入库后，提示“入库成功”，该书本添加到总表。

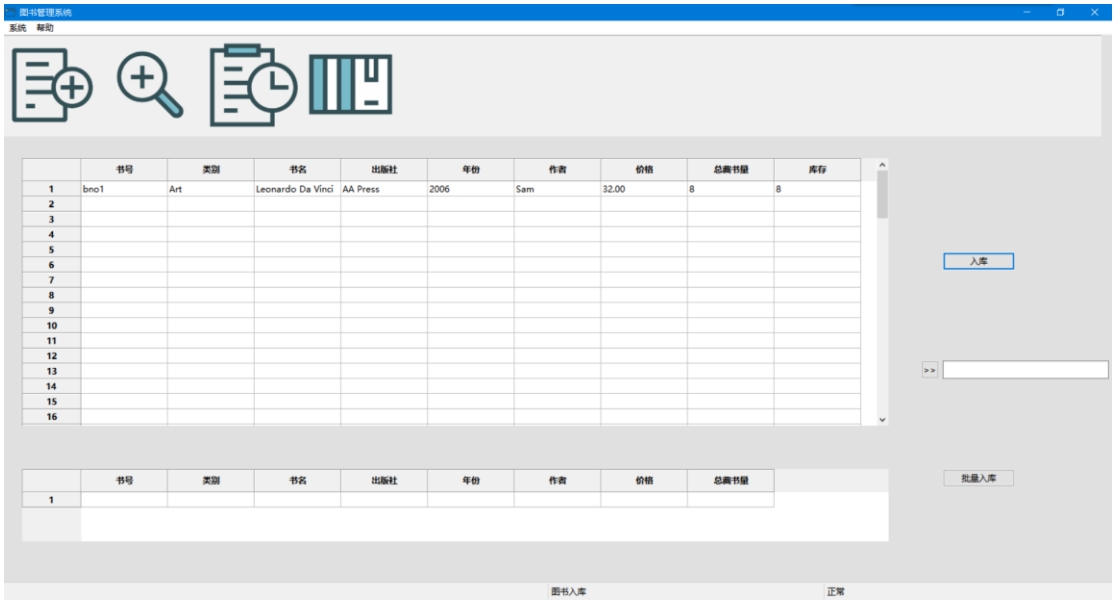


若信息有误（如信息不全或者书号重复），会有相应提示：



同一本书多次添加是可以增加库存的：





关于批量入库，详见视频演示。

具体的实现很简单，读取表中的输入信息作为插入到 SQL 语句中的关键字，然后执行相关的插入或更新 SQL 语句即可。如果出现执行错误，会直接返回错误，回滚插入或更新事务。

查询

具体的演示详见视频演示。

本工程中对于查询，使用的都是模糊查询，即 SQL 中的 LIKE 子句。部分 SQL 语句的编写如下。待查值的获取来自于对图形界面中文本框的字符串读取：

```
sql = "SELECT * FROM book WHERE "
if title != '':
    sql += "title LIKE '%{}%' AND ".format(title)
if category != '':
    sql += "category LIKE '%{}%' AND ".format(category)
if press != '':
    sql += "press LIKE '%{}%' AND ".format(press)
if author != '':
    sql += "author LIKE '%{}%' AND ".format(author)
sql += "year BETWEEN {} AND {} AND ".format(int(year_s), int(year_e))
sql += "price BETWEEN {} AND {} ".format(float(price_s), float(price_e))
sql += "ORDER BY {} ".format(attr)
if method == '升序':
    sql += "ASC"
elif method == '降序':
    sql += "DESC"
out = self._dbapi.ExcuteDMLSQLSelect(sql)
return (out)
```

借书

具体演示详见视频演示。

要说明的一点是，并没有在借还书这里的附表中完全显示所借图书的所有信息，而是只包括了其书号。这是因为图书总表始终显示在工作界面中，为了减少不必要的信息显示。只要用一对一的书号，就可以在总表中很快查找到这本书的全部信息，因为总表是用书号排序的。

在借书中规定了几点：1.同一个卡号不可以同时拥有同一本书超过 1 本，即不可反复借书；2.库存不足时不可以借。这两个的实现是通过触发器实现的。触发器编写的 SQL 语句如下：

```
tr_borrow_check = """
            CREATE TRIGGER tr_borrow_check
            ON borrow
            INSTEAD OF INSERT AS
            IF 0=(SELECT stock FROM inserted, book
                    WHERE inserted.bno=book.bno)
            BEGIN
                RAISERROR('库存不足',16,1)
                ROLLBACK
            END
            ELSE IF
                EXISTS (SELECT * FROM borrow, inserted
                    WHERE borrow.bno=inserted.bno AND borrow.cno=inserted.cno
                    AND borrow.return_date is null)
            BEGIN
                RAISERROR('已经借入',16,1)
                ROLLBACK
            END
            ELSE
            BEGIN
                INSERT INTO borrow SELECT * FROM inserted
            END
            """
```

同时，还要注意在借书之后，书籍的库存需要发生改变。这个实现同样是通过触发器：

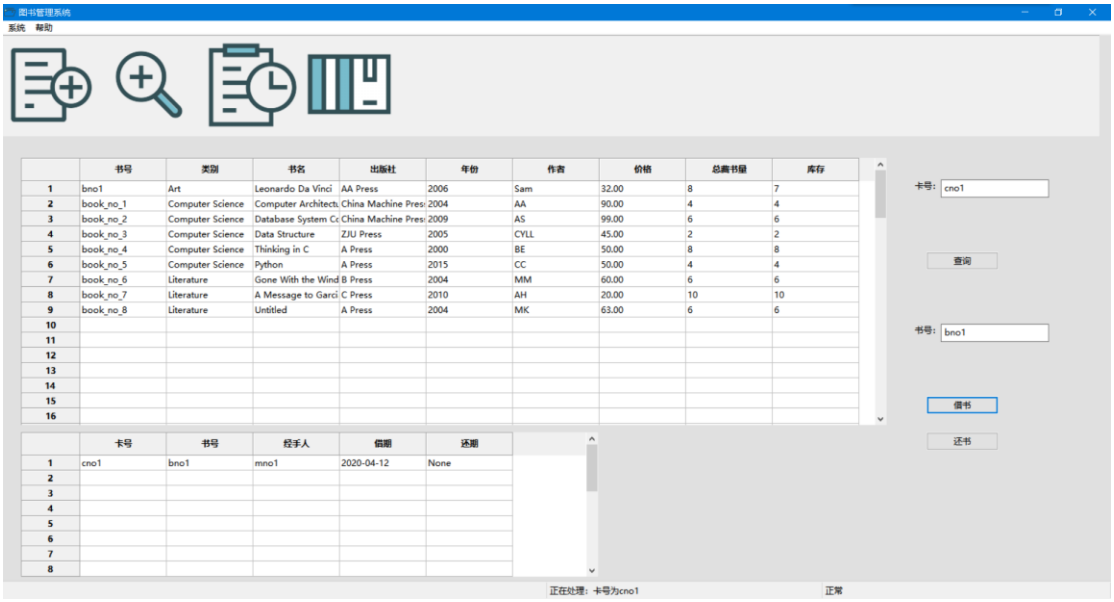
```
def borrow_update(self):
    '''借出数据更新'''
    tr_borrow_update = """
                CREATE TRIGGER tr_borrow_update
                FOR borrow
                AFTER INSERT AS
                BEGIN
                    UPDATE book
                    SET stock = (SELECT stock - 1 FROM book, inserted
                                WHERE book.bno = inserted.bno)
                                WHERE book.bno = (SELECT bno FROM inserted)
                END
                """

    out = self._dbapi.ExcuteDMLSQL(tr_borrow_update)
    print(out)
```

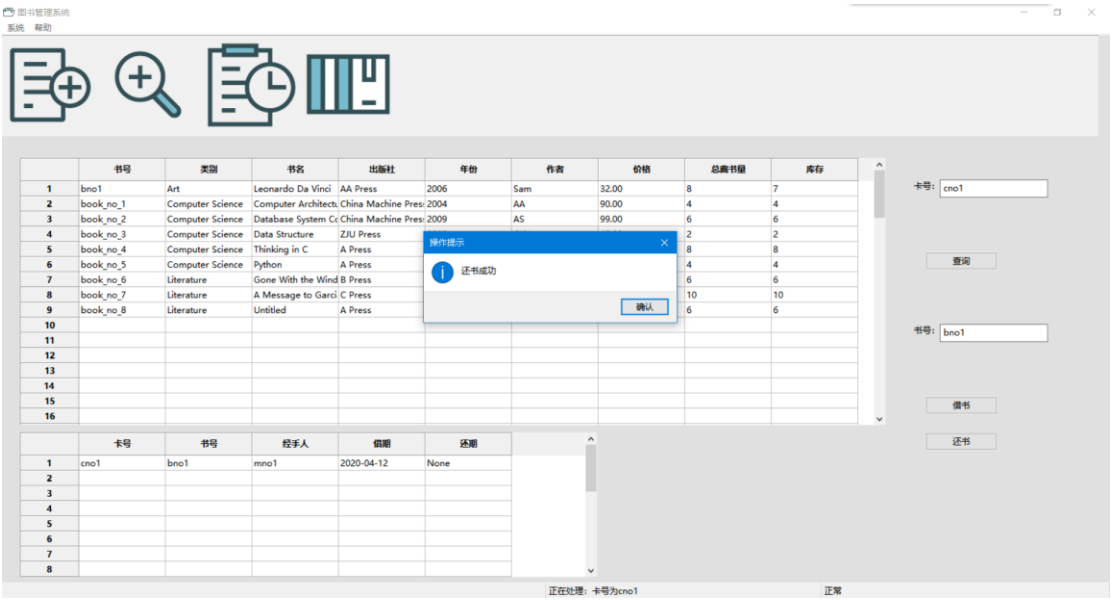
还书

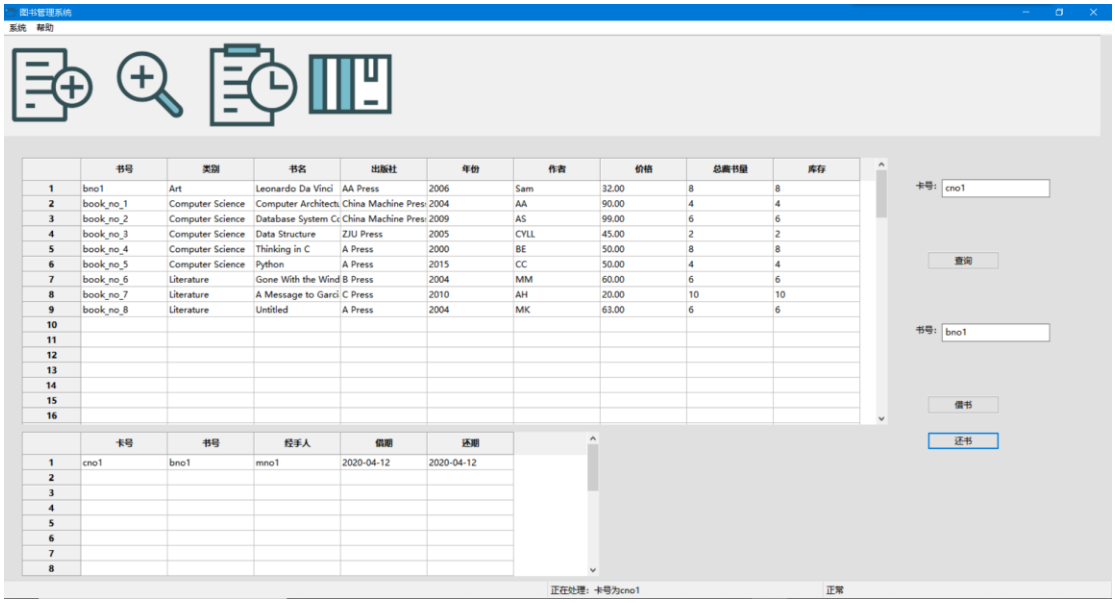
还书操作：

输入卡号，点击“查询”可查看到该卡号的借书记录

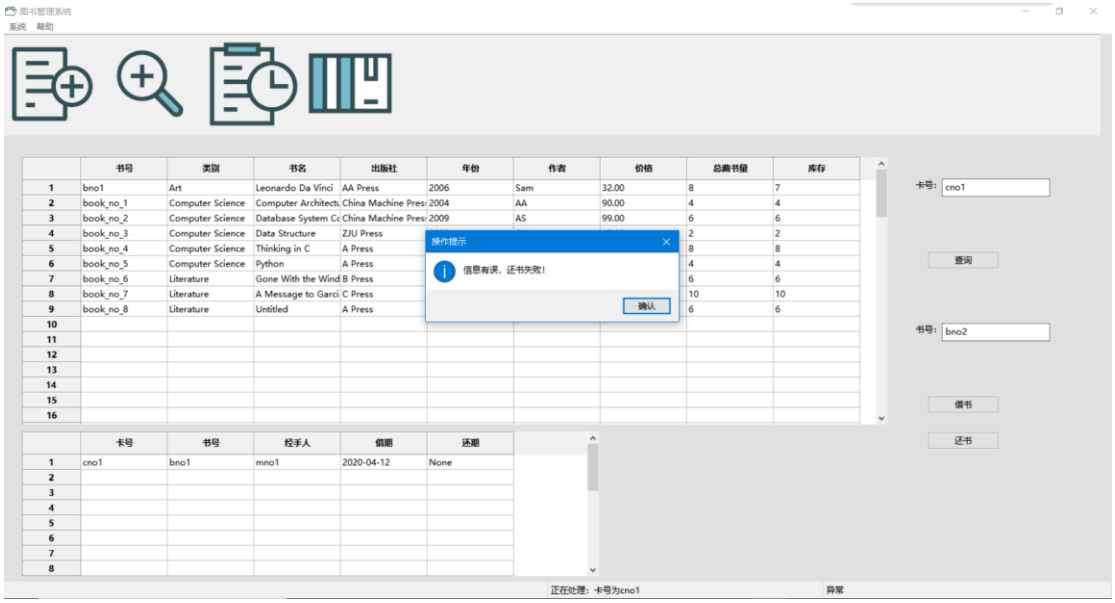


输入要还的书号，点击“还书”完成。确认提示信息后，可以看到借书记录更新，库存更新。





若输入的书号并不存在于该卡号借书记录中，则会有相应提示。



在还书中需要注意的一点是还书后库存的更新。是使用了对 borrow 表的更新触发器。对应的触发器编写如下：

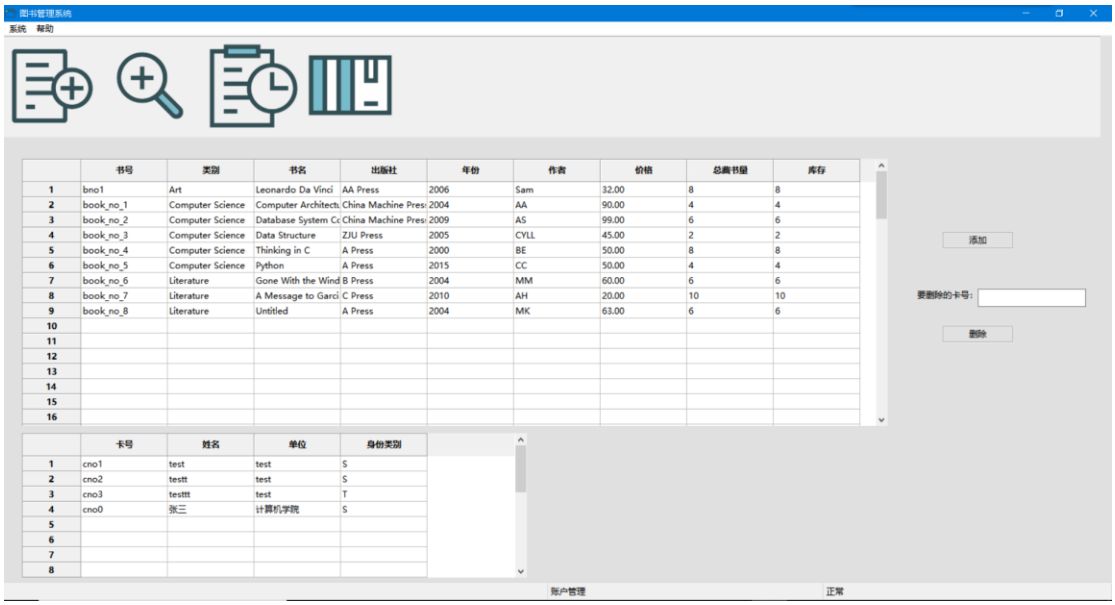
```
def return_update(self):
    '''还书数据更新'''
    tr_return_update = """
                CREATE TRIGGER tr_return_update
                ON borrow
                AFTER UPDATE AS
                IF UPDATE(return_date)
                BEGIN
                    UPDATE book
                    SET stock = (SELECT stock + 1 FROM book, inserted
                                 WHERE book.bno = inserted.bno)
                                 WHERE book.bno = (SELECT bno FROM inserted)
                END
                """

    out = self._dbapi.ExcuteDMLSQL(tr_return_update)
    print(out)
```

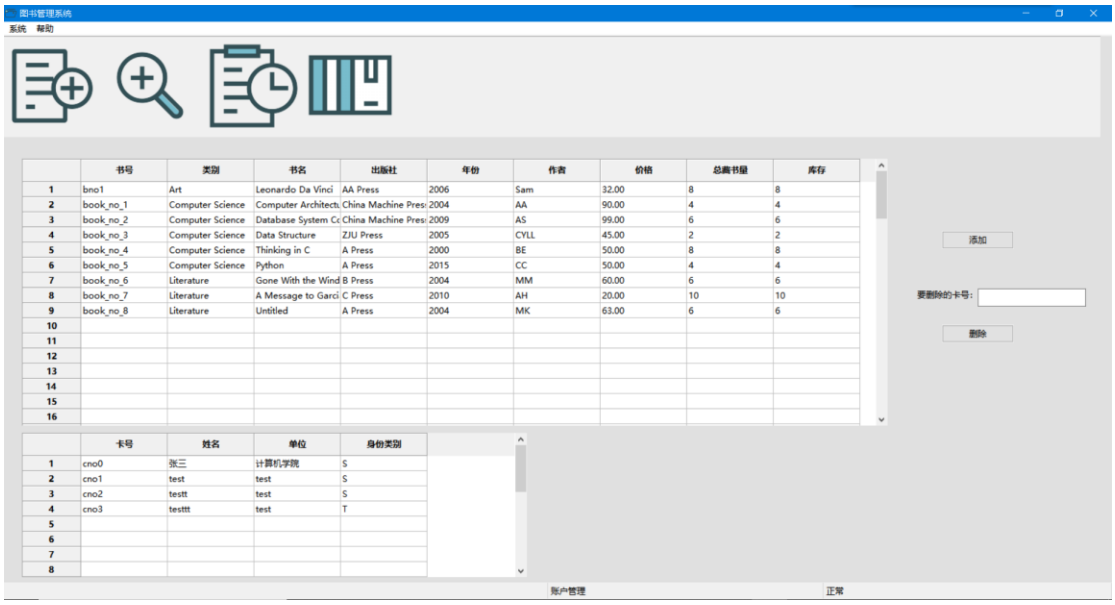
借书证添加与删除

借书证的添加操作：

在副表紧接着下面的一行中输入要添加的卡的信息。



点击“添加”。刷新后即可看见。



借书证的删除操作详见视频演示。

为了保证不要误删有未归还记录的借书证，编写了如下触发器：

```
def delete_card_check(self):
    '''检查注销借书证时该证有无未还记录'''
    tr_del_card = """
            CREATE TRIGGER tr_del_card
            ON card
            INSTEAD OF DELETE AS
            IF EXISTS(SELECT * FROM deleted, borrow
                    WHERE deleted.cno=borrow.cno AND borrow.return_date IS NULL)
            BEGIN
                RAISERROR ('有记录未处理不可注销',16,1)
                ROLLBACK
            END
            ELSE
            BEGIN
                DELETE FROM card WHERE cno=(SELECT cno FROM deleted)
            END
            """

    out = self._dbapi.ExcuteDMLSQL(tr_del_card)
    print(out)
```

图形界面的一些设计细节

菜单栏：

状态栏：

这些都是利用 wx 库中的相应类所制作出来的，具体实现在报告中略去不表。整个图形界面设计的原则是尽量简单以及减少那些不必要的弹出窗口，所以把图书总表整个都始终呈现在工作界面中。根据功能的不同，会出现不同的副表。

PART 6: 总结与反思

通过本实验的学习和训练，我认为自己对于数据库的操作以及 SQL 语句的编写基本上掌握了，而且对于数据库的整体设计也有了一定的思路和认识，即把需求转换为设计与模式。

还有一些需要改进的地方，这些集中在图形界面——我认为 wxPython 这个库还是有一些做的不到位的地方，在界面更新等地方，由于偏重考虑 MFC 程序的跨平台迁移，wxWidgets 面向对象封装做得差强人意。。有机会的话可以了解一下 PyQt。Qt 目前已经不存在兼容性问题了。Qt 库也是所有的 GUI 工具库中最为面向对象化的，同时也是最为稳定的。也许在做 MiniSQL 的时候会考虑用 Qt 编程。

另外本程序所连接的数据库仅限于本地，这也是考虑到这个实验的实际意义——图书管理系统对于远程操作的需求不高。但是如果加以改进的话，可以在工程中添加数据库创建的模块，对于一台新的电脑也可以运行。或者连接到其他数据库，设置这样一个端口。

最后，Python 的打包非常的变态，试验了很久没有成功打包成 exe。暂且通过 PyCharm 来运行目标文件，也可以跑起来吧。