

《数据库系统原理》实验报告

(实验名称：图书管理系统设计报告)

专 业	<u>计算机科学与技术</u>
班 级	<u>1302</u>
学 号	<u>3130100677</u>
学生姓名	<u>黄卓斐</u>
指导老师	<u>庄越挺</u>

浙江大学

2015 年 4 月 21 日

一、设计平台

- (1) MYSQL
- (2) Qt Creator 5.4.0 (图形界面)
- (3) QMYSQL 驱动

二、总体设计

- (1) 数据库表结构(SQL)

book:

```
create table book(  
    bno char(8),  
    category char(10),  
    title varchar(40),  
    press varchar(30),  
    year int,  
    author varchar(20),  
    price decimal(7,2),  
    total int,  
    stock int,  
    primary key (bno)  
);
```

card:

```
create table card(  
    cno char(7),  
    name varchar(10),  
    department varchar(40),  
    type enum('T','U','G','O'),  
    primary key (cno)  
);
```

borrow:

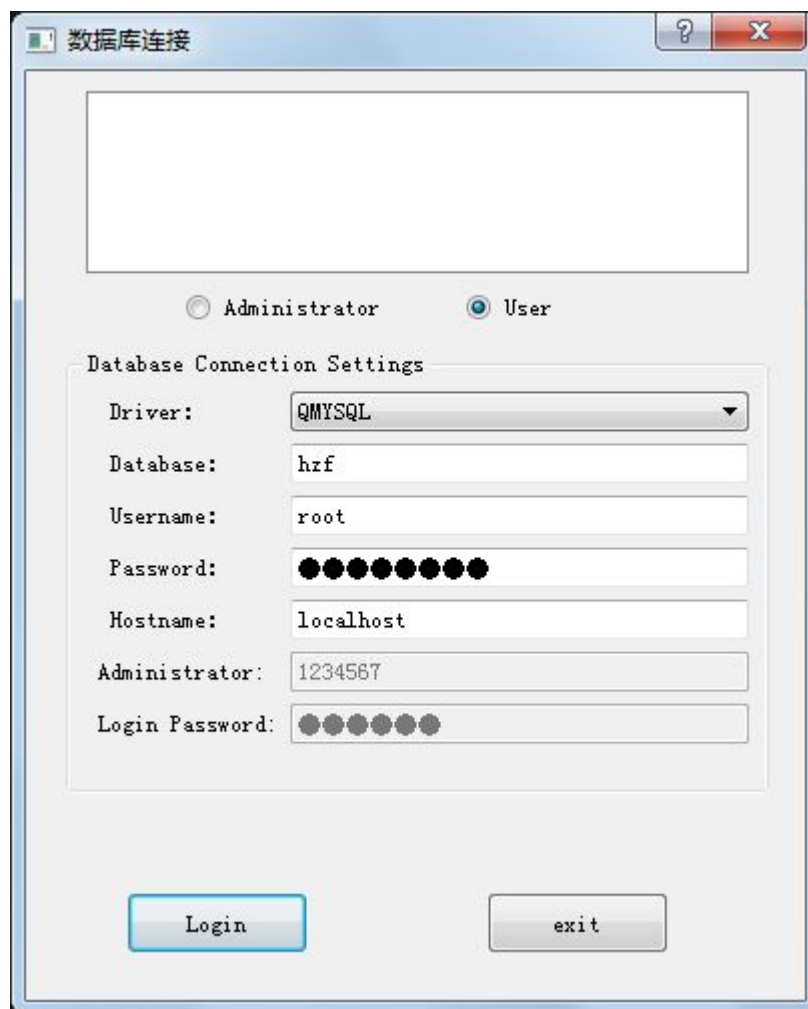
```
create table borrow(cno char(7),  
    bno char(8),  
    borrow_date datetime,  
    return_date datetime,  
    primary key (cno, bno, borrow_date),  
    foreign key (cno) references card(cno)  
        on delete cascade  
        on update cascade,  
    foreign key (bno) references book(bno)  
);
```

administrator:

```
create table administrator(  
    ano char(7),  
    aname varchar(20),  
    password char(6),  
    primary key (ano)  
);
```

(2) 连接界面

程序启动时的连接数据库界面，从编辑框中输入的内容获取数据库名、用户名和密码等信息，可以完成连接数据库的操作。连接完成后分成两种登录方式：普通用户和管理员。普通用户登录后直接进入图书查询界面（普通用户的唯一操作），而管理员登录需要输入数据库中已有的用户名和密码。



数据库连接

☐ Administrator ☒ User

Database Connection Settings

Driver: QMYSQL

Database: hzf

Username: root

Password: ●●●●●●●●●●

Hostname: localhost

Administrator: 1234567

Login Password: ●●●●●●

Login exit

(3) 功能选择界面

管理员登录后显示功能选择界面。功能选择分为如下图所示四个按钮



(4) 图书入库

图书入库界面分为两个部分：单本入库和多本入库。单本入库如图直接输入书的信息。多本入库则是把书的信息存在文件中，以两种方式插入：一是在输入框中输入文件路径，而是点选右边的文件选择按钮来选择本地文件。

(5) 图书查询界面

界面上方为六个给定的查询条件，点选复选框后可以在框中输入信息进行查询，底部两个按钮为查询排序按钮，可以根据某一属性名进行升序或降序排列显

示。若不选中任何复选框，则查询操作的结果是库中所有图书。

Query

☐ category

☐ title

☐ year

☐ press

☐ author

☐ price

~

Query

bno	category	title	press	year	author	price
-----	----------	-------	-------	------	--------	-------

Order By ASC

Order By DISC

(6) 借/还书

借还书在同一界面中，先在左方 **card number** 框中输入卡号，查询卡内已借书籍。然后在右方 **book number** 框中输入书号进行借书操作。还书操作与借书操作类似。

Borrow/Return

Card Number:

BookNumber:

Query

Borrow

Return

Corresponding Book Number:

(7) 借书证管理

界面打开时自动显示所有借书证用户信息。插入借书证时在 add card 栏内输入相关信息即可，删除借书证时则在下方 table 中选中一行按下 delete 按钮则可删除（前提是合法插入和合法删除）。

Add Card

Card Number:

Name:

Department:

Type: ☒ Teacher

☐ Graduate

☐ Undergraduate

☐ Owner

Add

State:

Delete Card

	cno	name	department	type
1	3130101	xsa	Ziyun	U
2	3130102	hzf	bifeng	G
3	3130236	Harden	White	G
4	3135216	Lebron	CaoGuangbiao	T

Delete

State:

三、详细设计

(1) 数据库连接

数据库连接界面包含一个名为 connectdlg 的类，其 public 成员函数包括构造和析构函数、返回输入框内容的函数、数据库连接函数 addConnection。

其 private 成员变量为功能选择界面所在的类 selectdlg 型的类变量 sdl。另有几个按钮触发的槽函数。

首先由界面中看到将密码框内容加密，运用 setEchoMode 函数：

```
ui->editPassword->setEchoMode(QLineEdit::Password);  
ui->loginPassword->setEchoMode(QLineEdit::Password);
```

可将密码框内容设置为黑色圆点（如下图）



用 drivers 函数获取可使用的驱动名，用 addItem 函数添入选择框内。

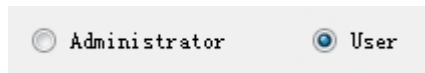
```
QStringList drivers = QSqlDatabase::drivers();  
ui->comboDriver->addItem(drivers);
```

按下连接按钮时执行连接函数，首先连接函数获取四个参数，驱动名 dbName、主机名 host、用户名 user 和密码 passwd。分别用下列四个函数设置后，调用 QSqlDatabase 类的 open() 函数连接数据库，若连接失败则返回值为 QSqlError 类的错误信息。

```
QSqlError connectdlg::addConnection(const QString &driver,  
const QString &dbName, const QString &host, const QString  
&user, const QString &passwd)  
{  
    QSqlError err;  
    QSqlDatabase db = QSqlDatabase::addDatabase(driver);  
    db.setDatabaseName(dbName);  
    db.setHostName(host);  
    db.setUserName(user);  
    db.setPassword(passwd);  
    if(!db.open())  
        err = db.lastError();  
}
```

```
return err;
}
```

按下连接按钮后，首先要根据单选框的内容判断是普通用户登录还是管理员登录，用 `isChecked()` 函数判断哪个框被选中。



若为普通用户登录，则直接显示图书查询界面 `qw`（本类的成员 `sdl` 的成员）。

```
if(ui->userRBtn->isChecked())
    sdl.qw.show();//直接进入查询界面
```

若为管理员登录，首先要进行数据库表格 `select` 操作，获取表中对应管理员账号的密码，用以判断管理员是否可以成功登录。`Select` 的方式为占位符查询，由 `QSqlQuery` 类的 `prepare` 函数设置占位符，`bindValue` 函数填充值，再由 `exec()` 函数执行 SQL 选择语句。

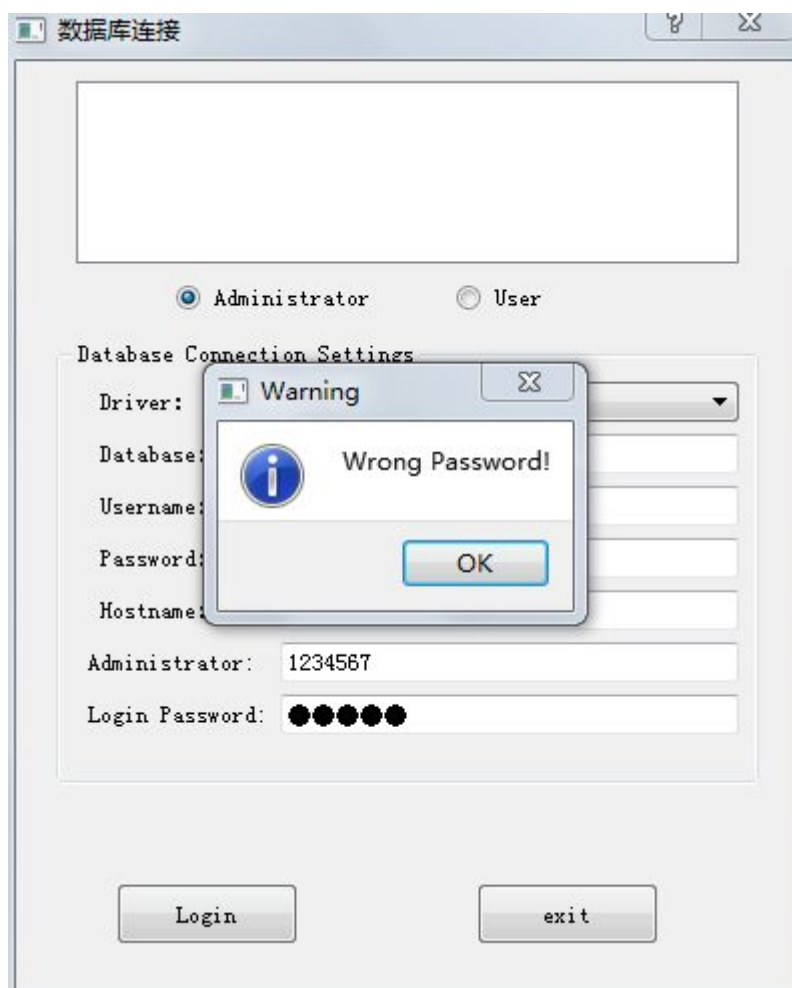
```
QSqlQuery query;
query.prepare("SELECT password FROM administrator where ano=?");
query.bindValue(0, adminID());//填充占位符
if(query.exec())
```

执行完后得到 `table`，只有一条记录（因为 `ano` 是 `primary key`）。因为执行完操作后 `query` 指向表头，故需调用 `QSqlQuery` 类的 `next()` 函数使其指向第一条记录，再调用 `value()` 函数以 `0` 为参数获取第一个信息（即管理员密码），以 `toString()` 函数转化为 `QString` 类字符串，储存在变量 `psw` 中。

```
if(query.exec())
{
    query.next();
    QString psw = query.value(0).toString();
```

判断输入框输入的密码与 `psw` 是否相等，若相等则登录成功，弹出成员变量 `sdl`

（功能选择界面），否则弹出错误信息对话框。



至此用 C++ 语言实现了数据库连接功能和登录。

（2）功能选择界面

此界面比较简单，仅包含了四个按钮，每按下一个按钮弹出不同的功能框（图书查询、借书还书、入库、借书证管理），每个按钮由对应的槽函数触发，原理同上述连接界面的按钮，故在此不再赘述。

（3）图书入库功能

无论是单本入库还是多本入库，入库前首先要进行一个判断：若插入的书已存在于数据库中，则作 update 操作，更新总量 total 和库存 stock；若不存在于数据库中，则以占位符方式执行 insert 操作。故对于管理员而言需要先做图书查询操作，确定该书号是否已存在于表中。而对于程序而言，可以加以适当的判断语句进行控制。以下分情况讨论：

A. 单本入库

自定义 inTable() 函数，判断输入框内的 bno 是否已存在于表格中。同上 select 后用 next 函数判断表格是否为空。

```
bool insertWindow::inTable(QString bno)
{
    QSqlQuery query;
    query.prepare("SELECT FROM book WHERE bno = ?");
    query.bindValue(0, bno);
    if(query.exec())
    {
        return query.next();
    }
    else
        return false;
}
```

若判断在表中，则执行 update 操作，total 和 stock 各加 1。

```
if(inTable(getBno()))
{
    query.prepare(
"UPDATE TABLE book SET total = total + 1 and stock = stock + 1 WHERE bno = ?");
    query.bindValue(0, getBno());
    if(query.exec())
        ui->stateLbl->setText(tr("Insert Successfully.));
    else
        ui->stateLbl->setText(tr("Failed insertion!"));
}
```

若不在表中，则执行正常的插入语句。

```

else
{
    query.prepare("INSERT INTO book VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)");
    //填充占位符
    query.bindValue(0, getBno());
    query.bindValue(1, getCategory());
    query.bindValue(2, getTitle());
    query.bindValue(3, getPress());
    query.bindValue(4, getYear());
    query.bindValue(5, getAuthor());
    query.bindValue(6, getPrice());
    query.bindValue(7, getTotal());
    query.bindValue(8, getStock());
    //检查插入记录操作是否成功
    if(query.exec())
        ui->stateLbl->setText(tr("Insert Successfully.));
    else
        ui->stateLbl->setText(tr("Failed insertion!));
}

```

B. 多本入库

对于多本入库，SQL 语句实现方式同上，多出的要求是需要读取文件操作。

Qt 中有文件操作相关的类 QFile 类。对每一行的记录使用 QFile 类的成员函数 getLine() 函数读入，同单条记录一样，判断第一个字符串 bno 是否存在于表中，存在则 Update，不存在则 insert。具体功能实现在下图的注释中详细说明：

```

void insertWindow::on_mulInsertBtn_clicked() //插入多条记录
{
    QFile file(getFileName()); //获取文件名并初始化对象file
    if(file.open(QIODevice::ReadOnly)) //以只读方式打开文件
    {
        QSqlQuery query; //QSqlQuery类对象用于插入记录
        char buffer[2048]; //读取一行内容
        bool flag = true;
        while(!file.atEnd()) //未到达文件结尾
        {
            //返回读取一行的子节数，然后指针指向下一行文本
            qint64 lineLen = file.readLine(buffer, sizeof(buffer));
            if(lineLen != -1) //读取失败readLine()返回-1，不等则读取成功
            {
                QString sqlStr = "INSERT INTO book VALUES";
                sqlStr += buffer;
                if(!query.exec(sqlStr)) //插入操作失败
                {
                    ui->mulStateLbl->setText("Failed insertion!");
                    flag = false;
                }
            }
            if(flag) //flag始终未被设置成0,说明全部插入成功
                ui->mulStateLbl->setText("All Insertions Successful!");
        }
        else //打开文件失败
            ui->mulStateLbl->setText("Fail to open the file !");

        file.close(); //关闭文件
    }
}

```

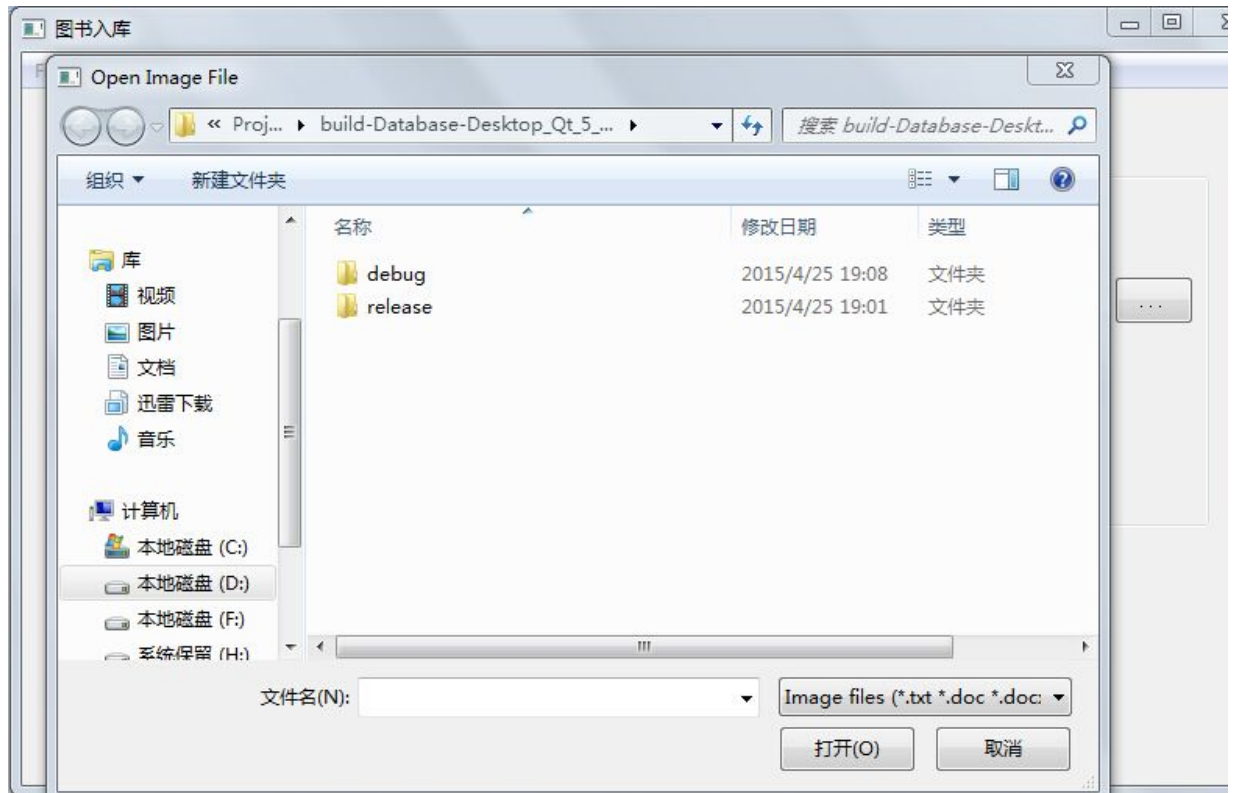
对于文件选择按钮的实现，通过 `QFileDialog` 类的成员函数

`getOpenFileName()` 来弹出文件选择窗口，可以供用户进行手动选择文件。

```

void insertWindow::on_selectFileBtn_clicked()
{
    //通过标准文件对话框获得所需要文件的文件名功能
    QString file = QFileDialog::getOpenFileName(this,
        "Open Image File",
        ".",
        "Image files (*.txt *.doc *.docx *.xls *.xlsx);;All files (*.*)");
    ui->editFileName->setText(file); //将文件名显示在文件名输入框内
}

```



(4) 图书查询功能

图书查询需要实现按单个或多个属性条件的查询，故在 UI 界面中引入了一个新的控件：复选框。实现方式是先初始化一条 SQL 的 select 查询语句，并默认以 bno 的 ASC（升序）排列结果。当选中某一个复选框的时候定义槽函数触发事件：在 SQL 语句末尾插入 WHERE 子句，从而达到实时按属性条件查询的功能。而查询内容显示在空间 tableView 中。



未选中复选框时，默认输出所有图书信息：


```
//形成SQL select语句字符串
const int count = countSelected();//获取被选中的复选框个数
int flag = count;//标志

if(count==0)//没选中任何复选框，则显示所有书的信息
    qStr = "SELECT * FROM book";
```

flag 变量获取选中的复选框个数，用以作为插入子句个数的标志。每插入一次子句 flag 自减 1。

以 category 属性为例，当选中此框时，在 qStr 字符串变量后面插入 WHERE 子句：

```
qStr = "SELECT * FROM book WHERE ";

if(ui->cateCheckBox->isChecked())
{
    qStr += "category = '";
    qStr += getCategory();
    qStr += "'";//第二个单引号
    if(flag!=1&&flag!=0){
        qStr += " and ";
        flag--;
    }
}
```

flag 的作用就是当复选框的内容全部插入完成后则不再需要插入 and，插入完成后末尾再加入 ORDER BY bno ASC 作为默认排序标准。

下图为实验结果，当复选框一个也不选的时候，输出了所有图书信息。

Query

☐ category ☐ title ☐ year ☐ press ☐ author ☐ price

~ ~

Query

	bno	category	title	press	year	author	price
1	1340	my certain	yes	cover	1995	James Harden	100.28
2	1341	my certain	no	cover	1995	James Harden	100.28
3	1342	my certain	yes	cover	1995	James Harden	100.28
4	1343	my certain	yes	cover	1995	James Harden	100.28
5	2650	my certain	yes	cover	1995	JamesHarden	100.28

当點選 title 并填入 no 时，结果只有一条记录：

Query

☐ category ☒ title ☐ year ☐ press ☐ author ☐ price

 no

~ ~

Query

	bno	category	title	press	year	author	price
1	1341	my certain	no	cover	1995	James Harden	100.28

设置 title 为 yes 后选中 bno，点击 order by disc，则记录按 bno 的降序排列。其实现方式如下：

(1) 用 QModelIndexList 类的成员函数 empty 判断是否选中一列记录，再调用 QModelIndex 类的 column() 函数获取选中列号，用 headerData() 函

数返回表头标题（属性名），然后将属性名插入到 SQL 语句的 order by 后，实现了按某个属性排序的功能。

```
QModelIndexList selection = ui->tableView->selectionModel()->selectedColumns(0);
if(!selection.empty())//判断是否在表中选中一条记录
{
    QModelIndex idIndex = selection.at(0);
    //idIndex.column();获取所选中的当前列号
    QString orderStr = qStr;
    QString s = model->headerData(idIndex.column(), Qt::Horizontal).toString();
    orderStr += (" ORDER BY " + s + " DESC");
    QSqlQuery query;
    if(query.exec(orderStr))
        displayTable(orderStr);
    else
        qDebug() << "Error";
}
```

实验结果如下：

Query

☐ category
☒ title
☐ year
☐ press
☐ author
☐ price

~

Query

	bno	category	title	press	year	author	price
1	2650	my certain	yes	cover	1995	JamesHarden	100.28
2	1343	my certain	yes	cover	1995	James Harden	100.28
3	1342	my certain	yes	cover	1995	James Harden	100.28
4	1340	my certain	yes	cover	1995	James Harden	100.28

Order By ASC

Order By DISC

升序排列的原理与降序完全相同，只需要修改 disc 为 asc 即可。至此图书查询的功能已完全实现。

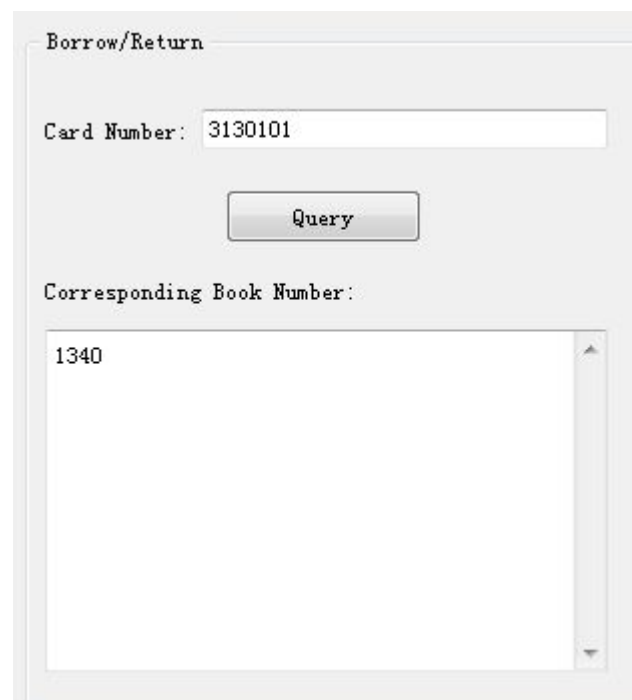
(5) 借/还书功能

首先输入借书证卡号要显示已借书籍的信息。转化为 SQL 语言则为在借书

记录的表 borrow 中搜出 return_date 为 Null 的记录的属性 bno。用控件

textBrowser 的 append 函数（结尾自带换行符）输出在表中，结果如图

```
void brwWindow::on_cardQueryBtn_clicked() //查询按钮触发
{
    ui->textBrowser->clear();
    QSqlQuery query;
    query.prepare("SELECT bno FROM borrow WHERE cno = ? and return_date is null");
    query.bindValue(0, getCno());
    if(query.exec())
    {
        while(query.next())
            ui->textBrowser->append(query.value(0).toString()); //在文本框显示搜索记录
    }
}
```



然后输入书号，注意要先用输入的 bno 在 book 表中查询该书是否有余量。如果还有库存 ($stock > 0$)，则对 book 表做 update 操作，把对应 bno 记录的 stock 减 1。然后更新借书记录 borrow，将输入框中的 cno 和 bno 插入，而 borrow_date 由 QDateTime 类的 currentDateTime() 函数获取实时本地时间而得到，toString() 函数设置时间格式。

```
QString brwWindow::getTime() //用于获取当前系统时间
{
    QDateTime time = QDateTime::currentDateTime(); //
    return time.toString("yyyy-MM-dd hh:mm:ss ddd");
}
```

```
query.next();
int oldStock = query.value(0).toInt(); //将所选记录(库存数)转化为int型
if(oldStock > 0) //若还有库存
{
    ui->stateText->append("Successfully Borrow!");
    //更新库存数量数据
    query.prepare("UPDATE book SET stock = stock - 1 WHERE bno = ?");
    query.bindValue(0, getBno());
    if(!query.exec())
        ui->textBrowser->append("Error!");
    //更新借书记录
    query.prepare("INSERT INTO borrow VALUES(?, ?, ?, null)");
    query.bindValue(0, getCno());
    query.bindValue(1, getBno());
    query.bindValue(2, getTime());
    if(!query.exec())
        ui->textBrowser->append("Update Error!");
}
```

如果没有库存 (stock = 0)，首先报错误信息，然后需要返回 borrow

表中最近的还书时间。这个功能由 SQL 语句的 max 来实现：

```
ui->stateText->append("No Stock!"); //输出无库存
ui->stateText->append("The latest return time: ");
query.prepare("SELECT max(return_date) FROM borrow WHERE bno = ?");
query.bindValue(0, getBno());
if(query.exec())
{
    query.next();
    ui->stateText->append(query.value(0).toString());
}
else
    ui->stateText->append("Selection Error!");
```

而对于还书操作，card number 的操作与借书操作是完全一样的，而还书

按钮按下后，首先要判断输入的书号 bno 是否存在于借书记录 borrow 表中，显然需要查询一遍 borrow 表：

```
//检查输入的书号是否存在于表中
bool hasLent = false;
if(query.exec())
{
    while(query.next())
    {
        if(query.value(0).toString() == getBno()){
            hasLent = true; //若表中有书与之匹配
            break;
        }
    }
}
```

若存在于表中，则读取本地时间更新到借书记录中 return_date 为 null 的部分，并更新该书库存+1。若不存在，则直接报错。

```
if(hasLent) //存在于表中
{
    //更新borrow记录中的return_date
    query.prepare("UPDATE borrow SET return_date = ? WHERE bno = ? and return_date is null");
    query.bindValue(0, getTime());
    query.bindValue(1, getBno());
    if(query.exec())
    {
        //更新库存数+1
        query.prepare("SELECT stock FROM book WHERE bno = ?"); //显示该借书证所有已借书籍
        query.bindValue(0, getBno());
        if(query.exec())
        {
            query.next();
            ui->stateText->append("Successfully Return.");
            query.prepare("UPDATE book SET stock = stock + 1 WHERE bno = ?"); //更新库存数量数据
            query.bindValue(0, getBno());
            if(!query.exec())
                ui->textBrowser->append("Error!");
        }
    }
}
```

(6) 借书证管理

该界面显示借书证信息需要建立 tableView 和 QStandardItemModel。

设置表头的函数如下：

```

void cardmanage::setupModel()
{
    model = new QStandardItemModel(4,4,this);
    //设置表头
    model->setHeaderData(0,Qt::Horizontal,QObject::tr("cno"));
    model->setHeaderData(1,Qt::Horizontal,QObject::tr("name"));
    model->setHeaderData(2,Qt::Horizontal,QObject::tr("department"));
    model->setHeaderData(3,Qt::Horizontal,QObject::tr("type"));
    //设置表格内容不可编辑
    ui->tableView->setEditTriggers(QTableView::NoEditTriggers);
}

```

添加一个借书证时，只需从编辑框中获取四个属性信息，执行一个 insert 操作将借书证插入，然后在 tableView 中调用自定义 addARow() 函数实时更新记录。

```

void cardmanage::on_addBtn_clicked()
{
    QSqlQuery query;
    query.prepare("INSERT INTO card VALUES(?, ?, ?, ?)");
    query.bindValue(0,getCno());
    query.bindValue(1,getName());
    query.bindValue(2,getDept());
    query.bindValue(3,selectedType);
    if(query.exec()){
        ui->stateLbl->setText(tr("Success")); //借书证添加成功
        addARow();
    }
    else
        QMessageBox::information(this, tr("Warning"),tr("Failed!"));
}

```

```

void cardmanage::addARow()
{
    int row = model->rowCount(QModelIndex()); //获取要插入的行号
    model->insertRows(row, 1, QModelIndex()); //插入一行内容
    model->setData(model->index(row, 0, QModelIndex()), getCno());
    model->setData(model->index(row, 1, QModelIndex()), getName());
    model->setData(model->index(row, 2, QModelIndex()), getDept());
    model->setData(model->index(row, 3, QModelIndex()), selectedType);
}

```

删除一个借书证则需要注意一个潜在问题：当该借书证仍在借书中，书还未归还时，该借书证不可删除。于是执行删除操作之前，应该调用自定义函数

onBorrow() 来检查 cno 在 borrow 表中是否存在 return_date 为 Null 的情况，如果有的话该借书证不可删除。几个核心函数如下：


```
bool cardmanage::onBorrow(QString cno)//该卡是否有书未还
{
    QSqlQuery query;
    query.prepare("SELECT * FROM borrow WHERE cno = ? AND return_date is null");
    query.bindValue(0, cno);
    if(query.exec())
    {
        if(query.next())
            return true;
    }
    return false;
}
```

```
void cardmanage::on_delBtn_clicked()
{
    QModelIndexList selection = ui->tableView->selectionModel()->selectedRows(0);
    if(!selection.empty())//判断是否在表中选中一条记录
    {
        QModelIndex idIndex = selection.at(0);
        QString cno = idIndex.data().toString();

        if(onBorrow(cno)){
            ui->delMessage->setText(tr("On borrow! Fail to Delete!"));
            return;
        }

        model->removeRow(idIndex.row()); //删除表格显示记录，但不删除实际记录
        //真正的删除
        QSqlQuery query;
        query.prepare("DELETE FROM card WHERE cno = ?");
        query.bindValue(0, cno);
        if(query.exec())//执行删除操作成功
            ui->delMessage->setText(tr("Delete Successfully.));
        else
            ui->delMessage->setText(tr("Fail to Delete!"));
    }
}
```

当我们用 3130103 卡号借了一本书后，将其 delete，结果显示不可删除，该功能实现。而 3130101 删除是成功的。

	cno	name	department	type	
1	3130101			T	
2	3130103			T	

On borrow! Fail to Delete!

Delete Card

	cno	name	department	type	
1	3130103			T	

Delete

Delete Successfully.

至此，该数据库系统的全部所需功能都已通过 Qt 平台 C++语言实现。