

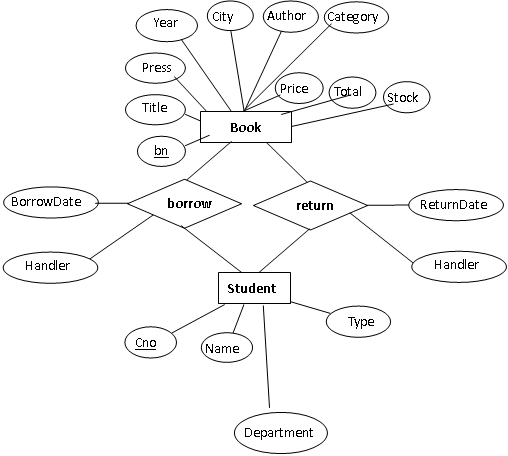
**数据库系统实验报告**

|  |  |
| --- | --- |
| 作业名称： | 数据库程序设计 |
| 姓 名： | 林馨怡 |
| 学 号： | 3170101791 |
| 电子邮箱： | xin751687372@163.com |
| 联系电话： | 17326088652 |
| 指导老师： | 孙建伶 |

2019年 4 月 17 日

**实验名称**

1. **实验目的**
   1. 掌握数据库应用开发程序设计方法
2. **系统需求**
   1. 实现图书查询，包括查询全部和对书编号、书名的查询
   2. 实现图书借阅，借阅时输入借阅证编号，借阅后相应的书的库存量减少
   3. 实现图书归还，归还时输入借阅证编号，归还后相应的书的库存量增加
   4. 实现图书插入，插入时需输入图书的所有信息，系统依据图书编号先判断库中是否已有该书信息，如有则更改库中该书库存使其加一，否则添加一条新书的记录
   5. 实现借阅证的管理，包括依据借阅证编号进行的借阅证的删除、插入和修改，其中借阅证的修改可以实现修改借阅证姓名、借阅证所属系，以及借阅证类型，但无法修改借阅证编号
3. **实验环境**
   1. 数据库管理系统：MySQL
   2. 开发语言：C
   3. 开发工具：Visual Studio 2017
   4. 交互界面：控制台字符界面
4. **系统设计及实现**
   1. **实体之间的关系E-R图**



* 1. **数据库逻辑结构设计**

create table book

(bno char(10),

category varchar(10),

title varchar(20),

press varchar(20),

year int,

author varchar(10),

price decimal(7,2),

total int,

stock int,

primary key(bno));

create table card

(cno char(7),

name varchar(10),

department varchar(40),

type char(1),

primary key(cno),

check(type in('T','S')));

create table borrow

(cno char(7),

bno char(8),

borrow\_date int,

return\_date int,

primary key(cno,bno),

foreign key (cno) references card(cno),

foreign key (bno) references book(bno));

* 1. **程序运行结果场景以及截图说明**

1. **图书入库和修改：**
   * + 1. **检验库中是否已存在编号相同的书的信息**

先调用SQLAllocStmt函数分配语句句柄，再输入想要插入的图书信息。

至于为什么要分配句柄，是因为通常使用ODBC API开发数据库应用程序需要经过如下步骤：

· 连接数据源。

· 分配语句句柄。

· 准备并执行SQL语句。

· 获取结果集。

· 提交事务。

· 断开数据源连接并释放环境句柄。

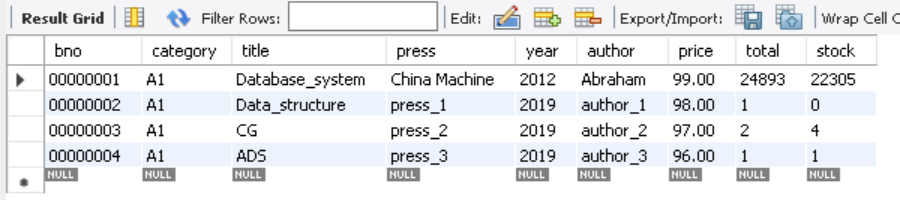
为了连接数据源，必须要建立一个数据源连接的环境句柄。通过调用SQLAllocEnv函数实现对环境句柄的分配，在ODBC 3.0里，这个函数已经被SQLAllocHandle取代，但是熟悉ODBC API的开发人员还是习惯用这个函数建立环境句柄，因为VC++开发平台有一个映射服务，这个服务将程序代码对函数SQLAllocEnv的调用转向对函数SQLAllocHandle的调用。

句柄是指向一个特殊结构的指针，而环境指的是驱动程序管理器需要为该驱动程序存储的有关系统和数据源的一般信息。由于这个时候还没有建立同数据源的连接，驱动程序还并不知道该使用哪一个驱动程序来完成这个任务，所以这个任务只能由驱动程序管理器来完成，利用这个环境句柄保留信息直到被使用。

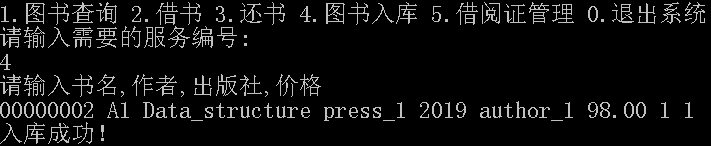
句柄是指向一个特殊结构的指针，而环境指的是驱动程序管理器需要为该驱动程序存储的有关系统和数据源的一般信息。由于这个时候还没有建立同数据源的连接，驱动程序还并不知道该使用哪一个驱动程序来完成这个任务，所以这个任务只能由驱动程序管理器来完成，利用这个环境句柄保留信息直到被使用。

接下来，我将要执行的SQL语句select以字符串形式赋值给指针query,在query语句中，未知的要查询的图书编号用“？”来代替，绑定参数后调用SQLExecute函数执行该SQL语句。如果找到了该书的编号，就用update语句将该书的库存加一。

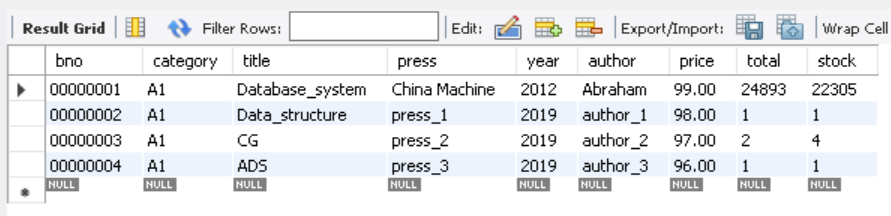
**插入前的book表**



**插入的图书信息**



**插入后的book表**

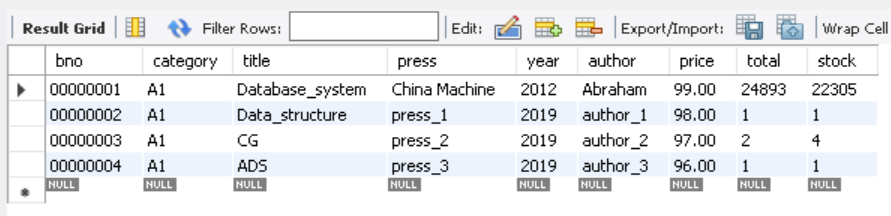


可以看到，编号为00000002的图书库存比原来多了1。

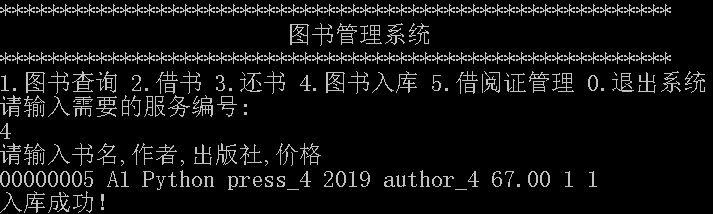
* + - 1. **库中不存在编号相同的书的信息**

先调用SQLPrepare函数，再调用SQLBindParameter函数对每一个属性的参数进行绑定，然后执行插入语句insert，将所有信息输入到库中。

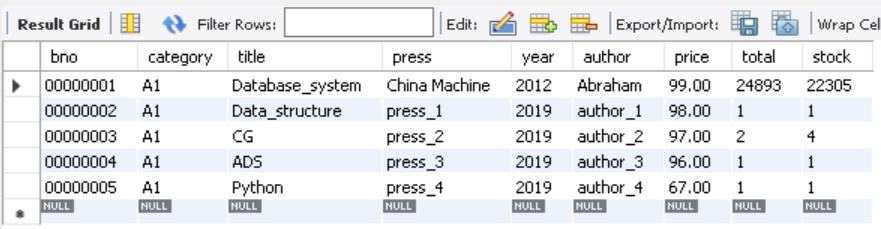
**插入前的book表**



**插入的图书信息**



**插入后的book表**



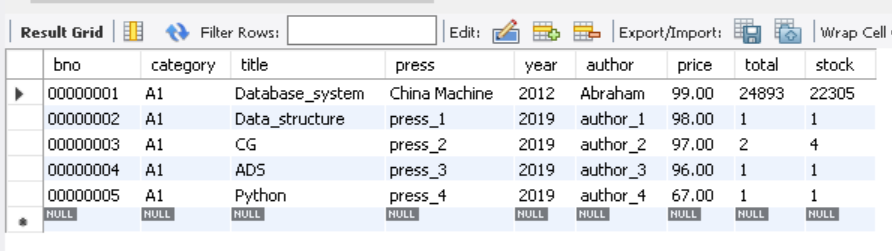
可以看到，编号为00000005的图书信息被成功插入book表中。

1. **借书：**

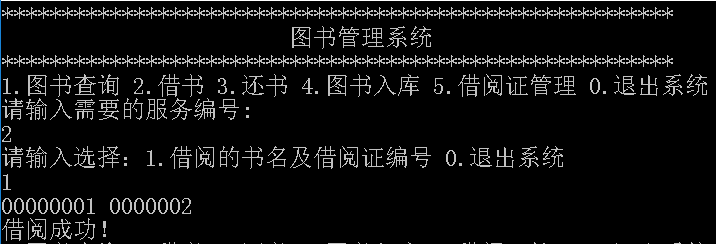
**检验要借阅的图书库存是否为0**

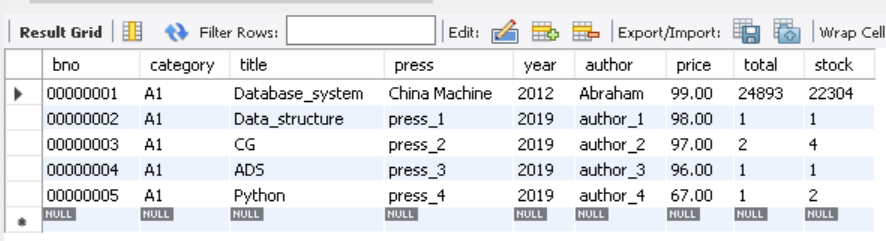
输入要借阅的图书编号和借阅证编号。先调用SQL的select语句查询该编号的图书库存是否为0。如果库存不为0，就调用update语句将该书的库存减一，并显示借阅成功。

**借阅前的book表**



**借阅图书的语句**



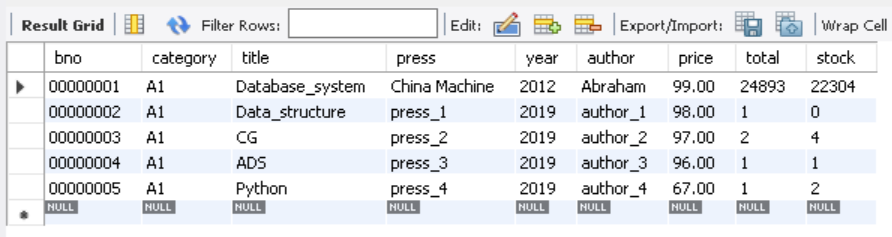
**借阅后的book表**

可以看到，编号为00000001的图书库存少了1。

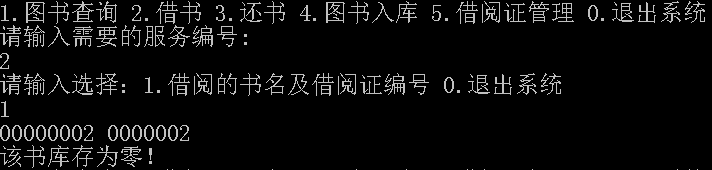
**要借阅的图书库存为0，借阅失败**

如果查询后用SQLFetch函数取出来的元组库存为0，则显示借阅失败。

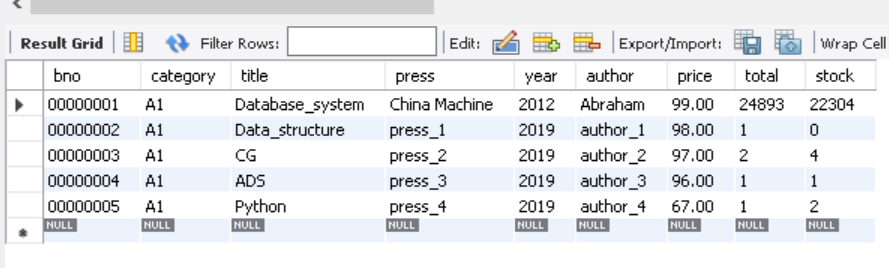
**借阅前的book表**



**借阅图书的语句**



**借阅后的book表**

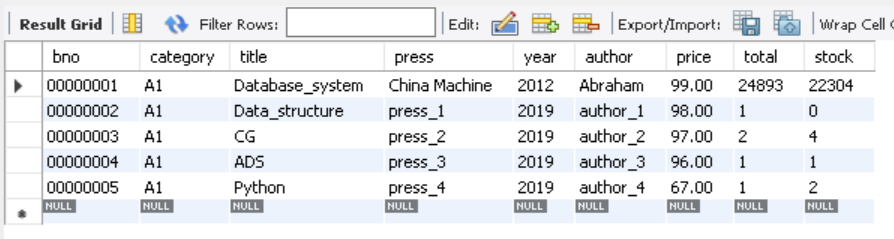


可以看到，由于编号为00000002的图书借阅前库存为0，无法借阅该书，显示借阅失败。

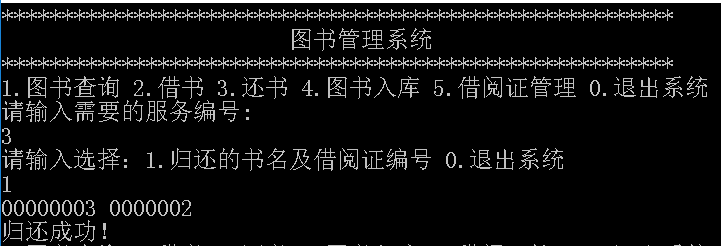
1. **还书：**

输入要归还的书的编号和借阅证编号，如果归还成功就显示归还成功。

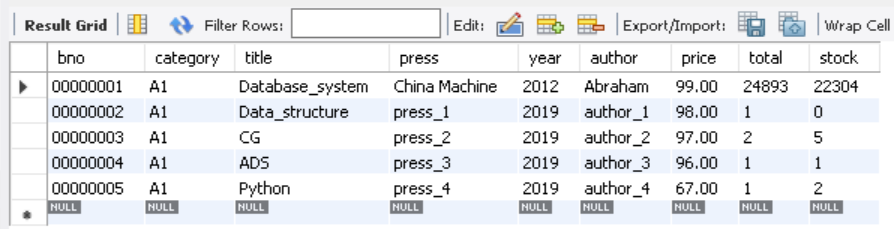
**归还前的book表**



**归还图书的语句**



**归还后的book表**

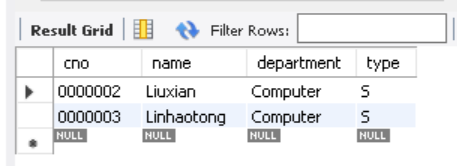


可以看到，编号为00000003的图书库存比原来多了1，显示归还成功。

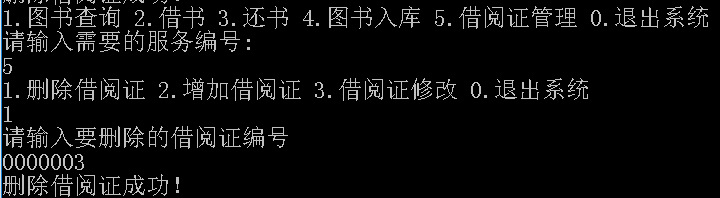
1. **借书证管理：**
2. **借书证的删除**

输入要删除的借书证编号，并调用SQL的delete语句将该编号的借书证删除，并显示删除成功。

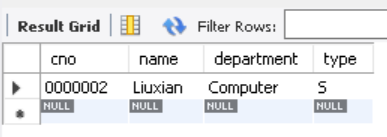
**删除前的card表**



**删除借阅证语句**



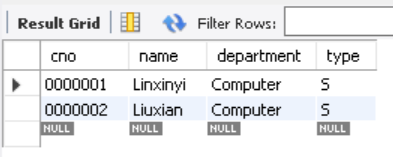
**删除后的card表**



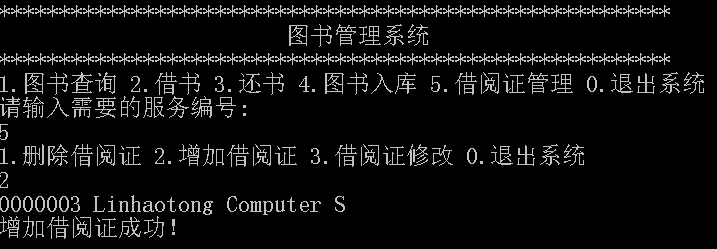
1. **借书证的增加**

输入要添加的借书证编号、姓名、系名和类型，使用预备语句并执行SQL的insert语句对card表进行插入。

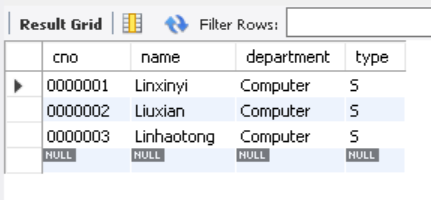
**插入前的card表**



**插入借阅证的语句**

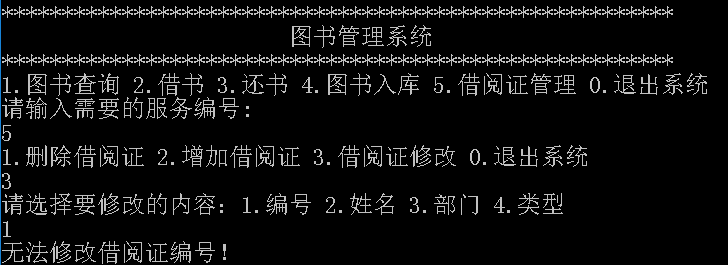


**插入后的card表**



1. **借书证的修改（编号）**

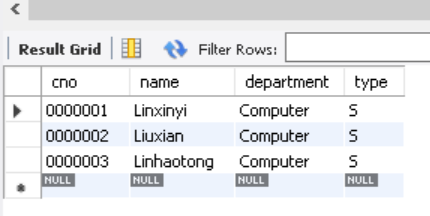
借书证的编号应无法修改，因此不让输入要修改的借书证编号。



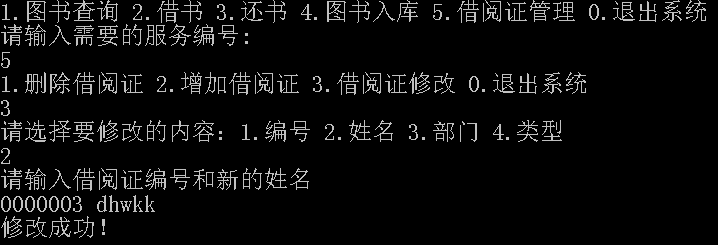
1. **借书证的修改（姓名）**

输入要修改的借阅证的编号和新的姓名，再调用SQL的update语句对该编号的借阅证进行修改。

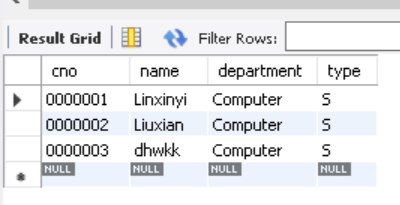
**修改前的card表**



**修改借阅证的语句**



**修改后的card表**

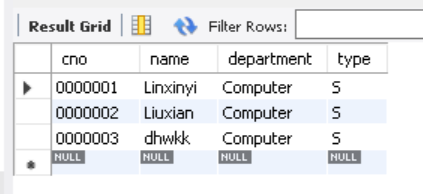


可以看到，编号为0000003的借阅证的姓名变成了dhwkk，修改成功。

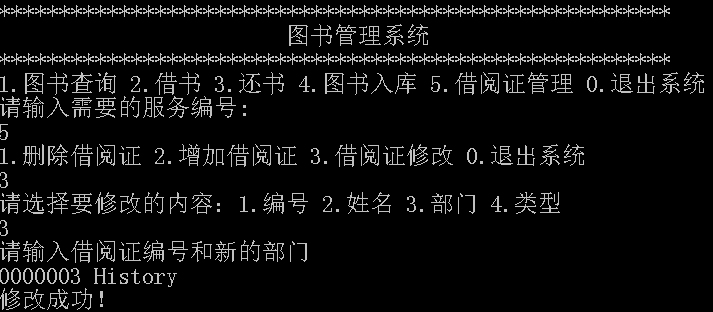
1. **借书证的修改（系名）**

输入要修改的借阅证的编号和新的系名，再调用SQL的update语句对该编号的借阅证进行修改。

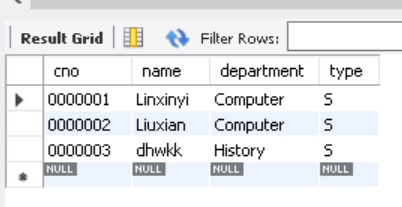
**修改前的card表**



**修改借阅证的语句**



**修改后的card表**

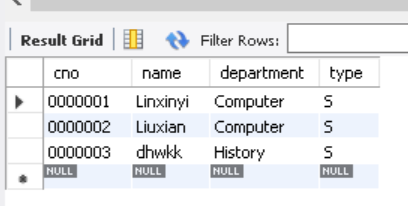


可以看到，编号为0000003的借阅证系名改为了History，修改成功。

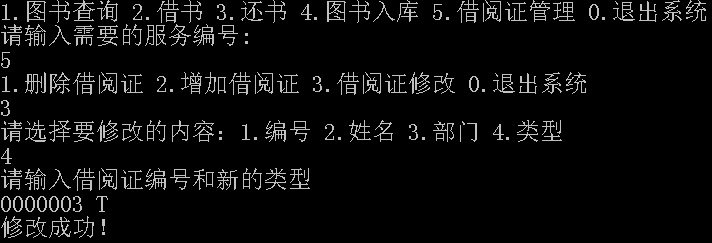
1. **借书证的修改（类型）**

输入要修改的借阅证的编号和新的系名，再调用SQL的update语句对该编号的借阅证进行修改。

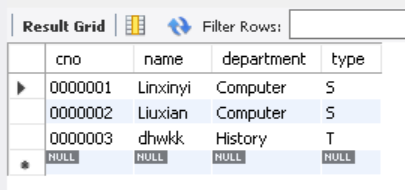
**修改前的card表**



**修改借阅证的语句**



**修改后的card表**



可以看到，编号为0000003的借阅证的类型变为了T，修改成功。

1. **图书查询：**
2. **查询全部**

先调用SQLAllocStmt函数分配语句句柄，再输入想要插入的图书信息，将要执行的SQL语句select以字符串形式赋值给指针query，然后调用SQLExecute函数执行该SQL语句。调用SQLBindCol函数将每一列属性与预设变量一一绑定，并不断用SQLFetch函数取出每一个表book中的元组，将每一列属性的值打印出来。

**查询全部语句**



1. **按书名查询**

输入要查询的图书的名字进行查询。

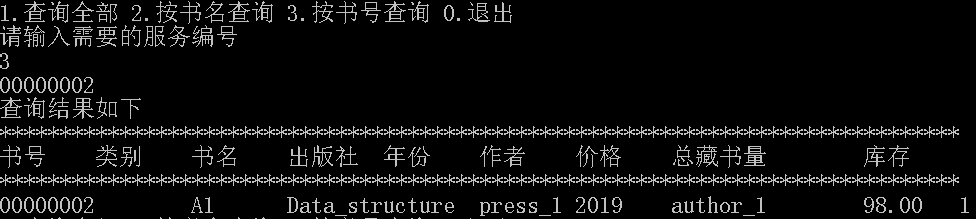
**按书名查询语句**



1. **按书号查询**

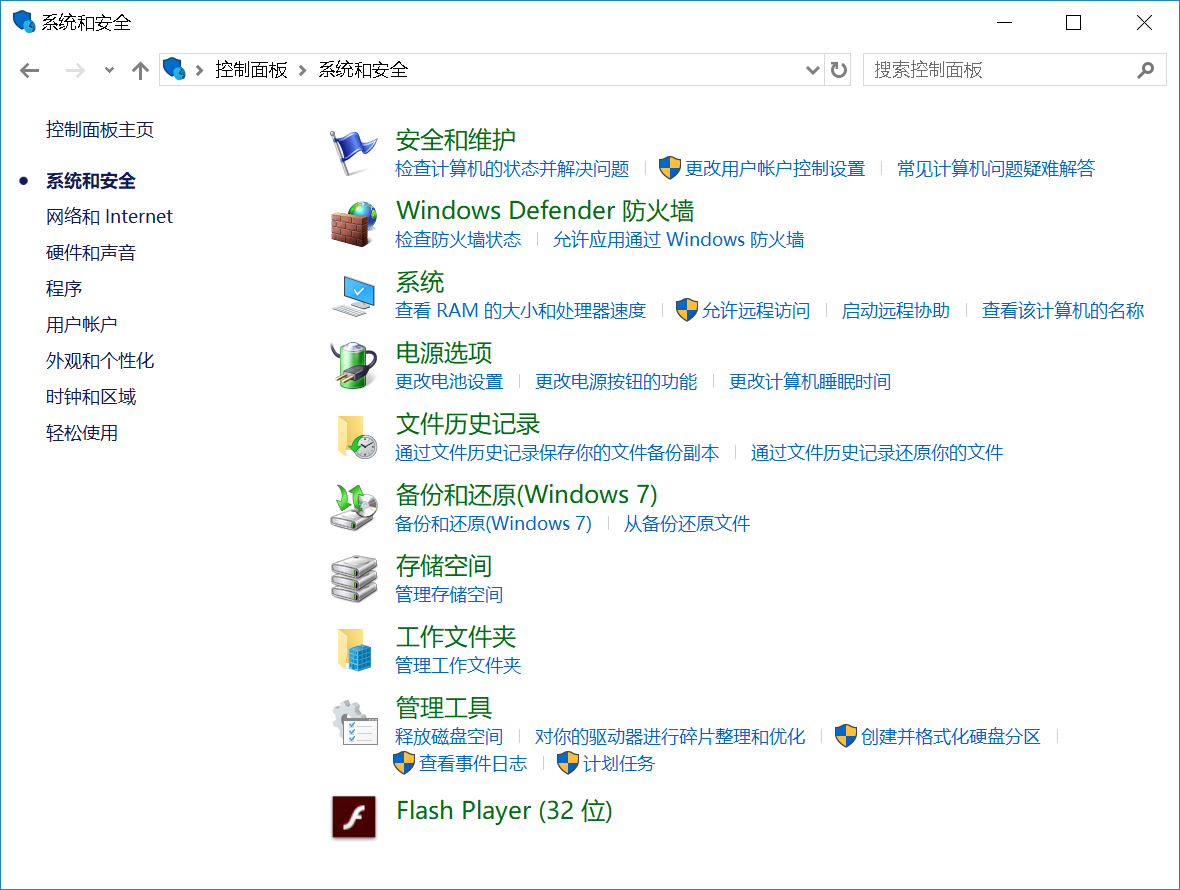
输入要查询的图书的编号进行查询。

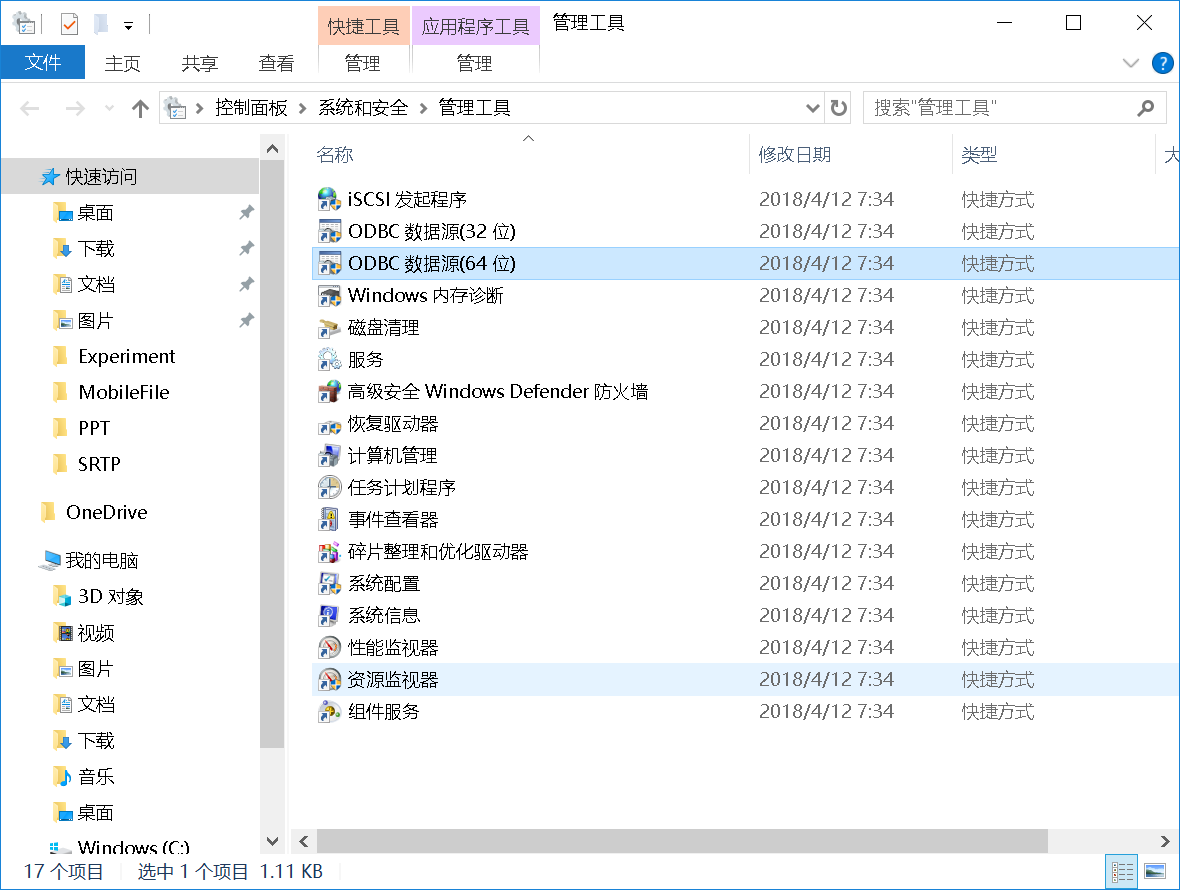
**按书号查询语句**

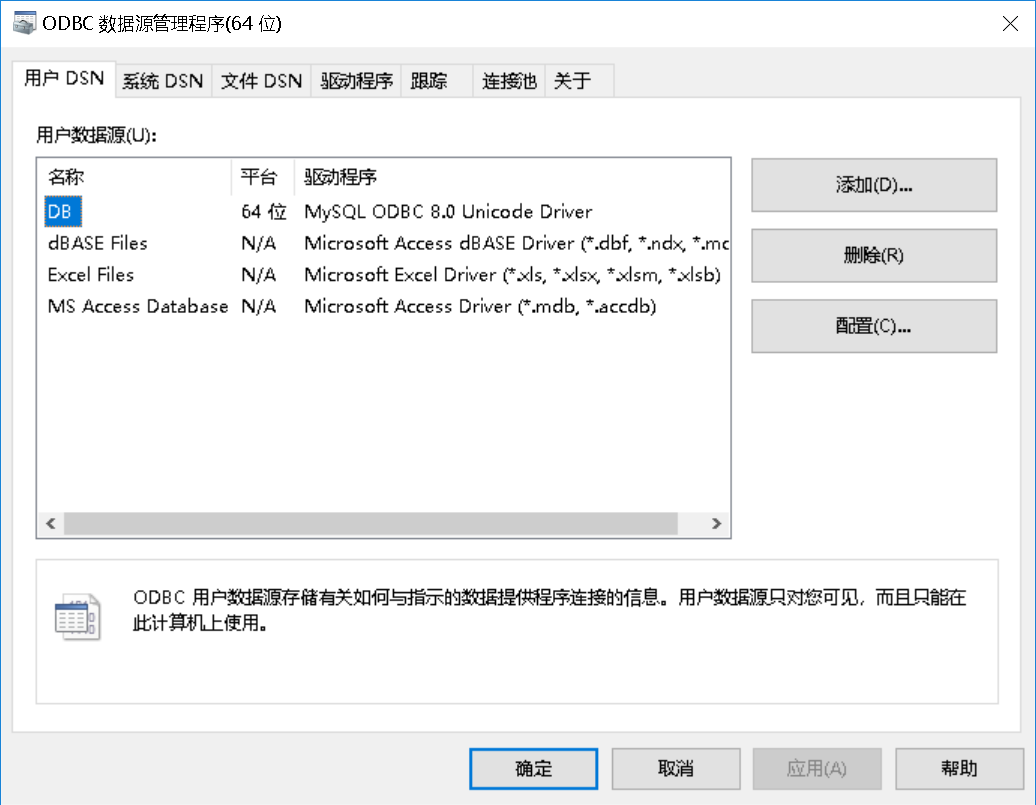


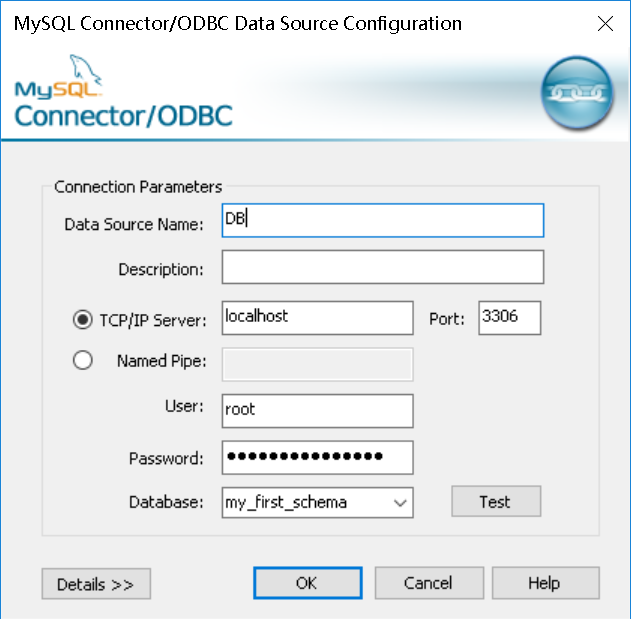
1. **遇到的问题及解决方法**
   1. **配置环境问题**

一开始我一直连不上数据库，我在控制面板里添加了数据库，步骤如下：

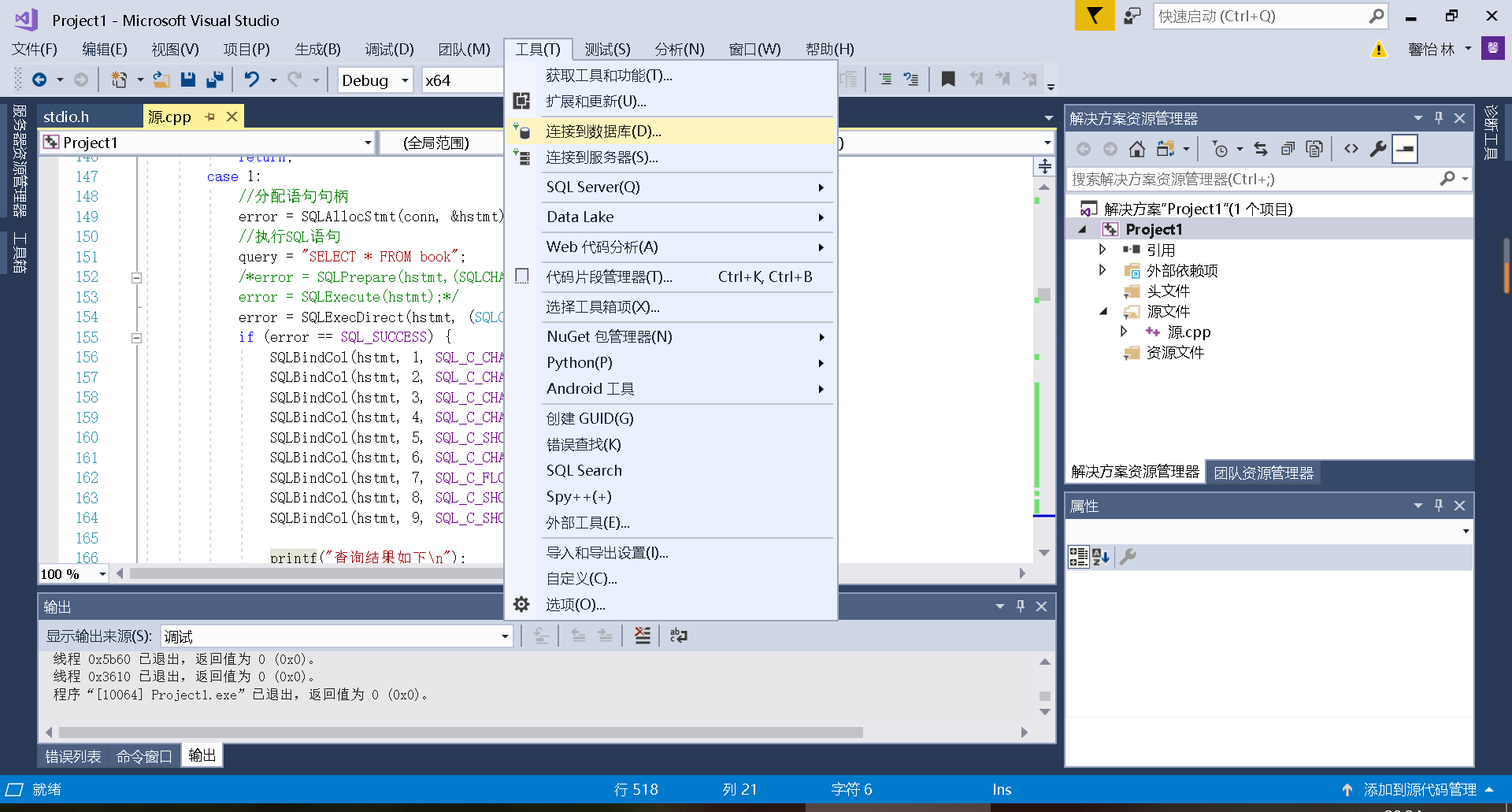




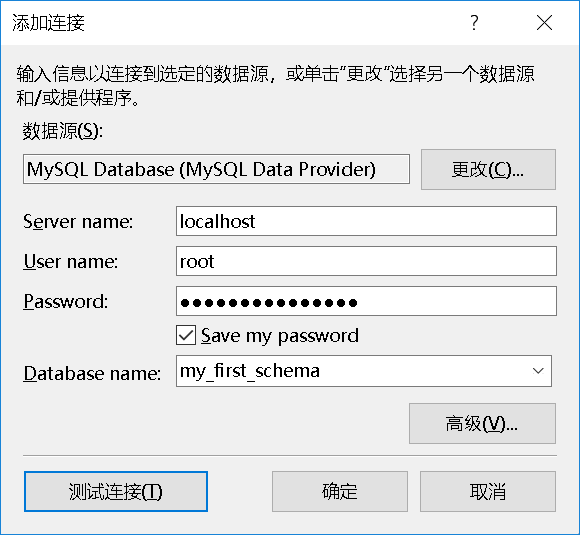


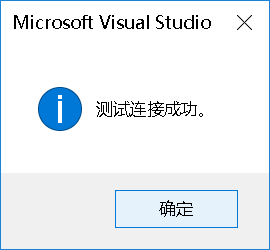


接下来，我在MySQL Workbench里新建了一个模式命名为my\_first\_schema，在VS里连接到数据库









但是，当我建表的时候，发现依然无法连接到数据库，建表一直不成功。

后来，我发现我应该把Debug改成x64，因为x64是64位，而x86是32位，如果我用32位的编译器，就无法连接到64位的数据库了。



改完之后，成功连接上数据库。

* 1. **字符串的输入**

我在输入数据时，需要连续输入多个字符串，并将每个字符串存储到相应的变量中，这样才能完成SQL语句的执行。但是无论我用gets\_s函数、scanf\_s函数还是sscanf\_s函数，都无法正确地把变量读入并存储进来。于是，我最后用了cin，成功将变量读入并存储。

1. **总结**

我觉得通过这次实验，我更加了解了ODBC的运行机制，也知道了该如何对一个数据模式进行具体的数据库操作。ODBC确实可以完成构建一个数据库的工作，但是可能并不是最佳选择。

同时，我也更加熟悉了Visual Studio的使用。Visual Studio的scanf\_s和sscanf\_s有一定的优势，但在多个连续字符串的读入和存储上有很大的局限性，而这些局限性用cin就可以成功避免。

在写ODBC时，要注意的是不能将SQL语句写错，如果写错了，执行结果会变成-1，而刚开始很难发现是哪里写错了，只能知道程序不能正确运行。

这次写一个大程序确实让我学到了很多，也对数据库本身有了更深的理解，我也希望通过后续的学习更加深入地了解到数据库的其他操作，更好地实现一个完整的数据库。