实验 5 数据库程序设计

赵前程　3140102489　竺可桢学院混合班

一、 实验目的

1. 设计并实现一个精简的图书管理系统，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。

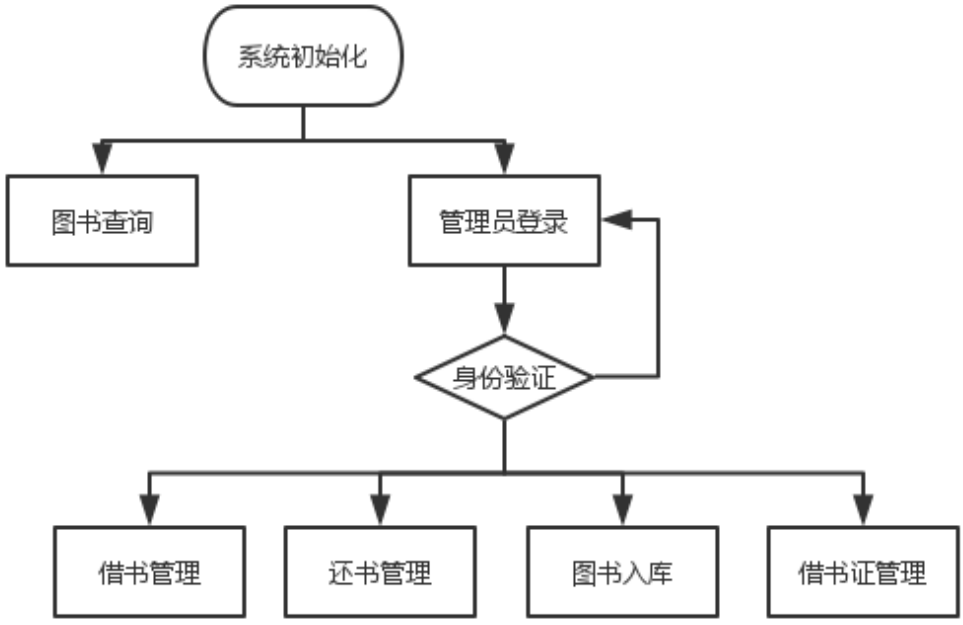
二、 实验平台

开发工具：Qt；数据库平台：MySQL

三、 总体设计

1. 系统架构描述

本系统主要包括管理员登录、图书入库、借书管理、还书管理、借书证管理六大功能模块。系统处理基本流程如下：



系统初始时仅有图书查询和管理员登录两个选项，同时界面左下角提示当前登录用户为“not login”。图书查询为公共功能模块，不需要登录也可以操作。管理员登录成功后，便可以进入图书入库、借书管理、还书管理、借书证管理功能。

各个功能模块说明如下：

模块名称	功能描述
管理员登录	根据管理员的 ID 和密码登录系统。ID 和密码均保存在数据库中
图书查询	1. 按书的类别、书名、出版社、年份、作者、价格进行查询 2. 支持模糊查询，例如输入书的部分书名 3. 查询内容可以为空

借书管理	1. 输入借书证卡号，自动显示该借书证所有已借书籍 2. 输入书号，如果该书还有库存，则借书成功，同时库存减 1，否则提示该书无库存
还书管理	1. 输入借书证卡号，自动显示该借书证所有已借书籍 2. 输入书号，如果该书在借书列表内且为归还，则还书成功，同时库存加 1，否则输出错误信息
图书入库	1. 根据图书信息单本入库，直接从程序界面上输入 2. 批量入库，从文本文件中批量导入图书数据
借书证管理	增加或产出借书证

2. 数据库表设计

图书信息表(Books)

字段名	数据类型	主键	说明
BookNo	Nvarchar(50)	Yes	书号
BookType	Nvarchar(50)		图书类别
BookName	Nvarchar(50)		书名
Publisher	Nvarchar(50)		出版社
Year	Int		出版年份
Author	Nvarchar(50)		作者
Price	Numeric(6,2)		图书单价
Total	int		总藏书量
Storage	Int		库存数
UpdateTime	Datetime		添加时间

借书证表(LibraryCard)

字段名	数据类型	主键	说明
CardNo	Nvarchar(50)	Yes	卡号
Nmae	Nvarchar(50)		姓名
Department	Nvarchar(50)		单位
CardType	Nvarchar(50)		卡类别
UpdateTime	Datetime		添加时间

管理员表(Admins)

字段名	数据类型	主键	说明
AdminID	Nvarchar(50)	Yes	用户名
PassWord	Nvarchar(50)		密码
Name	Nvarchar(50)		姓名
Contact	Nvarchar(50)		联系方式

借书记录表(LibraryRecords)

字段名	数据类型	主键	说明
FID	Int	Yes	标识列
CardNo	Nvarchar(50)		借书卡号

BookNo	Nvarchar(50)		书号
LentDate	Datetime		借书日期
ReturnDate	Datetime		还书日期
Operator	Nvarchar(50)		经手人

3. 所用开发技术

我使用的数据库系统为 MySQL。作为一个免费的数据库系统，它支持大多数的数据库操作，且为程序提供了方便的接口。使用 MySQL WorkBench 图形用户界面使用数据库，使得创建表、添加约束更加方便，同时查询结果更加直观。

我使用的软件开发工具是 QT Creator。这是一款跨平台的 C++图形用户界面应用程序框架，它提供给应用程序开发者建立图形用户界面所需的所有功能。使用 QT 可以直观地绘制出所需的图形用户界面，且很多图形用户功能可以直接使用，非常方便。值得一提的是 QT 中内置了访问数据库的类，通过它访问数据库、查询处理数据非常方便有效。

四、 详细设计

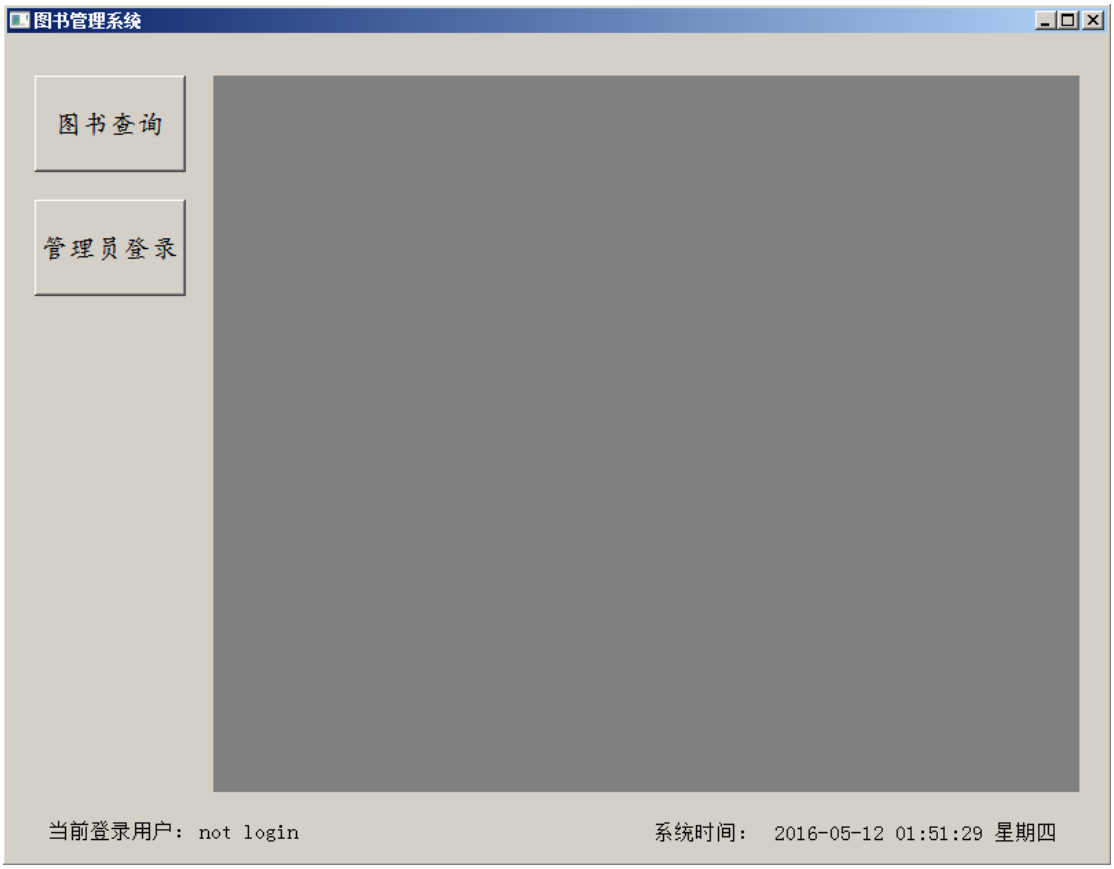
1. 连接数据库

按照 Qt 软件常用的连接数据库方法，连接数据库的函数 createConnction 作为静态函数被写在头文件 connection 中，它创建一个 QSqlDatabase 类的对象，通过这个对象来连接数据库。connection.h 代码如下

```
#ifndef CONNECTION_H
#define CONNECTION_H
#include <QMessageBox>
#include <QSqlDatabase>
static bool createConnection(){
    QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");
    db.setHostName("localhost");
    db.setPort(3306);
    db.setDatabaseName("sys");
    db.setUserName("root");
    db.setPassword("password");//password is password
    if(!db.open())
        return false;
    else
        return true;
}
#endif // CONNECTION_H
```

2. 主界面设计

程序主界面由三部分组成：导航栏、任务栏、状态栏。主界面的功能集中在导航栏，点击导航栏中的图标，任务栏就会出现对应功能的子窗口标签，各个任务在子窗口中完成。状态栏位于主界面底部，显示当前的用户登录状态和系统时间。



具体实现使用了 Qt 的 QWidget 控件，将一个 QWidget 对象作为主窗口，在其中加入 Label, PushButton, MdiArea 等控件，并对这些控件进行相应设置，如添加信号和槽等。

3. 图书查询模块

图书查询不需要管理员登录，在系统初始化完成后可直接开始使用，其基本工作流程是：输入查询条件→组合查询条件→执行查询→返回结果。

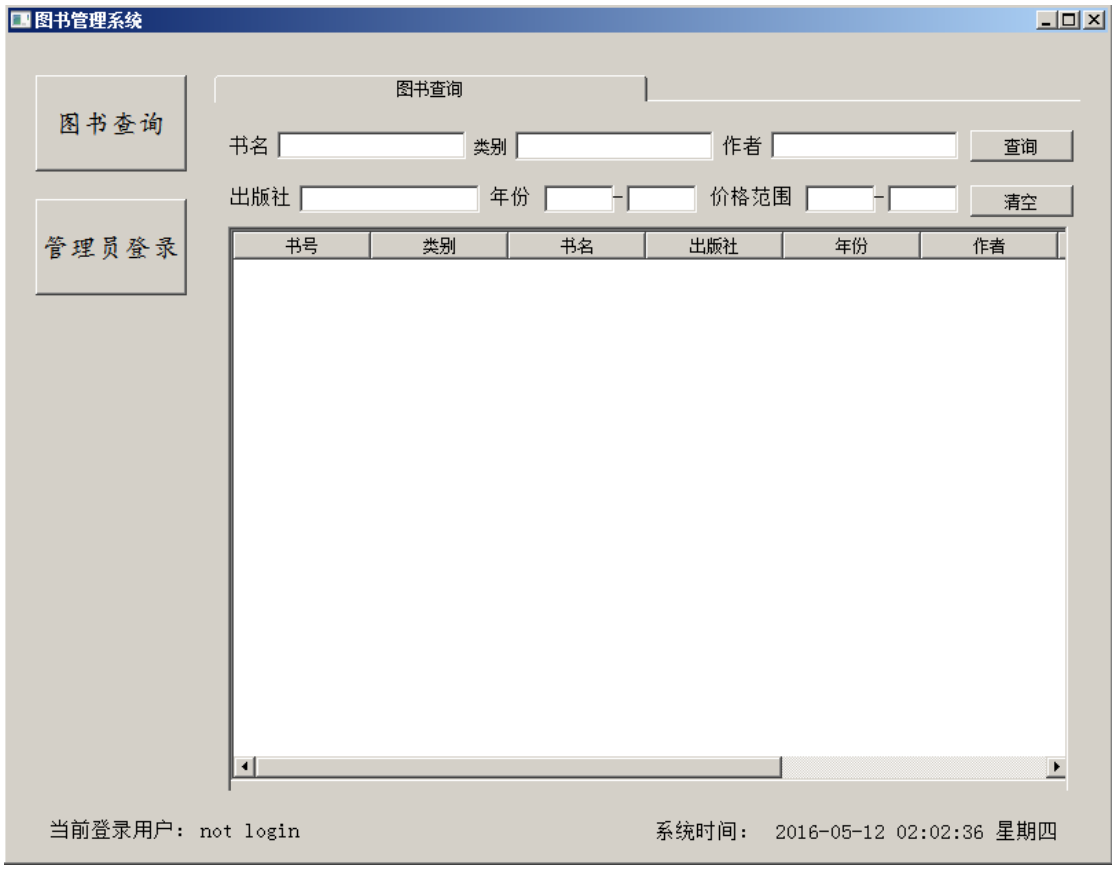
在查询图书时，有可能某个查询条件为空，或者查询的书名不完整。我在查询时使用了一些小技巧，可以用一个 selec 语句完成查询。

如果某一个查询条件不为空，如'String'，则将该字符串前后加一'%'，如'%String%'；如果查询条件为空，则将该条件的字符串设置为'null'。具体的查询语句如下：

```
QSqlQuery query;
    query.prepare("select
BookNo,BookType,BookName,Publisher,Year,Author,Price,Total,Storage "
                  "from Books where "
                  "(BookName like case when :BookName='null' then BookName
else :BookName end) and "
                  "(BookType like case when :BookType='null' then BookType
else :BookType end) and "
                  "(Author like case when :Author='null' then Author
else :Author end) and "
                  "(Publisher like case when :Publisher='null' then
Publisher else :Publisher end) and "
                  "(Price >= :Price1) and "
                  "(Price <= case when :Price2 = 0 then Price else :Price2
end) and "
                  "(Year >= :Year1) and "
```

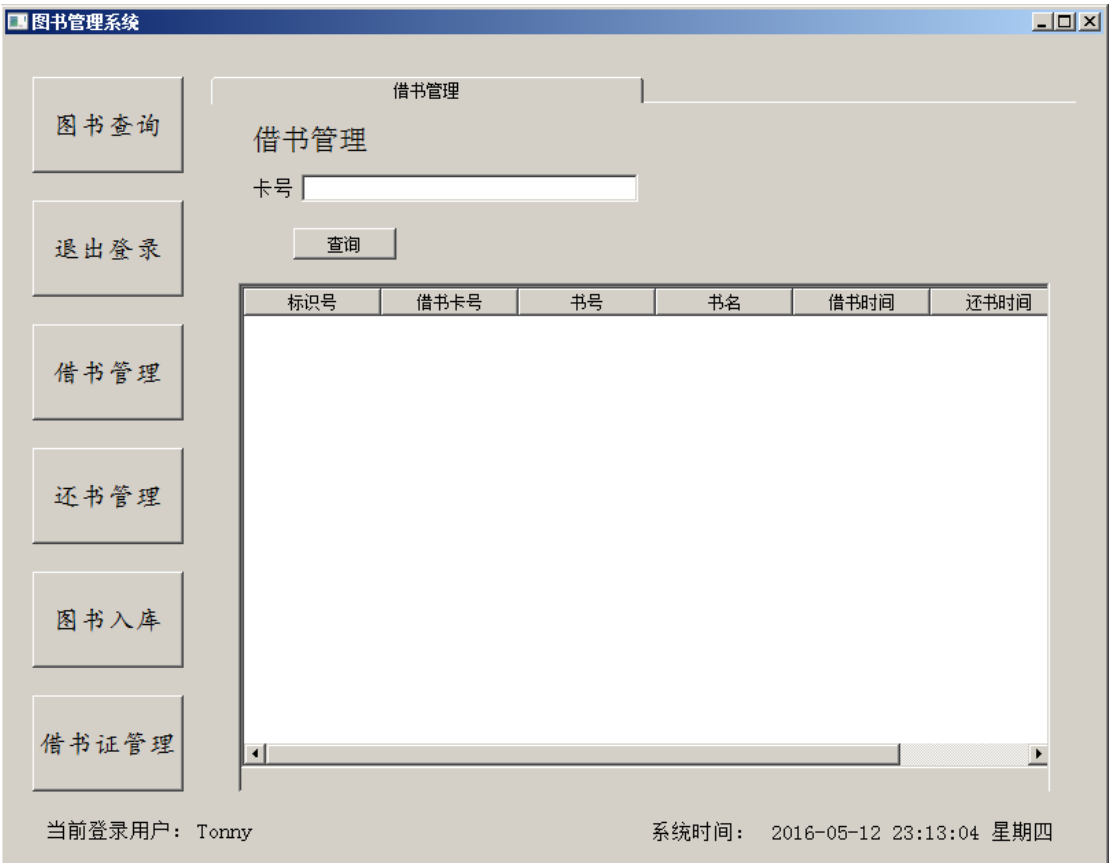
```
                "(Year <= case when :Year2 = 0 then Year else :Year2 end)
"
                "order by BookNo;");
    query.bindValue(":BookName", BookName);
    query.bindValue(":BookType", BookType);
    query.bindValue(":Author", BookAuthor);
    query.bindValue(":Publisher", BookPublisher);
    query.bindValue(":Price1", price1);
    query.bindValue(":Price2", price2);
    query.bindValue(":Year1", year1);
    query.bindValue(":Year2", year2);
    query.exec();
```

界面如下：



4. 借书管理模块

本模块主要实现了查询对应借书证借书记录和借书登记功能。当管理员在借书证卡号里录入数据时，会自动执行查询，如果借书证卡号存在，将在下面的表格中自动显示出该借书证所借的图书信息。录入书号后，点击借书按钮，将把该书的借出信息添加到 LibraryRecords 表中，同时 Books 表中的库存自动减 1。界面如下：



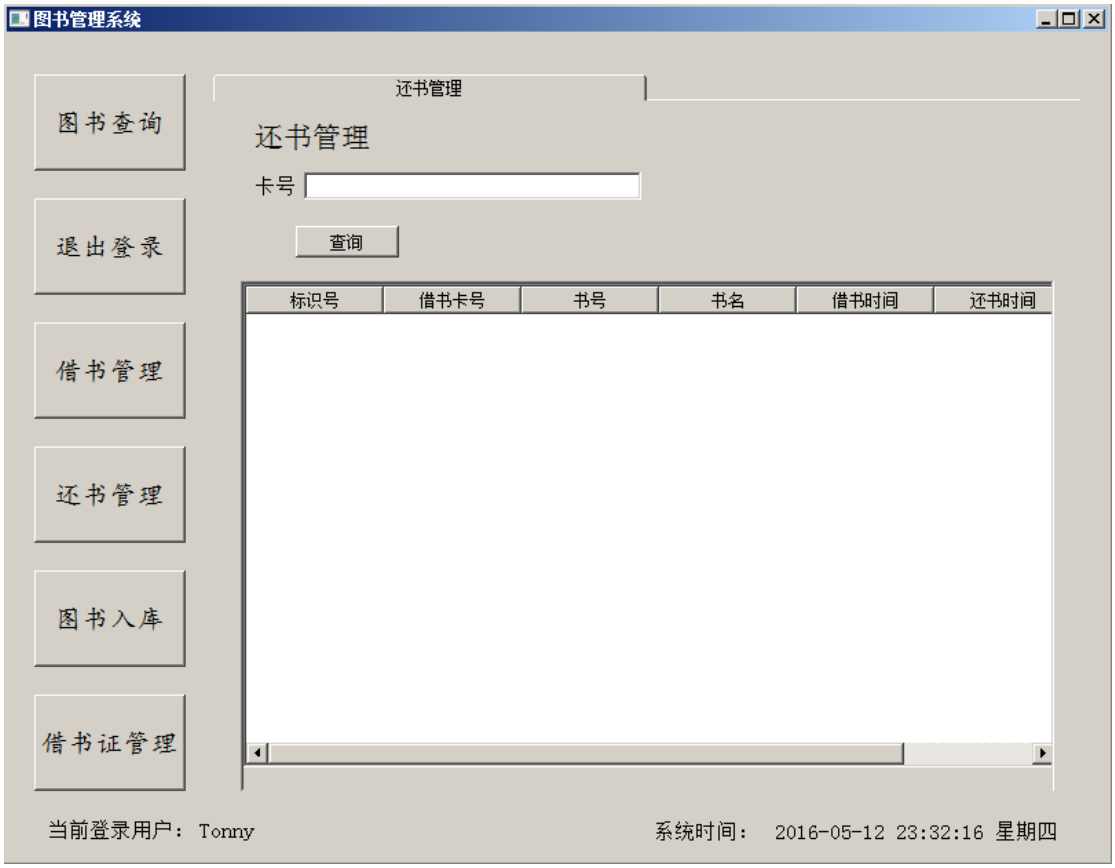
上图为输入借书证卡号前的界面，输入借书证卡号，查询借书记录。

在输入借书证卡号之前，不显示还书选项，只有在输入了有效的借书证卡号后，才会显

示借书的选项，同时显示该借书证的借书记录。

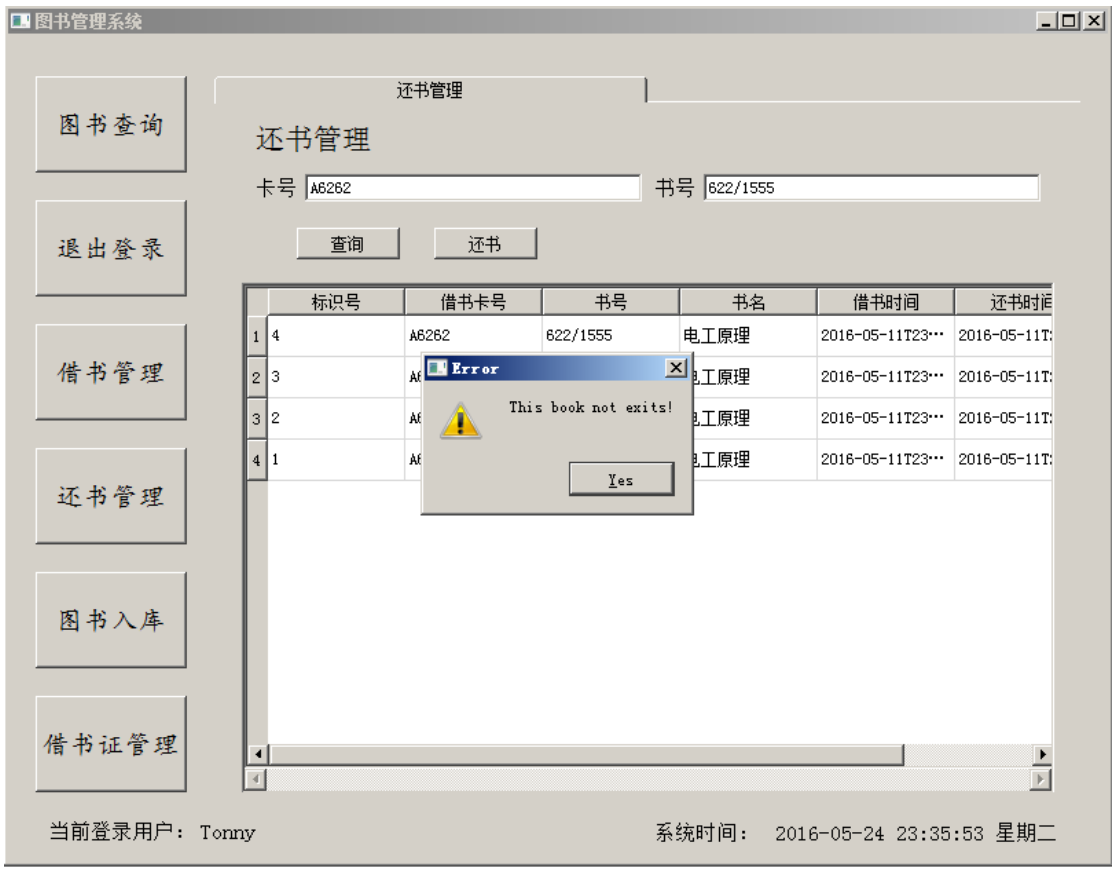
5. 还书管理模块

本模块主要实现了查询对应借书证借书记录和还书登记功能。当管理员在借书证卡号里录入数据时，会自动执行查询，如果借书证卡号存在，将把该书的还书信息更新到 LibraryRecords 表中，同时 Books 表中的库存里自动加 1.如果并未借出指定的图书，则提示没有借出此图书。界面如图：



上图为没有输入卡号的时候，这时不显示还书选项。输入正确的卡号后，对该借书证的借书记录进行查询显示，并出现还书选项。如图所示：

接着输入要还书的书号，点击还书即可，如果没有借过次书或已经归还，则返回一个错误，提示操作不成功。



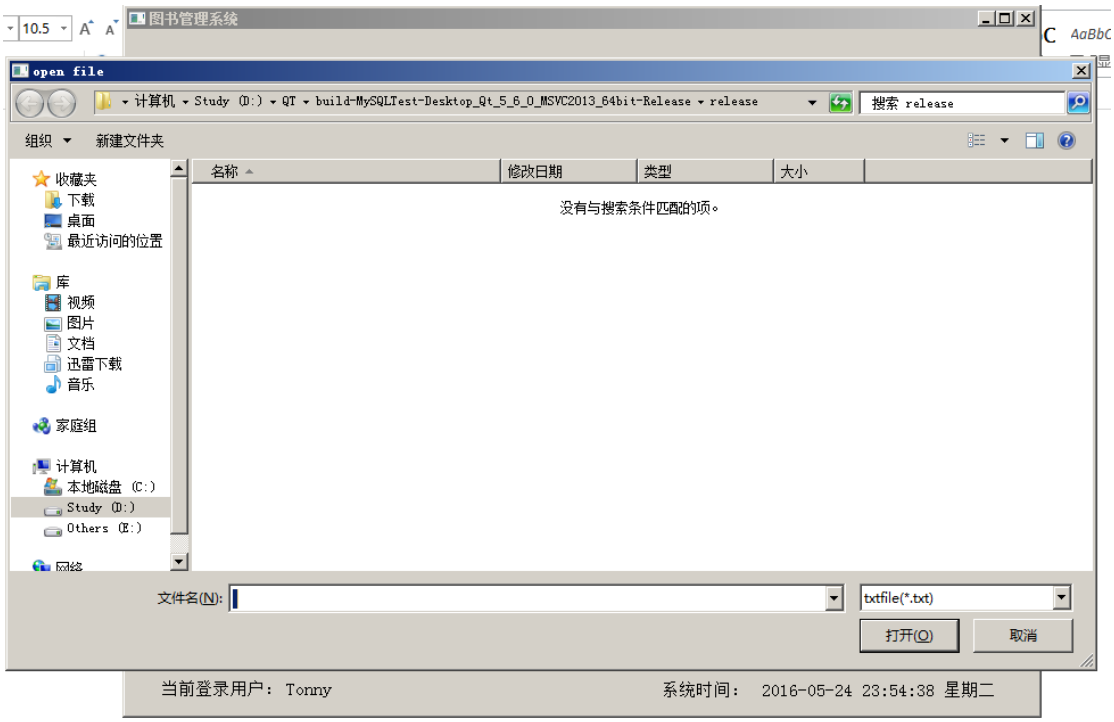
这里的检测机制是，如果借出一本书，则借书日期为当前的系统日期，还书日期为'0000-00-00 00:00:00'，当点下还书按钮时，检测目标书号对应的记录的还书日期是否有效。如果此书未被归还，则该记录的还书时间更新为现在的系统时间，且该书库存+1。实现代码如下：

```
void ReturnBook::on_ReturnBtn_clicked() //当还书按钮被点下
{
    QString BookNo = ui->BookNoEdit->text();
    QSqlQuery query;
    query.prepare("select * from libraryrecords where "
                  "bookNo = :BookNo and returndate = '0000-00-00 00:00:00';");
    query.bindValue(":BookNo", BookNo);
    query.exec();
    if(query.size() > 0){
        query.prepare("update libraryrecords set returndate = :date where "
              "bookNo = :BookNo and returndate = '0000-00-00 00:00:00';");
        QDateTime time = QDateTime::currentDateTime();
        QString date = time.toString("yyyy-MM-dd hh:mm:ss");
        query.bindValue(":date", date);
        query.bindValue(":BookNo", BookNo);
        query.exec();
        query.prepare("update books set storage = storage + 1 where "
                      "bookNo = :BookNo;");
        query.bindValue(":BookNo", BookNo);
        query.exec();
        on_QueryBtn_clicked();
    }
    else
        QMessageBox::warning(this,tr("Error"),
                             tr("This book not exits!"),
                             QMessageBox::Yes);
}
```

6. 图书入库模块

本模块主要实现了图书信息的录入，以及批量导入功能。管理员可以在软件界面上逐条录入图书的各项信息，然后点击添加按钮，就会把当前录入的图书信息保存到数据库中(如果录入的图书已有记录,则将它的库存数添加输入的数目,其他输入项目不管)。如果要导入大量数据，则可以从文本文件中导入，格式是：书号，类别，书名，出版社，出版年份，作者，图书单价，总藏书量，库存数。每项之间用英文逗号分隔开来，每行一条记录。界面如图：

选择对应的行点击删除键可以删除一个图书记录，点击导入按钮可以选择要导入的文本文件：



实现数据批量导入的代码如下所示：

```
void AddBook::on_ImportBtn_clicked() //当导入键被按下
{
```

```
    QString fileName = QFileDialog::getOpenFileName(this, tr("open
file"), " ",  tr("txtfile(*.txt)"));
    QFile file(fileName);
    if(file.open(QIODevice::ReadOnly | QIODevice::Text)){
        QString Values;
        QString
BookNo,BookType,BookName,Publisher,YearString,Author,PriceString,Nu
mString;
        int Year, Num;
        double Price;
        int i;
        while(!file.atEnd()){
            Values = QString::fromLocal8Bit(file.readLine(0));
            for(i = 0, BookNo.clear(); Values[i] != ','; i++)
                BookNo += Values[i];
            for(i++, BookType.clear(); Values[i] != ','; i++)
                BookType += Values[i];
            for(i++, BookName.clear(); Values[i] != ','; i++)
                BookName += Values[i];
            for(i++, Publisher.clear(); Values[i] != ','; i++)
                Publisher += Values[i];
            for(i++, YearString.clear(); Values[i] != ','; i++)
                YearString += Values[i];
            for(i++, Author.clear(); Values[i] != ','; i++)
                Author += Values[i];
            for(i++, PriceString.clear(); Values[i] != ','; i++)
                PriceString += Values[i];
            for(i++, NumString.clear(); Values[i] != '\n'; i++)
                NumString += Values[i];
            Year = YearString.toInt();
            Num = NumString.toInt();
            Price = PriceString.toDouble();

            QSqlQuery query;
            query.prepare("select * from books where bookNo
= :BookNo;");
            query.bindValue(":BookNo", BookNo);
            query.exec();
            if(query.size() > 0){
                query.prepare("update books set total = total
+ :Num, storage = storage + :Num "
                              "where bookNo = :BookNo;");
                query.bindValue(":Num", Num);
                query.bindValue(":BookNo", BookNo);
```

```
                if(!query.exec())
                    QMessageBox::warning(this,tr("Error"),
                                         tr("Add book failed!"),
                                         QMessageBox::Yes);
            }
            else{
                QDateTime time = QDateTime::currentDateTime();
                QString Date = time.toString("yyyy-MM-dd
hh:mm:ss");
                query.prepare("insert into books "

"values(:BookNo,:BookType,:BookName,:Publisher,:Year,:Author,:Price
,:Total,:Storage,:Date);");
                query.bindValue(":BookNo", BookNo);
                query.bindValue(":BookType", BookType);
                query.bindValue(":BookName", BookName);
                query.bindValue(":Publisher", Publisher);
                query.bindValue(":Year", Year);
                query.bindValue(":Author", Author);
                query.bindValue(":Price", Price);
                query.bindValue(":Total", Num);
                query.bindValue(":Storage", Num);
                query.bindValue(":Date", Date);
                query.exec();
            }
        }
    }
    else
        QMessageBox::warning(this,tr("Error"),
                             tr("Can't find the file!"),
                             QMessageBox::Yes);
    QueryBook();
}
```

7. 借书证管理

本模块实现了借书证的添加和删除。当输入数据后，首先对数据完整性进行校验，通过后再把数据插入到数据库中。选中某一行，点击删除按钮，即可删除一个借书证。界面如下：

实现代码如下：

```
void CardManage::on_QueryBtn_clicked()
{
    QString CardNo = ui->CardNoEdit->text();
    QString Name = ui->NameEdit->text();
    QString Dept = ui->DeptEdit->text();
    QString Type = ui->TypecomboBox->currentText();
    if(CardNo == NULL) CardNo = "null";
    if(Name == NULL) Name = "null";
    if(Dept == NULL) Dept = "null";
    QSqlQuery query;
    query.prepare("select cardNo,name,department,cardtype from librarycard where
"
                  "(cardNo = case when :CardNo = 'null' then cardNo else :CardNo
end) and "
                  "(name = case when :Name = 'null' then name else :Name end)
and "
                  "(department = case when :Dept = 'null' then department
else :Dept end) and "
                  "(cardtype = :Type);");
    query.bindValue(":CardNo", CardNo);
    query.bindValue(":Name", Name);
    query.bindValue(":Dept", Dept);
    query.bindValue(":Type", Type);
    query.exec();
    ui->tableWidget->setRowCount(query.size());
    ui->tableWidget->setEditTriggers(QAbstractItemView::NoEditTriggers);
    ui->tableWidget->setSelectionBehavior(QAbstractItemView::SelectRows);
```

```
    ui->scrollArea->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOn);
    ui->scrollArea->setVerticalScrollBarPolicy(Qt::ScrollBarAsNeeded);
    int row = 0, col = 0;
    while(query.next()){
        for(col = 0; col < 4; col++)
            ui->tableWidget->setItem(row, col, new
QTableWidgetItem(query.value(col).toString()));
        row++;
    }
}

void CardManage::on_AddBtn_clicked()
{
    QString CardNo = ui->CardNoEdit->text();
    QString Name = ui->NameEdit->text();
    QString Dept = ui->DeptEdit->text();
    QString Type = ui->TypecomboBox->currentText();
    if(CardNo!=NULL && Name!=NULL && Dept!=NULL){
        QSqlQuery query;
        query.prepare("select * "
                      "from librarycard "
                      "where cardNo = :CardNo;");
        query.bindValue(":CardNo", CardNo);
        query.exec();
        if(query.size() > 0)
            QMessageBox::warning(this,tr("Error"),
                                 tr("The card ID has existed!"),
                                 QMessageBox::Yes);
        else{
            QDateTime time = QDateTime::currentDateTime();
            QString date = time.toString("yyyy-MM-dd hh:mm:ss");
            query.prepare("insert into librarycard "
                          "values(:CardNo,:Name,:Dept,:Type,:Date);");
            query.bindValue(":CardNo", CardNo);
            query.bindValue(":Name", Name);
            query.bindValue(":Dept", Dept);
            query.bindValue(":Type", Type);
            query.bindValue(":Date", date);
            if(query.exec()){
                QMessageBox::warning(this,tr("Succeed"),
                                     tr("Add library card succeed!"),
                                     QMessageBox::Yes);
                ui->CardNoEdit->clear();
                ui->DeptEdit->clear();
                ui->NameEdit->clear();
                ui->CardNoEdit->setFocus();
                on_QueryBtn_clicked();
            }
            else
                QMessageBox::warning(this,tr("Error"),
                                     tr("Add library card failed!"),
                                     QMessageBox::Yes);
        }
```

```
    }
    else
        QMessageBox::warning(this,tr("Error"),
                             tr("Input information invalid!"),
                             QMessageBox::Yes);
}

void CardManage::on_DeleteBtn_clicked()
{
    QString CardNo = ui->tableWidget->item(ui->tableWidget->currentRow(),
0)->text();
    QSqlQuery query;
    query.prepare("delete from librarycard "
                  "where cardNo = :CardNo;");
    query.bindValue(":CardNo", CardNo);
    query.exec();
    ui->CardNoEdit->clear();
    ui->DeptEdit->clear();
    ui->NameEdit->clear();
    ui->CardNoEdit->setFocus();
    on_QueryBtn_clicked();
}
```