实验5 数据库程序设计

3190105008 洪常凯

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

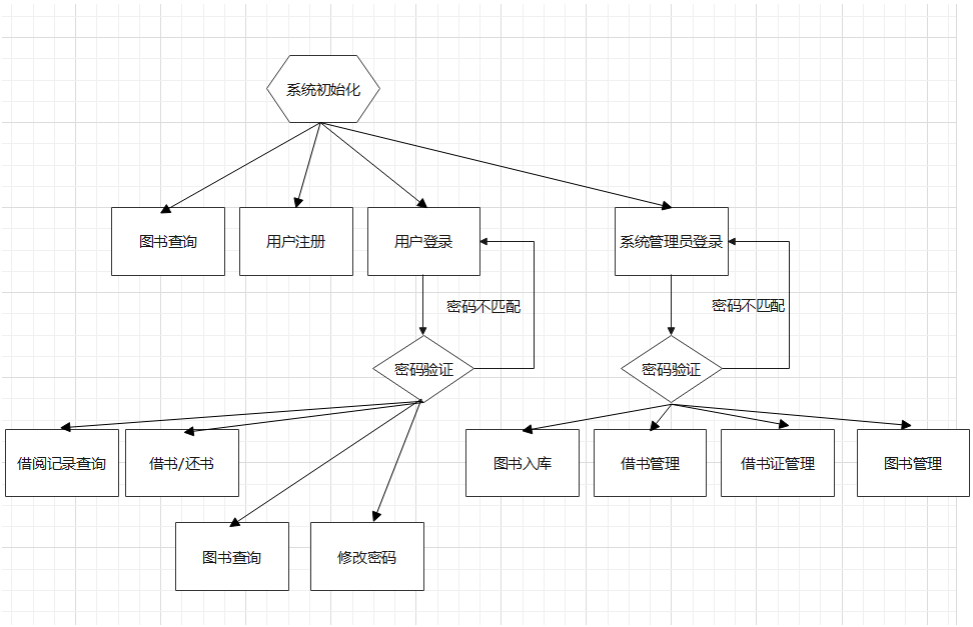
开发工具：Qt

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、图书查询、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：



系统初始时有图书查询、用户注册、用户登录、管理员登录四个选项，图书查询为公共功能模块，不需要登录也可操作。管理员成功登录后，便可以选择进入图书入库、借书管理、借书证管理功能。

模块名称	功能描述
图书查询	可以再输入栏中输入书籍名称，点击“搜索”查询该本书的信息
用户注册	新用户输入用户名、密码且确认密码后即可完成注册
用户登录	用户输入用户民和密码后登录，从而拥有用户的借书和还书功能
系统管理员登录	管理员输入用户民和密码后登录，从而拥有管理员的 图书入库、借书管理和借书证管理功能
借书	用户借书
还书	用户还书
修改密码	用户登录后可以修改自己的密码
图书入库	管理员输入书籍信息，点击“入库”完成图书添加
借书管理	查看图书和用户的借阅情况，可选择删除借阅信息
借书证管理	增加或删除借书证
图书管理	可以查询、更新、删除图书

2、数据库表设计

图书信息表（Book）

字段名	数据类型	主键	说明
Bid	varchar(20)	是	图书编号
BookName	varchar(20)		书名
Publisher	varchar(30)		出版社
Author	varchar(30)		作者
Year	int		出版年份
Total	int		总藏书量
Storage	int		库存数
UpdateTime	date		添加时间

用户表（User）：

字段名	数据类型	主键	说明
Uid	varchar(20)	是	用户名
Password	varchar(20)		密码
Type	varchar(10)		用户类型 （普通用户还是管理员）

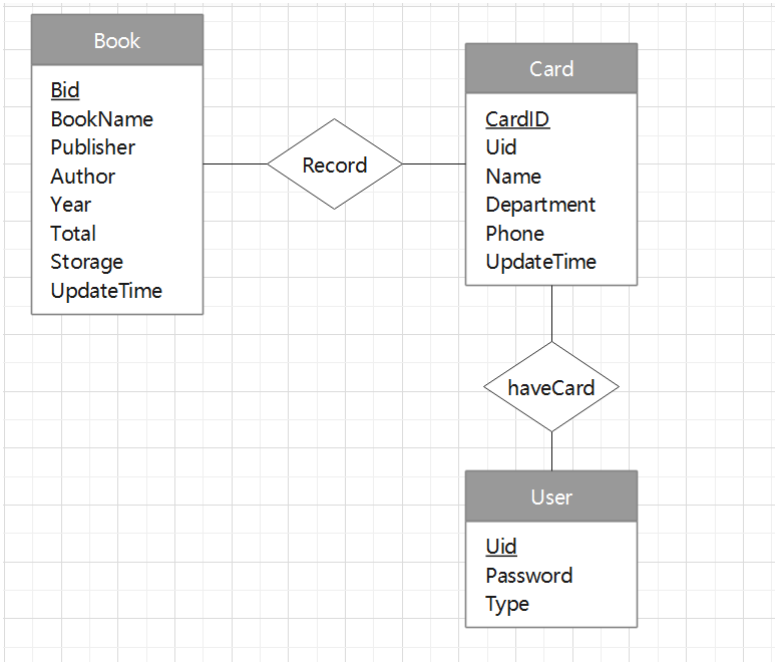
借书证表（Card）：

字段名	数据类型	主键	说明
CardID	varchar(20)	是	卡号
Uid	varchar(20)		对应用户
Name	varchar(20)		姓名
Department	varchar(20)		持卡人所在系
Phone	varchar(20)		联系方式
UpdateTime	date		办卡时间

借阅表（Record）：

字段名	数据类型	主键	说明
Rid	varchar(20)	是	记录序列号
CardID	varchar(20)		借书卡号
Bid	varchar(20)		图书编号
BorrowTime	date		借书日期
ReturnTime	date		还书日期

下面是构建的E-R图：

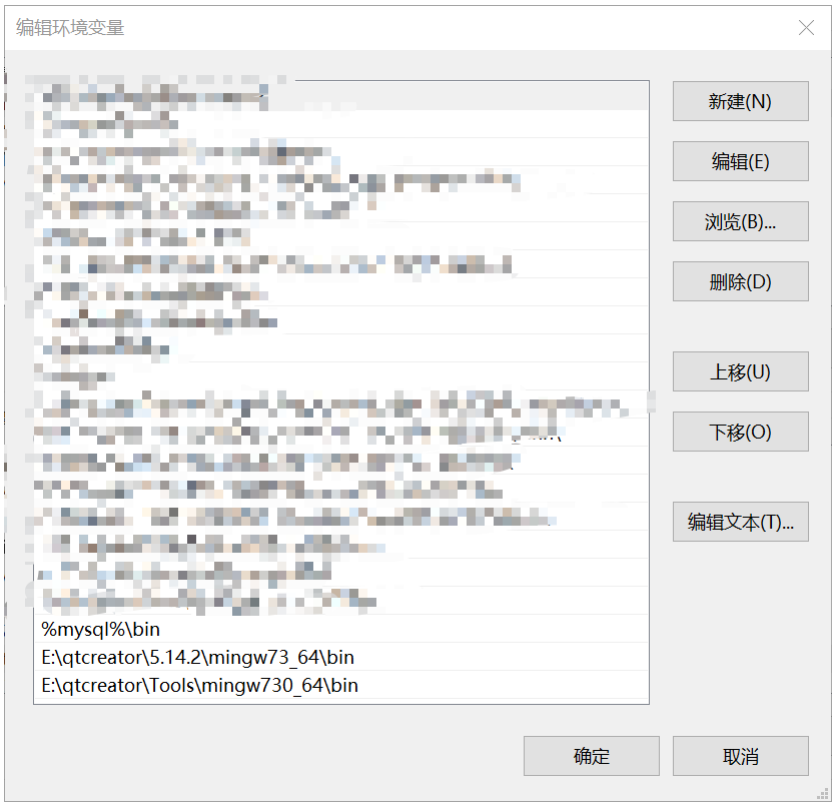


3、各个功能的具体实现

1、前期准备工作和qt连接mysql

（1）安装QT

（2）配置path环境变量



主要是添加两个bin文件夹，需要根据自己的安装位置来调整

（3）编译mysql驱动（用Qt打开mysql.pro文件）

参考路径如下：E:\qtcreator\5.14.2\Src\qtbase\src\plugins\sqldrivers\mysql

用QT打开该文件夹中的mysql.pro，会自动打开一个工程文件，在mysql.pro里修改及添加代码

```

#QMAKE_USE += mysql     //第6行左右，在QMAKE_USE += mysql 前面加上#

win32:LIBS += -LD:/mysql/mysql-8.0.19-winx64/lib -llibmysql

INCLUDEPATH += D:/mysql/mysql-8.0.19-winx64/include

DEPENDPATH += D:/mysql/mysql-8.0.19-winx64/include

//注意修改斜杠方向，复制下来的路径是向右的，编译需要向左的
```

修改完成后，重新构建项目（ctrl + R）

（4）拷贝文件

若上一步正常完成，则会在QT下载的同级目录(我这里是E盘)下生成一个plugins文件夹，将其中的qsqlmysql.dll 、qsqlmysqld.dll 拷贝至Qt所对应mingw的sqldrivers中，拷贝目标的路径参考如下：

E:\qtcreator\5.14.2\mingw73_64\plugins\sqldrivers

（5）测试连接

首先需要在.pro文件添加：

```
QT       += core gui sql
QT       += sql
```

这里将连接实现成了一个.h文件加.cpp文件的形式，方便以后可能的连接（本实验只用到了一次），具体代码如下：

connection.h：

```
#ifndef CONNECTION_H
#define CONNECTION_H
#include <QMessageBox>
#include <QSqlDatabase>
#include <QSqlQuery>
#include <QtDebug>
#include <QtGui>
#include <qdebug.h>


bool createConnection();


#endif // CONNECTION_H
```

connection.cpp：

```
#include "connection.h"

```

```
bool createConnection()
{
    QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");
    db.setHostName("127.0.0.1");
    db.setDatabaseName("db01");
    db.setUserName("root");
    db.setPassword("hongchangkai123");
    bool ok = db.open();
    if (ok)
    {
        qDebug() << "连接成功" ;
        return 1;
    }
    else
    {
         qDebug() << "连接失败" ;
         return 0;
    }
}

```

main.cpp：

```
#include "mainwindow.h"
#include "connection.h"
#include <QApplication>
#include <QLabel>
#include <QPushButton>
#include <QDebug>
#include <QSqlDatabase>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    if (!createConnection())
          return 1;
    return a.exec();
}
```

构建整个项目，如果下方的应用程序输出显示连接成功则完成连接

注意：Qt中没有自带的mysql数据库插件 qsqlmysql.dl 与 qsqlmysql.dlld，需要到自带的mysql.pro中自行编译。

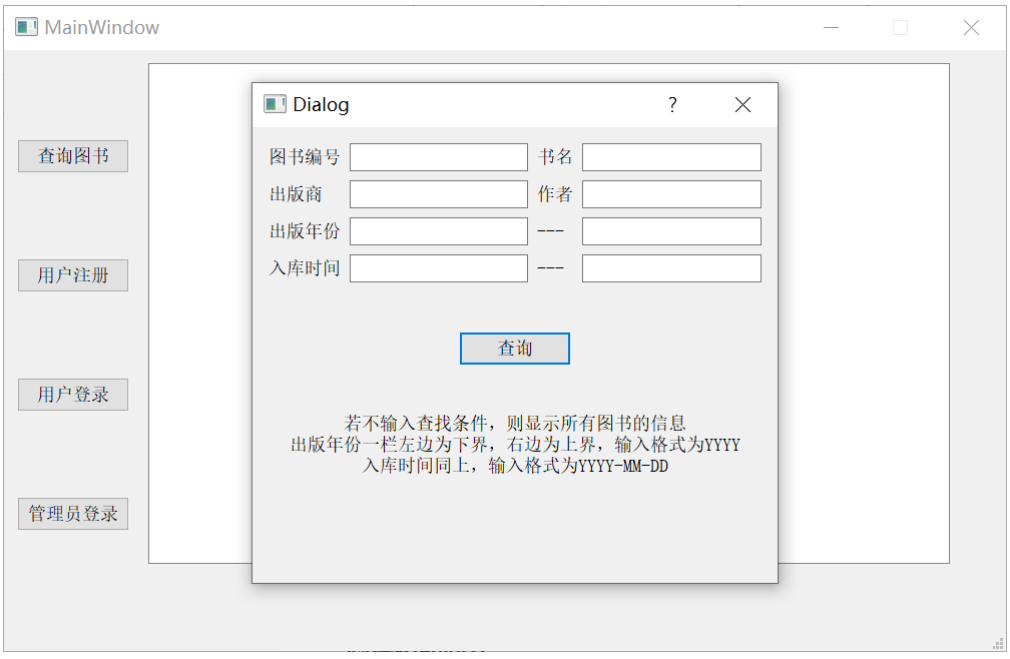
2、系统初始界面

这里先简要介绍一下qt的一些比较特别的功能：

- ui设计的各个组件：
  - lineEdit：提供给使用者输入信息的地方，同时可以用text()函数来获取使用者输入的信息
  - pushBotton：按钮组件，可将其与`clicked()`槽函数（后面介绍）结合起来来执行一些功能
  - tableview：表格视图组件。可以用`setModel()`函数来打印sql执行语句返回的结果
  - stackedWidget：可以实现在一个窗口下不同页面之间的切换，避免设计多个ui

- qt提供了QSqlQuery和QSqlQueryModel两个类。前者用于执行sql语句，`exec()`函数用于执行sql语句，其返回一个bool值来判断sql语句是否执行成功。同时根据`first()`或`next()`函数来判断执行成功情况下返回的结果指向第一条记录（`next()`能继续指向下一条记录），可以用于判断结果集是否为空，之后可以用`value()`函数来获取返回集里的各个属性。后者可以和tableview组件结合起来，利用`setQuery()`函数以表格的形式打印出sql语句（主要是select语句）的返回结果
- qt的信号与槽机制。首先是槽函数，通过将槽函数和组件结合起来，可以在组件进行了某些行为下用槽函数来执行某些功能。本次实验主要是用到了pushBotton的`clicked()`槽函数和tableview的`doubleclicked()`函数来获取使用者所提供的一些信息。其次是信号机制，通过`connect()`函数将不同的槽函数联系起来，并传递一些信息。

（1）查询图书



查询图书分为两块，查询条件设置和查询结果显示。使用者进入系统初始化界面后，点击左边导航栏中的查询图书即转到查询图书界面并弹出查询条件框。用户可以按照图中所给的输入框输入条件来查询图书。

从lineEdit获取使用者输入的信息，点击查询按钮后，即开始执行与该查询按钮结合的槽函数`void Query::on_pushButton_clicked()`，将所有语句组合成一个select查询语句。执行了sql语句后在查询图书页面的tableview组件里打印返回图书的信息。其中出版年份和入库时间可以是一个有限区间也可以是一个无穷区间，取决于使用者是填写了左边还是右边的输入框亦或是二者都填写。mysql可以支持日期类型的直接比较。

点击了查询按钮后，槽函数将条件组合完毕，通过信号`send_Bookquery_condition(Condition)`将该条件发送给主窗口的接受槽函数最终执行查询语句。然后通过信号`sendsignal()`向主窗口的槽函数`show()`发送信息表明当前窗口关闭，并显示主窗口界面。最终调用`close()`函数关闭窗口

获取并组合查询条件：

```
void Query::on_pushButton_clicked()
{
    QString Condition;
    QString str, str1;

    str = ui->lineEdit_query_Bid->text();
    if(str != "") {      //非空
        str = "'" + str + "'";
        Condition += "Bid = " + str;
    }

    str = ui->lineEdit_query_BookName->text();
    if(str != "") {
        if(Condition != "") Condition += " and ";
        str = "'" + str + "'";
        Condition += "BookName = " + str;
    }

    str = ui->lineEdit_query_publisher->text();
    ...

    str = ui->lineEdit_query_Author->text();
    ...

    str = ui->lineEdit_query_Year1->text();
    str1 = ui->lineEdit_query_Year2->text();
    if(str == "" && str1 != "") {
        if(Condition != "") Condition += " and ";
        str1 = "'" + str1 + "'";
        Condition += "Year <= " + str1;
    }else if(str != "" && str1 == "") {
        if(Condition != "") Condition += " and ";
        str = "'" + str + "'";
        Condition += "Year >= " + str;
    }else if(str != "" && str1 != "") {
        if(Condition != "") Condition += " and ";
        str = "'" + str + "'";
        str1 = "'" + str1 + "'";
        Condition += "(Year >= " + str + "and Year <= " + str1 + ")";
    }

    str = ui->lineEdit_query_UpdateTime1->text();
    str1 = ui->lineEdit_query_UpdateTime2->text();
    ...

```

```
    Condition += ";" ;
    //qDebug() << Condition;
    emit send_Bookquery_condition(Condition);
    emit sendsignal();
    this->close();
}
```

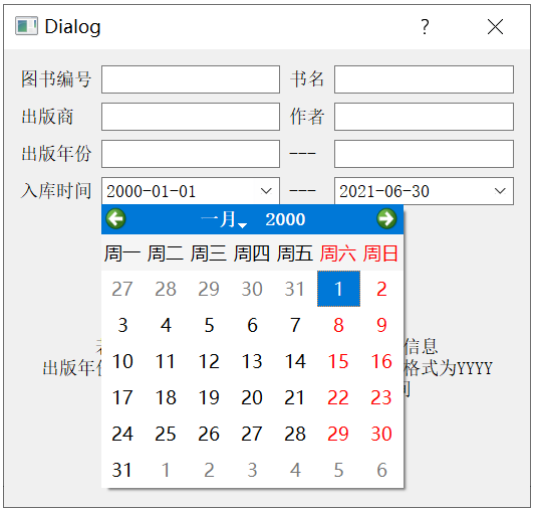
后续将入库时间的输入更改为日历组件Date Edit, 在初始化时默认的时间格式为yyyy-MM-dd，并将右边框设置为当前时间，左边不设置，默认为2000-01-01

```
Query::Query(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::Query)
{
    ui->setupUi(this);
    ... //其他初始化
    ui->query_dateEdit_lowerbound->setDisplayFormat("yyyy-MM-dd");   //设置输出格式;
    ui->query_dateEdit_lowerbound->setCalendarPopup(true);  //呈日历表的样式进行显示
    ui->query_dateEdit_upperbound->setDateTime(QDateTime::currentDateTime()); //设置当前时间为开始时间
    ui->query_dateEdit_upperbound->setDisplayFormat("yyyy-MM-dd");   //设置输出格式;
    ui->query_dateEdit_upperbound->setCalendarPopup(true);
}
```

相应的，更改获取入库时间的方式

```
    QDateTime current_date_time = QDateTime::currentDateTime(); //获取当前时间
    QString current_date = current_date_time.toString("yyyy-MM-dd");//时间格式变成字符串
    QString lowerbound = "2000-01-01";
    str = ui->query_dateEdit_lowerbound->text();
    str1 = ui->query_dateEdit_upperbound->text();
    if( !str.compare(lowerbound) && str1.compare(current_date)) {
        if(Condition != "") Condition += " and ";
        str1 = "'" + str1 + "'";
        Condition += "UpdateTime <= " + str1;
    }else if(str.compare(lowerbound) && !str1.compare(current_date)) {
        if(Condition != "") Condition += " and ";
        str = "'" + str + "'";
        Condition += "UpdateTime >= " + str;
    }else if(str1.compare(lowerbound) && str1.compare(current_date)) {
        if(Condition != "") Condition += " and ";
        str = "'" + str + "'";
        str1 = "'" + str1 + "'";
        Condition += "(UpdateTime >= " + str + "and UpdateTime <= " + str1 + ")";
    }
```

效果呈现如下：



打印图书：

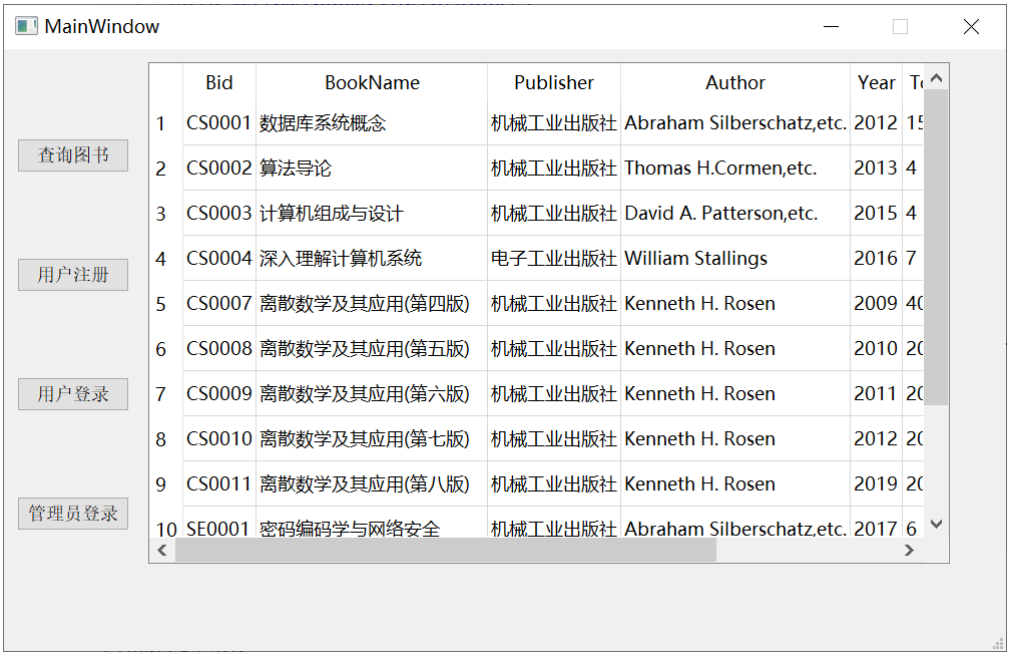
```
void MainWindow::PrintBook()
{
    QString str;
    QSqlQueryModel *dbmodel = new QSqlQueryModel();
    ui->tableView->setModel(dbmodel);
    ui->tableView->horizontalHeader()->
        setSectionResizeMode(QHeaderView::ResizeToContents);
    if(0 != Condition_main.compare(";"))
        str = "select Bid as 图书编号, BookName as 书名, Publisher as 出版社, "
              "Author as 作者, Year as 出版年份, Total as 总量, Storage as 余量, "
              "UpdateTime as 最新入库时间 from Book where " + Condition_main;
    else
        str = "select* from Book";
    qDebug() << str;
    dbmodel->setQuery(str);
}
```

用于是执行和打印二者同时执行，因此直接利用QSqlQueryModel类要比从QSqlQuery类变量中一行一行的读取要简练得多。同时，通过

`ui->tableView->horizontalHeader()->setSectionResizeMode(QHeaderView::ResizeToContents);`

来设置表格打印的形式——自适应字段长度

查询结果图示：



(2) 注册和登录

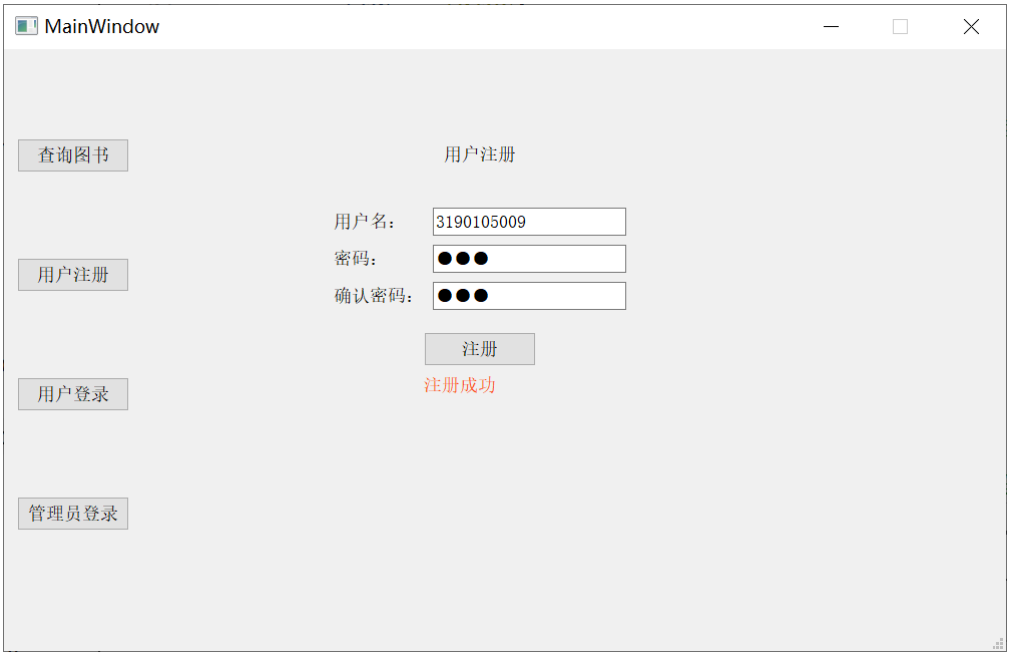
主要是从lineEdit获取使用者输入的用户名和密码，不同页面给予了登录者不同的属性，组合成select语句后判断返回集是否为空，从而决定是否登录成功，并输除提示信息

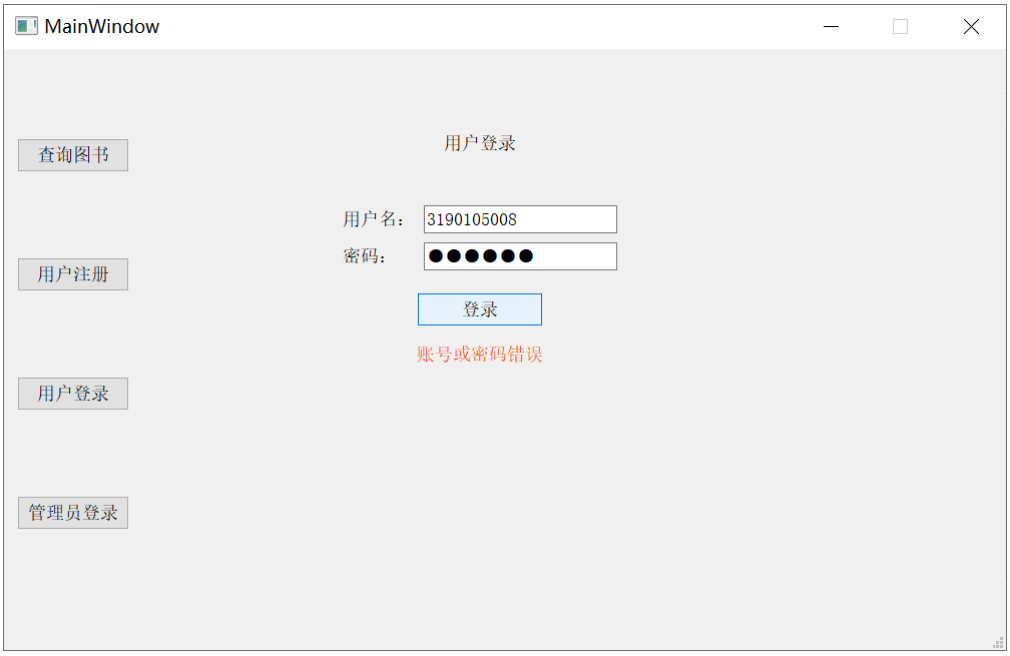
登录成功后，根据用户的类型新建一个我们自定义的User类变量或是Adm（管理员）类变量，隐藏主窗口界面，显示用户或管理员界面。当退出用户或管理员界面时，通过发送信号给show()函数重新显示主窗口界面；

需要注意的是，当我们登录成功并跳转的时候，我们需要将当前用户的Uid一并发送过去，因此还需要一个信号来发送Uid，即`emit send_uid(uid);`，同时用User类的槽函数`receive_uid(QString)`来接受

```
if(query.next()) {  //判断结果集是否为空，即判断用户名、密码是否正确
    ui->label_rootlog_infor->setText("登录成功");
    this->hide();
    User *u = new User(this);
    u->setFixedSize(1000, 600);
    connect(u, SIGNAL(sendsignal()), this, SLOT(show()));
    connect(this, SIGNAL(send_uid(QString)), u, SLOT(receive_uid(QString)));
emit send_uid(uid);
    emit send_uid(uid);
    u->show();  //转到用户界面
}
```

```cpp
void User::receive_uid(QString ID)
{
    this->uid = ID;
}
```



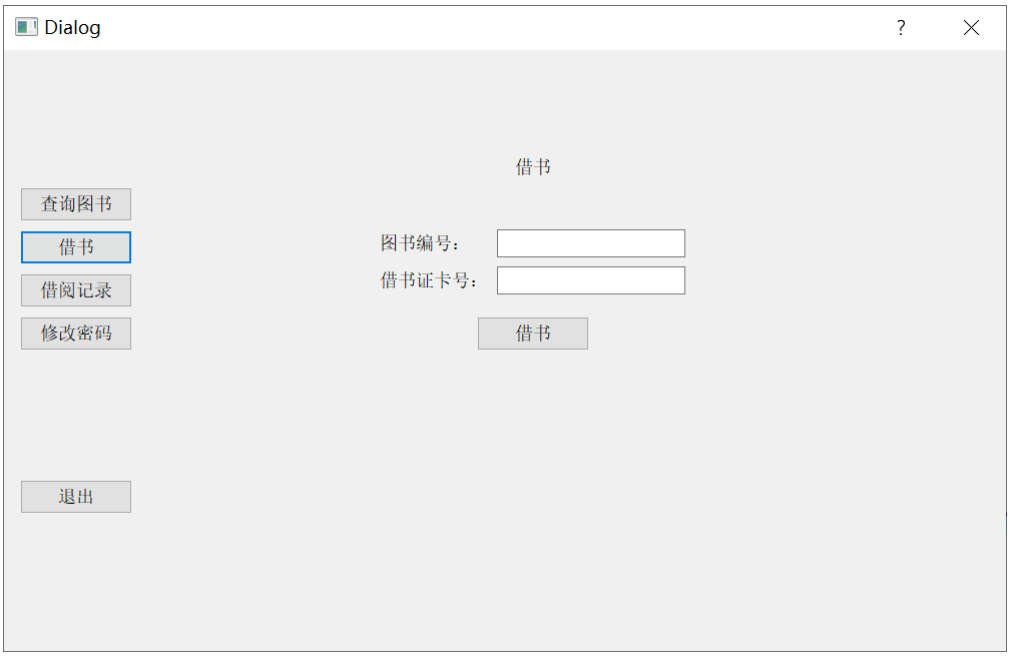


3、用户界面

（1）查询图书

查询图书与系统初始界面的查询图书效果相同，主要是更改组件的设置，这里不做更多的介绍

(2) 借书



通过第一部分查询到的图书编号，和自己拥有的借卡证，首先检查输入的图书编号和借书证卡号是否正确，通过QsqlQuery类的exec()函数和next()函数来检查各种输入错误。然后将这两个条件组合成一个Insert语句插入到Record表中，同时更新Book表里的Storage属性（减一）

同时，qt提供了QDateTime类来获取系统的当前时间，利用toString函数来将时间转换成字符串的形式。

输入条件检查：

```
if(0 == BookID.size())
    {
        ui->label_userBorrow_infor->setText("图书编号不能为空!");
        flag = false;
    }

    if(0 == CardID.size())
    {
        ui->label_userBorrow_infor->setText("卡号不能为空!");
        flag = false;
    }


    if(flag) {
        BookID = "'" + BookID + "'";
        CardID = "'" + CardID + "'";
        str = "select * from Book where Bid = " + BookID + ";" ;
        //qDebug() << str;
        bool successBook = query.exec(str);
        if(successBook) {       //查找图书语句是否能正确执行
            if(query.next()) {  //结构是否为空集
                str.insert(str.length()-1, " and Storage > 0");
                bool successStorage = query.exec(str);
                //qDebug() << str;
                if(successStorage) {        //查询余量语句是否能正确执行
                    if(query.next()) {
                        str = "select * from Card where CardID = " + CardID
+ " and "
```

```
                        "Uid = " + this->uid + ";" ;
                        //qDebug() << str;
                        bool successCard = query.exec(str);
                        if(successCard) {
                            if(query.exec()) {
                                Update_RecordandBook(BookID, CardID);
                            }else {
                                ui->label_userBorrow_infor->setText("此卡号不属于当前用户");
                            }
                        }else {
                            ui->label_userBorrow_infor->setText("查询卡号发生错误");
                        }
                    }else {
                        ui->label_userBorrow_infor->setText("此书已全部被借走");
                    }
                }else {
                    ui->label_userBorrow_infor->setText("查询余量发生错误");
                }
            }else {
                ui->label_userBorrow_infor->setText("馆内没有收录该书");
            }
        }else {
            ui->label_userBorrow_infor->setText("查找书目发生错误");
        }
    }
```

更新借书记录和图书余量信息：

```
void User::Update_RecordandBook(QString BID, QString CID)
{
    QSqlQuery query;
    QString str;
    //更新Record表
    QDateTime current_date_time = QDateTime::currentDateTime();
    QDateTime return_date_time = current_date_time.addMonths(1);
    QString current_date = current_date_time.toString("yyyy-MM-dd");
    QString return_date = return_date_time.toString("yyyy-MM-dd");
    current_date = "'" + current_date + "'";
    return_date = "'" + return_date + "'";
    str = "Insert Record values(" +  CID + ", " + BID + ", "
            + current_date + ", " + return_date + ");";
    bool success = query.exec(str);
    if(!success) {
        //qDebug() << query.record();

        ui->label_userBorrow_infor->setText("您已借阅过相同的书籍");
        return;
    }
    //

    //更新Book表
    str = "update Book set Storage = Storage - 1 where Bid = " + BID + ";" ;
    qDebug() << str;
    success = query.exec(str);
```

```
    //

    if(success) ui->label_userBorrow_infor->setText("借阅成功");
    return;
}
```

（3）借阅记录&还书

点击旁边导航栏的借阅记录按钮转到借阅记录界面，该界面的tableview组件会直接显示当前用户的借阅图书的情况。通过tableview的doubleclicked()函数来获取用户所点击的行的信息，然后点击该页面的还书按钮即可实现还书

qt提供了QModelIndex类来表示用户所选中单元格的信息，`row()`函数获得其所在行，`index()`函数获取所需要的行和列，即图书编号和借书证卡号的位置信息，最终用`data()`函数和`toString()`函数获取所需要的的字符串信息。

最终，通过自定义信号和槽函数来传递这些信息，储存到定义的全局静态变量`return_BID`和`return_CID`中（因为后面需要频繁用到这些信息，故定义为全局变量，静态变量是避免该类外的文件使用）

tableview的槽函数：

```
void User::on_tableView_userRecord_doubleClicked(const QModelIndex &index)
{
    QAbstractItemModel *Imodel = ui->tableView_userRecord->model();
    QModelIndex Iindex0 = Imodel->index(index.row(),0);//index.row()为算选择的行号,0为所选中行的第一列,即Bid
    QModelIndex Iindex5 = Imodel->index(index.row(),5); //CID
    QVariant datatemp0 = Imodel->data(Iindex0);
    QVariant datatemp5 = Imodel->data(Iindex5);
    QString BID = datatemp0.toString();//BID即为所选择行的第一列的值
    QString CID = datatemp5.toString();
    //qDebug() << BID;
    connect( this, SIGNAL(send_return_BID(QString)), this,
            SLOT(receive_return_BID(QString)) );
    connect( this, SIGNAL(send_return_CID(QString)), this,
            SLOT(receive_return_CID(QString)) );
    emit send_return_BID(BID);  //传递得到的信息
    emit send_return_CID(CID);
}
```

还书：

```
void User::on_pushButton_Nav_user_return_clicked()
{
    QSqlQuery query;
    QString str;

    //从Record表中删除借阅记录
    str = "delete from Record where Bid =" + return_BID + " and CardID = "
            + return_CID + ";" ;
    bool success_del = query.exec(str);
    qDebug() << str;
```

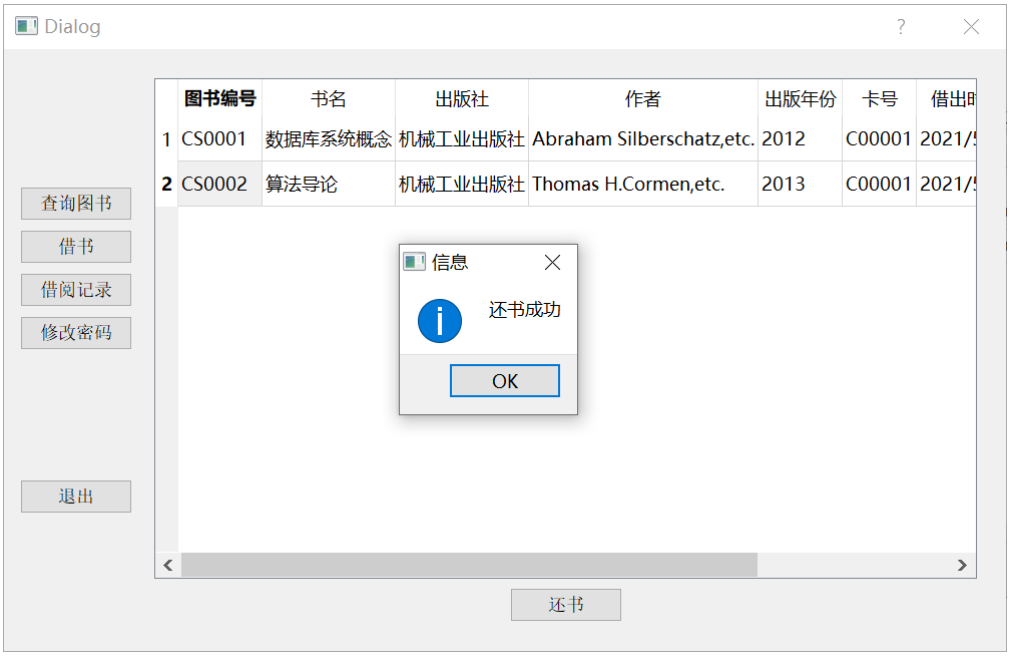
```
    //更新Book表的图书余量Storage
    str = "update Book set Storage = Storage + 1 where Bid = " + return_BID
+ ";" ;
    bool success_upd = query.exec(str);

    if(success_del && success_upd) {
        QMessageBox::information(this, "信息", "还书成功", QMessageBox::Ok);
    }else {
        QMessageBox::critical(this, "错误", "还书失败", QMessageBox::Ok);
    }

    PrintRecord();
}C++
```

前面所说的还书和余量更新利用了delete和update语句来完成。qt提供了QMessageBox类来提供一个提示框的显示，其第一个变量代表关闭消息框后回到的位置，后面两个为消息框显示的信息，最后一个为按钮的类型

总体效果：



(4) 修改密码

修改密码与前面登录注册实现的方法基本一致，即利用select语句来判断匹配情况，这里不再具体介绍

(5) 退出

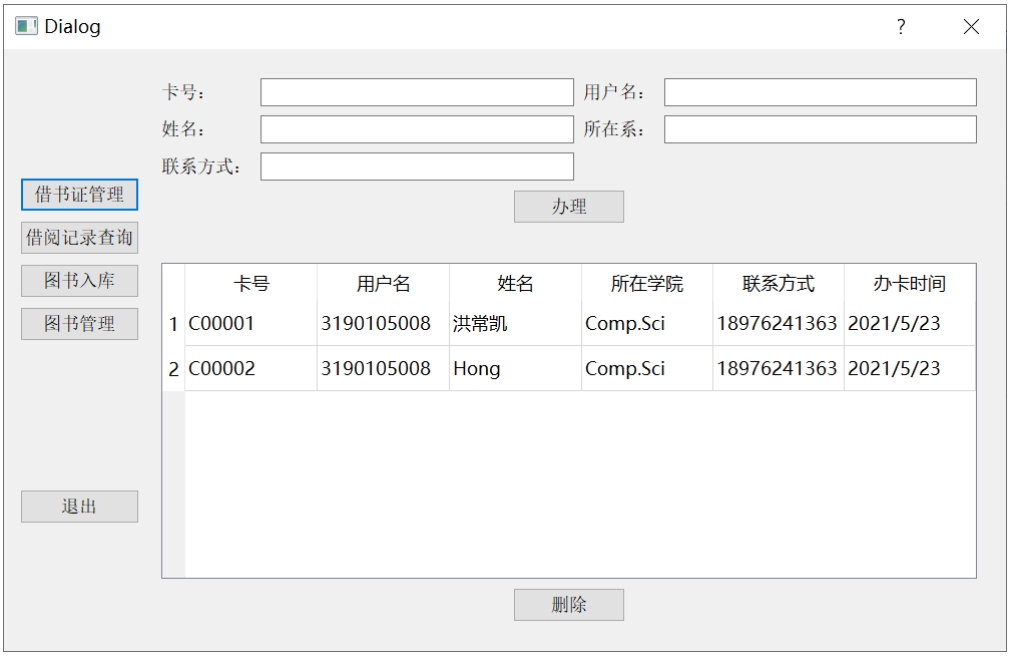
通过前面连接的信号与槽，当点击退出按钮时，像show()函数发送一个信号，关闭用户界面的同时显示主窗口界面

```
connect(u, SIGNAL(sendsignal()), this, SLOT(show()));
```

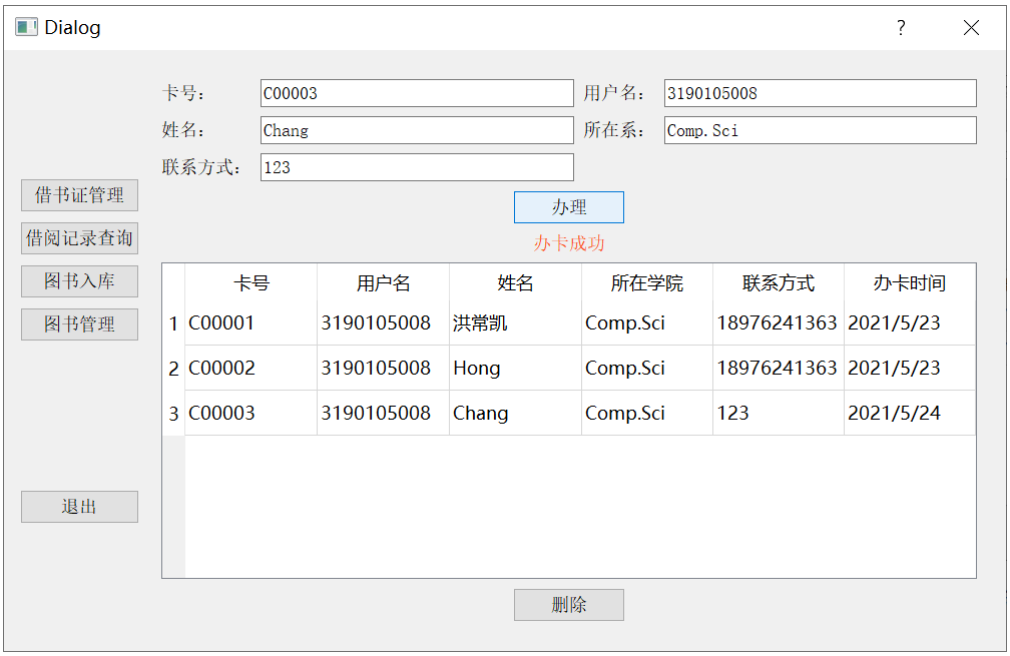
```
void User::on_pushButton_Nav_user_exit_clicked()
{
    emit sendsignal();
    this->close();
}
```

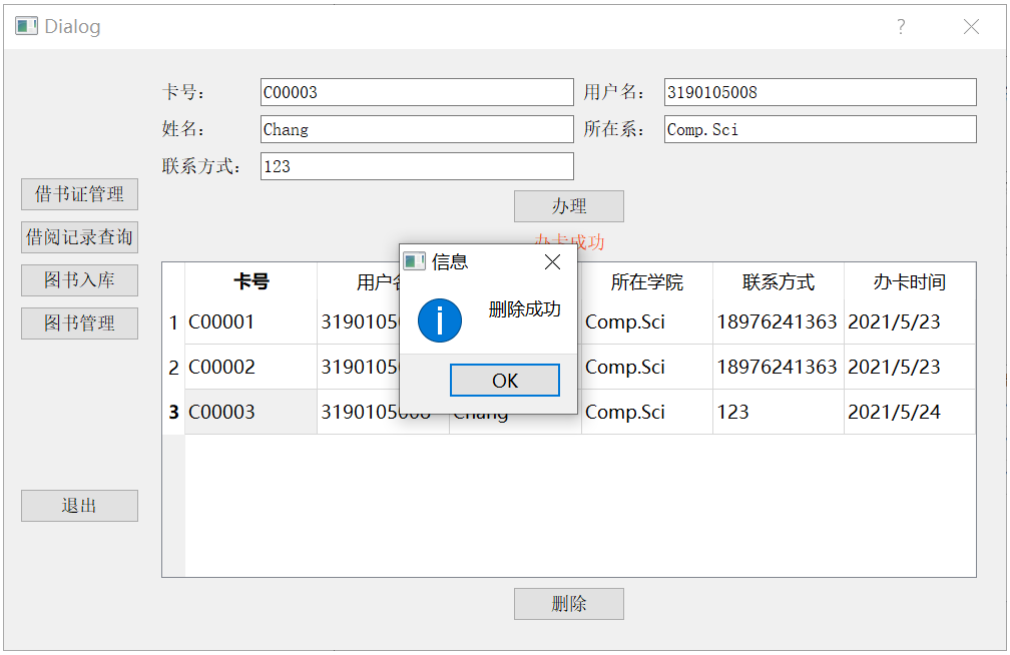
管理员界面

（1）借书证管理



管理员输入相应的信息后点击办理即可完成借书证的添加，同时会在该页面显示添加成功与否的提示。同样，双击选中其中的一行中一个属性后，点击删除，即可完成借书证的删除





	卡号	用户名	姓名	所在学院	联系方式	办卡时间
1	C00001	3190105008	洪常凯	Comp.Sci	18976241363	2021/5/23
2	C00002	3190105008	Hong	Comp.Sci	18976241363	2021/5/23

图形界面的具体实现方法与之前的查询界面和还书类似，这里不再展示具体代码，主要介绍与之前不同的一些地方以及sql字符串

打印设置：

```
ui->tableView_adm_allCard->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);
```

由于需要打印的字段较少，因此将显示模式更改为Stretch，可以自适应填充整个tableview

sql字符串：

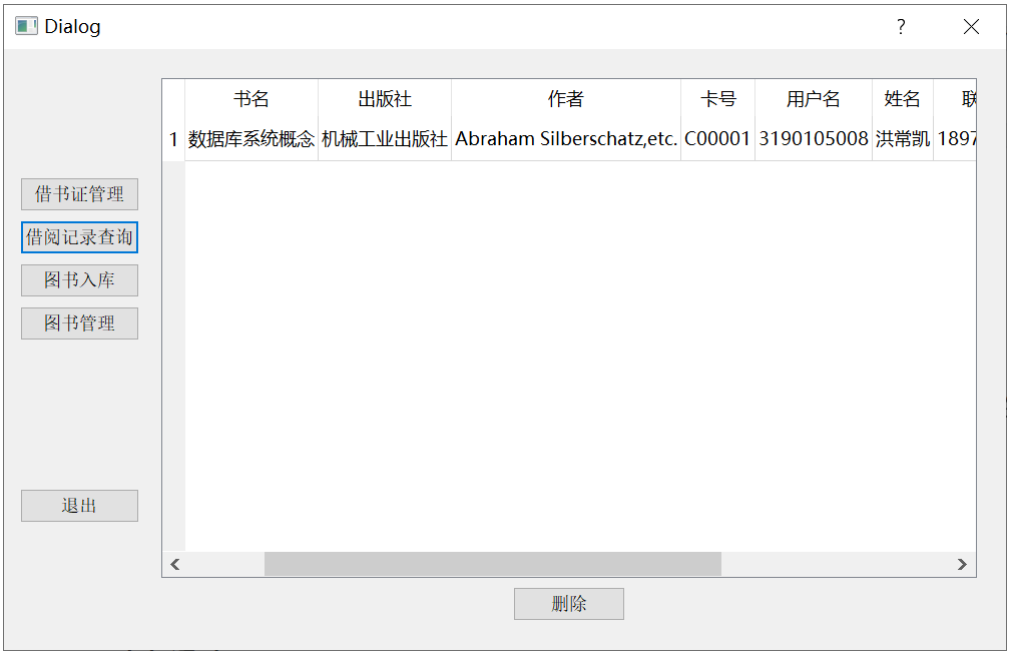
打印：`str = "select CardID as 卡号, Uid as 用户名, Name as 姓名, Department as 所在学院, ""Phone as 联系方式, UpdateTime as 办卡时间 from Card;" ;`

插入：`str = "Insert Card values(" + CID + ", " + UID +", " + Name + ", " + Dept + ", " + Phone + ", " + current_date + ");" ;`

删除：`str = "delete from Card where CardID =" + return_CID + ";" ;`

（2）借阅记录管理

该界面可以显示所有的借阅记录，同样可以双击选中进行删除，具体与用户界面的还书基本一致



(3) 图书入库

单个图书录入时，图书编号和添加册数不能为空，否则会无法入库并提示错误信息。

```
if(0 == BID.size()) {
    flag = false;
    ui->label_storage_infor->setText("图书编号不能为空\n");
}
if(0 == Amount.size()) {
    flag = false;
    ui->label_storage_infor->setText("添加册数不能为空\n");
}
```

图书入库时，有可能该书已经存在从而导致insert语句执行错误。为此，需要先执行select语句来判断需要入库的图书是否存在。

```
str = "select Bid from Book where Bid = " + BID + ";" ;
qDebug() << str;
query.exec(str);
if(query.next()) {
    该图书已存在，增加册数（总量和余量）
}else {
    该图书尚未录入过，插入新图书
}
```

如果图书已经存在，那么执行update语句来更新Book表的图书信息；如果图书不存在，那么执行insert语句来更新Book表的图书信息

同时，此处还提供了批量导入的功能。QFileDialog类可以打开文件导入的窗口，在“C:\\”默认初始路径打开文件窗口（QFileDialog 类型），并设定过滤器只选择 txt 后缀的文件。读取后“文件路径”行编辑器设为读取的文件路径。

获取文件后，需要用QFile类来打开文件获取信息。`file.open(QIODevice::ReadOnly);`以只读的方式打开文件，`isOpen()`函数检查打开文件是否成功。QTextStream类用于从文件中读取信息，通过`readline()`函数来讲信息读取到字符串`line`中。同时定义一个QList容器，其搜索是基于索引(index)的，并且可以接受字符串按照分隔符划分后的各个字符串(`linelist = line.split(",");`)，对其中一些特殊的数据类型进行为空时的处理。还可以利用`simplified()`函数实现开头结尾空格的消除，和中间多个空格归为一个空格，完成数据形式统一。

```
void Adm::on_pushButton_BulkImporting_clicked()
{
    QString  fileName = QFileDialog::getOpenFileName(this,"choose file",
                                                     "C:\\",tr("*.txt"));
    if(!fileName.isEmpty()) ui->label_adm_file->setText("文件路径: "+fileName);

    QFile file(fileName);
    file.open(QIODevice::ReadOnly);
    if(file.isOpen()) {
        qDebug() << "打开成功";
        QTextStream in(&file);
        in.setCodec("utf-8");
        QString line = in.readLine();
        QList<QString>linelist;
        while(!line.isNull()) {
            qDebug() << line;
            //line = line.mid(0, line.length());//返回line从index = 0开始长度为line的长度的字符串
            linelist = line.split(",");

            if(linelist.count() < 1) {
                break;
            }
            for(int i = 0; i < linelist.count(); i++) {
                linelist[i] = linelist[i].simplified();
                if(i == 3) {
                    if(0 == linelist[i].size()) linelist[i] = "";
                }else if(i == 4 || i == 5) {
                    if(0 == linelist[i].size())linelist[i] = "0";
                }
```

```
                qDebug() << linelist[i];
            }
            InsertBook(linelist[0], linelist[1], linelist[2], linelist[3],
                    linelist[4], linelist[5]);  //自定义插入函数
            line = in.readLine();
        }
    }else {
        qDebug() << "打开失败";
    }
}
```

(4) 图书管理

图书管理主要有三个功能：查询、更新和删除。其中查询和删除的主要实现方式前面已经介绍过了，这里主要介绍更新的一些实现方式。

双击需要更新的行之后点击更新按钮，tableview的`doublecliked()`槽函数会将所选中行的信息传送给更新窗口（主要是图书编号）。更新窗口根据获取到的信息，在lineEdit组件上利用

`ui->lineEdit_update_Bid->setPlaceholderText(oldBID);`

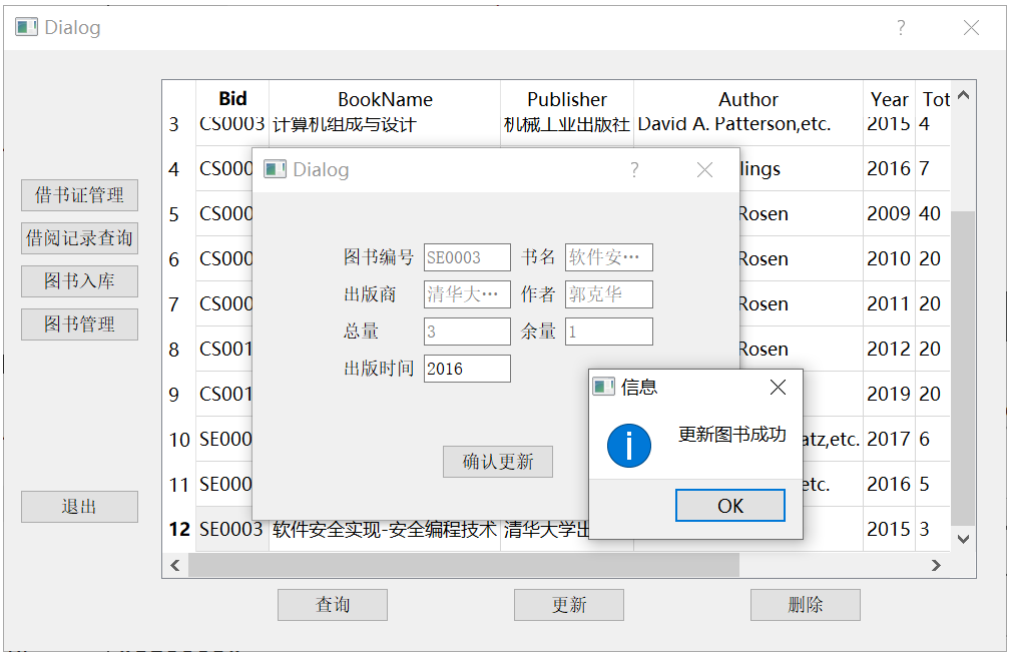
来显示原先图书的信息，需要注意不能在构造函数里调用该函数，否则oldBID会得到一个空值。可以定义一个公有函数`prepare()`来封装该功能

```
void Update::Placeholder()
{
    ...
    QString str = "select * from Book where Bid = '" + oldBID + "';" ;
    query.exec(str);
    qDebug() << str;
    if(query.next()) {
        qDebug() << query.value(0).toString();
        oldBookName = query.value(1).toString();
        oldPublisher = query.value(2).toString();
        ...
    }
```

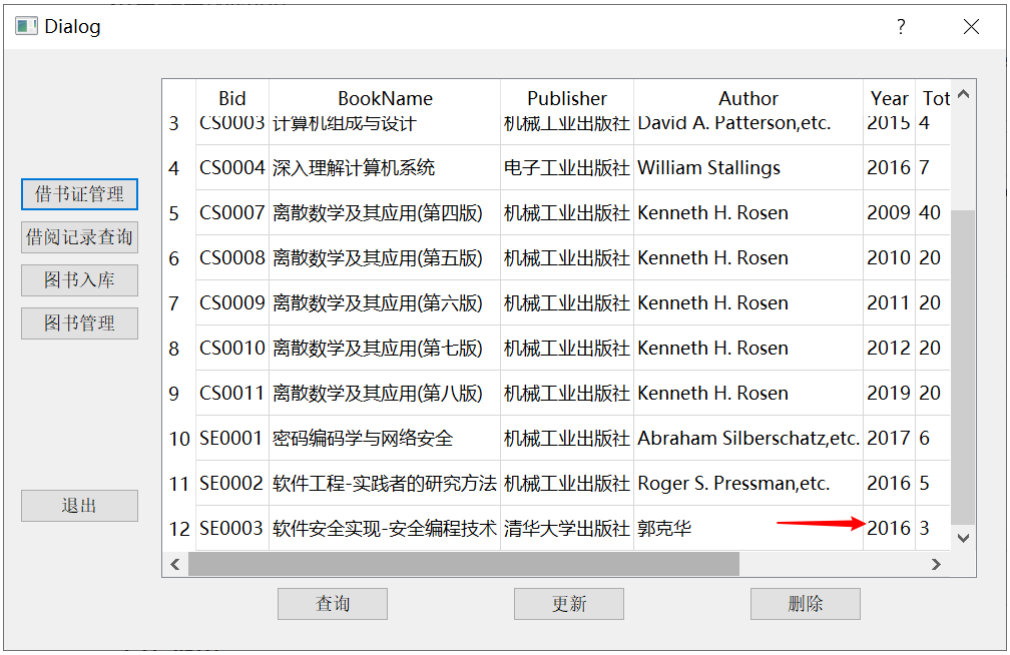
```
    qDebug() << oldBID << oldBookName << oldPublisher << oldAuthor <<
oldTotal
             << oldStorage << oldYear;
    ui->lineEdit_update_Bid->setPlaceholderText(oldBID);
    ui->lineEdit_update_BookName->setPlaceholderText(oldBookName);
    ui->lineEdit_update_publisher->setPlaceholderText(oldPublisher);
    ...
}
```

管理员可以修改图书的任意属性，但需要满足约束条件（如主键约束、外键约束、格式要求、范围约束），否则会提示更新错误。条件正确时将其整合成update语句执行重新打印即可

仅更新出版时间：

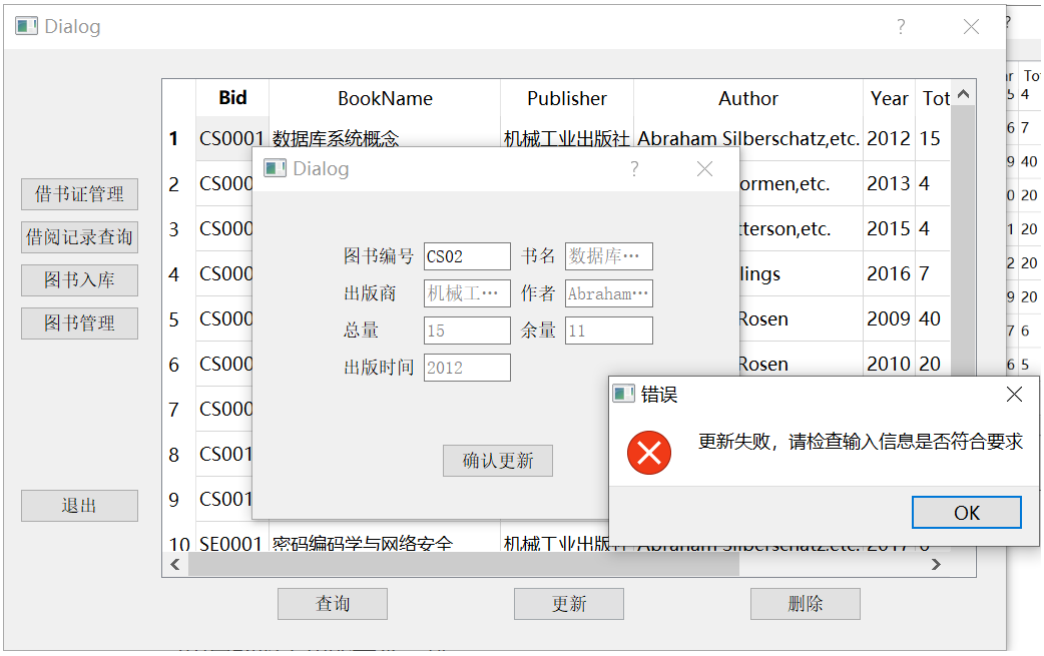


更新结果：



	Bid	BookName	Publisher	Author	Year	Tot
3	CS0003	计算机组成与设计	机械工业出版社	David A. Patterson,etc.	2015	4
4	CS0004	深入理解计算机系统	电子工业出版社	William Stallings	2016	7
5	CS0007	离散数学及其应用(第四版)	机械工业出版社	Kenneth H. Rosen	2009	40
6	CS0008	离散数学及其应用(第五版)	机械工业出版社	Kenneth H. Rosen	2010	20
7	CS0009	离散数学及其应用(第六版)	机械工业出版社	Kenneth H. Rosen	2011	20
8	CS0010	离散数学及其应用(第七版)	机械工业出版社	Kenneth H. Rosen	2012	20
9	CS0011	离散数学及其应用(第八版)	机械工业出版社	Kenneth H. Rosen	2019	20
10	SE0001	密码编码学与网络安全	机械工业出版社	Abraham Silberschatz,etc.	2017	6
11	SE0002	软件工程-实践者的研究方法	机械工业出版社	Roger S. Pressman,etc.	2016	5
12	SE0003	软件安全实现-安全编程技术	清华大学出版社	郭克华	2016	3

不满足外键约束：



（5）退出

与用户的退出功能基本一致

四、实验总结

本次实验主要使用了qt creator开发平台，利用C++语言完成一个数据库微应用的实现。不同于前面作业、实验等对sql语言的学习或是数据库的设计，本次实验主要练习了如何将数据库与实际应用相结合，利用宿主语言完成数据库的功能和结果的显示，使我对数据库的概念和应用又有了新的认识

五、参考网站

1、QT Help https://doc.qt.io/qt-5/qthelp-index.html

2、QT连接mysql

https://www.bilibili.com/video/BV1Jk4y167tt?p=1&share_medium=android&share_plat=android&share_source=QQ&share_tag=s_i×tamp=1621134962&unique_k=fLNqe4

3、QT窗体之间数据传递 https://www.cnblogs.com/91program/p/5284556.html

4、QT如何设计UI https://blog.csdn.net/Fdog_/article/details/107522283

5、百度 https://www.baidu.com/