《数据库系统》课程设计报告

题目	图书销售管理系统

姓名	学号	班级	分工
冯大纬	18340040	计科	进货、退货、加载数据库、备份数据库、日志
邓明昱	18340034	计科	统计、销售、库存、供应商查询、交易记录查询

提交时间：2021 年 1 月 5 日

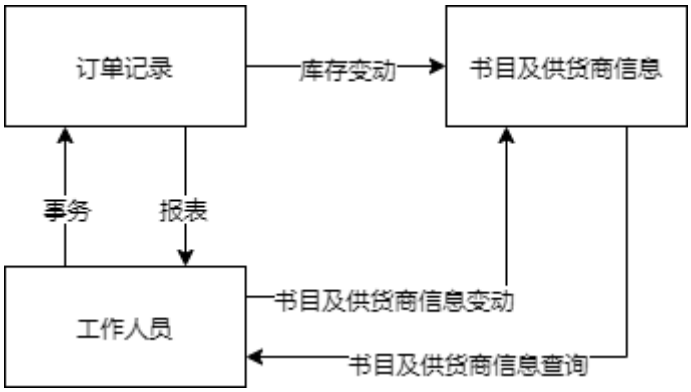
1.开发环境与开发工具

- 操作系统：Windows 10 Ver.1903
- DBMS：MYSQL 8.0.20
- 开发工具：Python 3.8.2

2.系统需求分析

书店工作人员需要通过图书销售管理系统完成书店的日常图书销售管理。考虑到日常销售管理设计的内容，系统设计的数据为书目的信息和销售的记录。具体操作上应当可以方便查询书目信息（库存、价格等）、销售订单记录（出入库、退货）、生成报表。

系统基本模型数据流图



系统数据字典

数据：

```
##### book：

    描述：书目编号、库存、定价信息

    数据项：
        book_id：
            主键
            描述：书目唯一标识的数字编号
            定义：int
```

- book_name：
- 描述：书目名字
- 定义：char(64)

- num：
- 描述：库存量
- 定义：int

- price：
- 描述：本店定价
- 定义：int

supplier:

- 描述：供货商订单信息

- 数据项：
- sup_id：
- 主键
- 描述：供货商提供的供应单唯一标识的数字编号
- 定义：int

- sup_book_id：
- 描述：这份供应单提供的书目编号
- 定义：int

- sup_price：
- 描述：供货商的供应价
- 定义：int

trade：

- 描述：订单信息，书籍库存变化

- 数据项：
- trade_id：
- 主键
- 描述：交易记录唯一标识的数字编号
- 定义：int

- book_id：
- 描述：这条交易记录对应的书目编号
- 定义：int

- trade_type：
- 描述：交易类型（in 入库，out 出库，buy 进货）
- 定义：enum('in','out','buy')

- trade_num：
- 描述：交易量
- 定义：int

数据流：

事务：
- 说明：进货、销售、退货
- 来源：工作人员
- 去向：订单记录

报表：
- 说明：月度统计信息
- 来源：订单记录
- 去向：工作人员

供货商信息变动：
- 说明：新的供应信息
- 来源：工作人员
- 去向：供货商信息

书目信息查询：
- 说明：书目价格、库存
- 来源：书目信息
- 去向：工作人员

供货商信息查询：
- 说明：供应信息
- 来源：供货商信息
- 去向：工作人员

3.功能需求分析

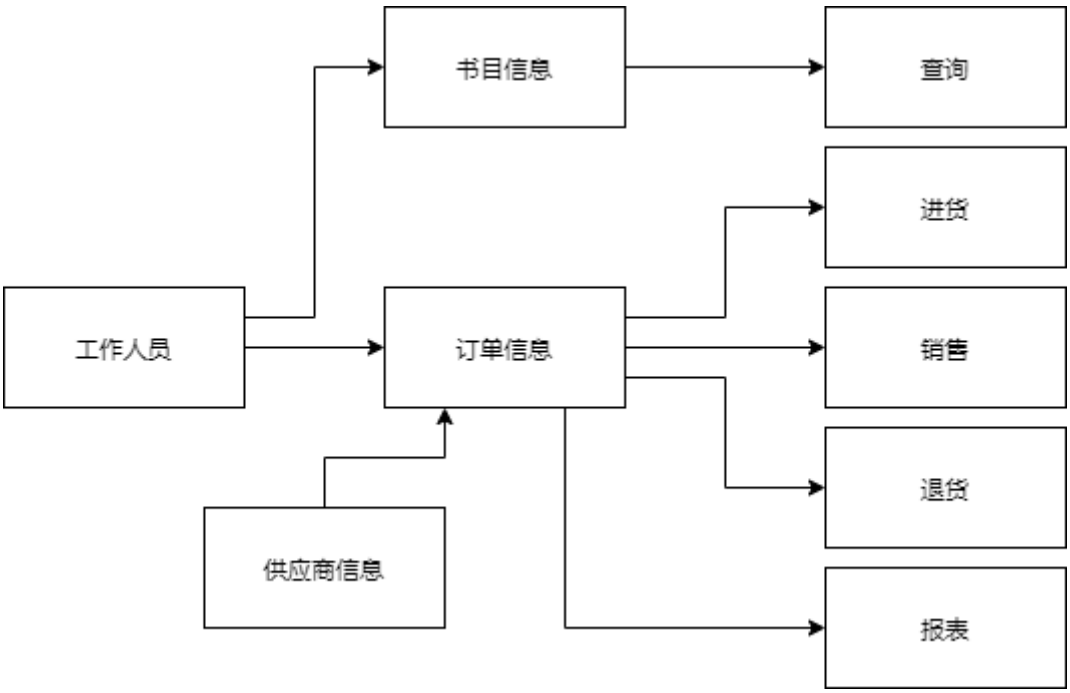
进货：根据某种书籍的库存量及销售情况确定进货数量，根据供应商报价选择供应商。输出一份进货单并自动修改库存量，把本次进货的信息添加到进货库中。

退货：顾客把已买的书籍退还给书店。输出一份退货单并自动修改库存量，把本次退货的信息添加到退货库中。

统计：根据销售情况输出统计的报表。一般内容为每月的销售总额、销售总量及排行榜。

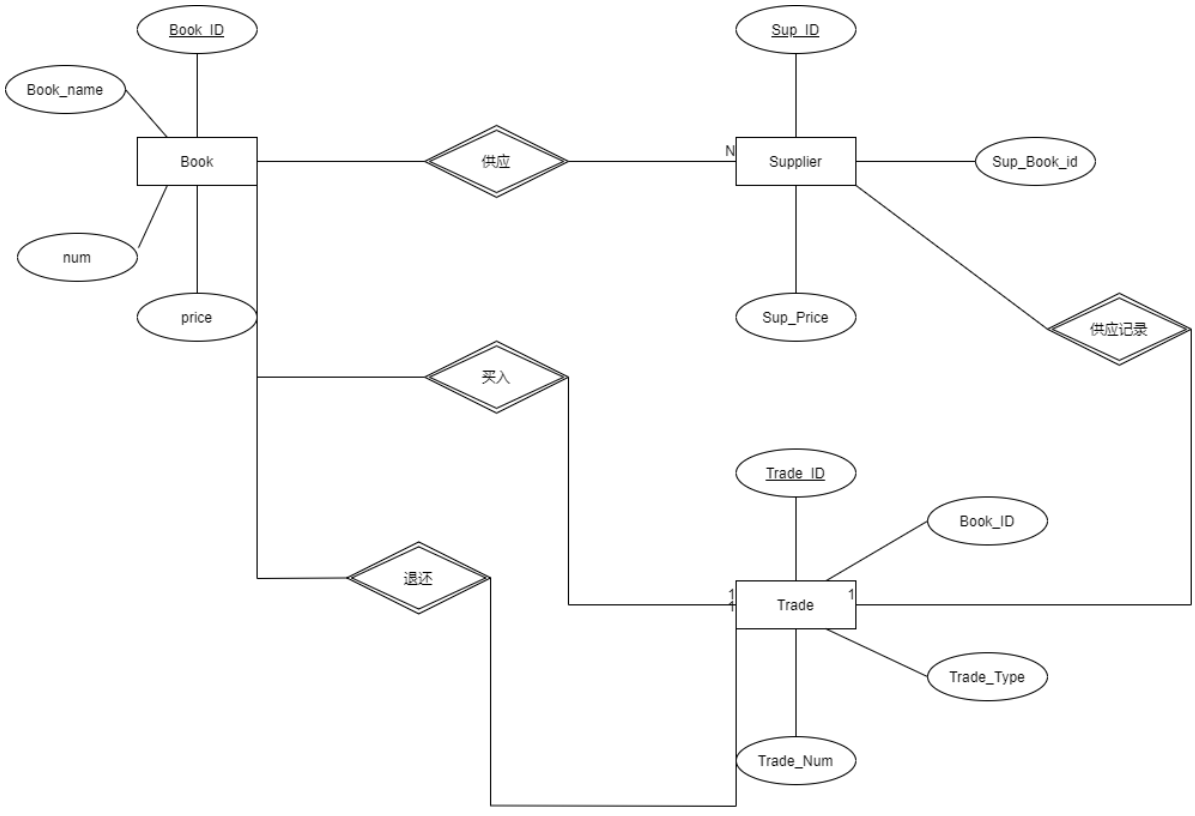
销售：输入顾客要买书籍的信息，自动显示此书的库存量，如果可以销售。打印销售单并修改库存，同时把此次销售的有关信息添加到日销售库中。

系统功能模块图



4.系统设计

4.1数据概念结构设计



系统ER图

4.2数据库关系模式设计

书籍（**书籍id**，书籍名称，库存量，价格）

供应商（**供应商id**，供应书籍id，供应价格）

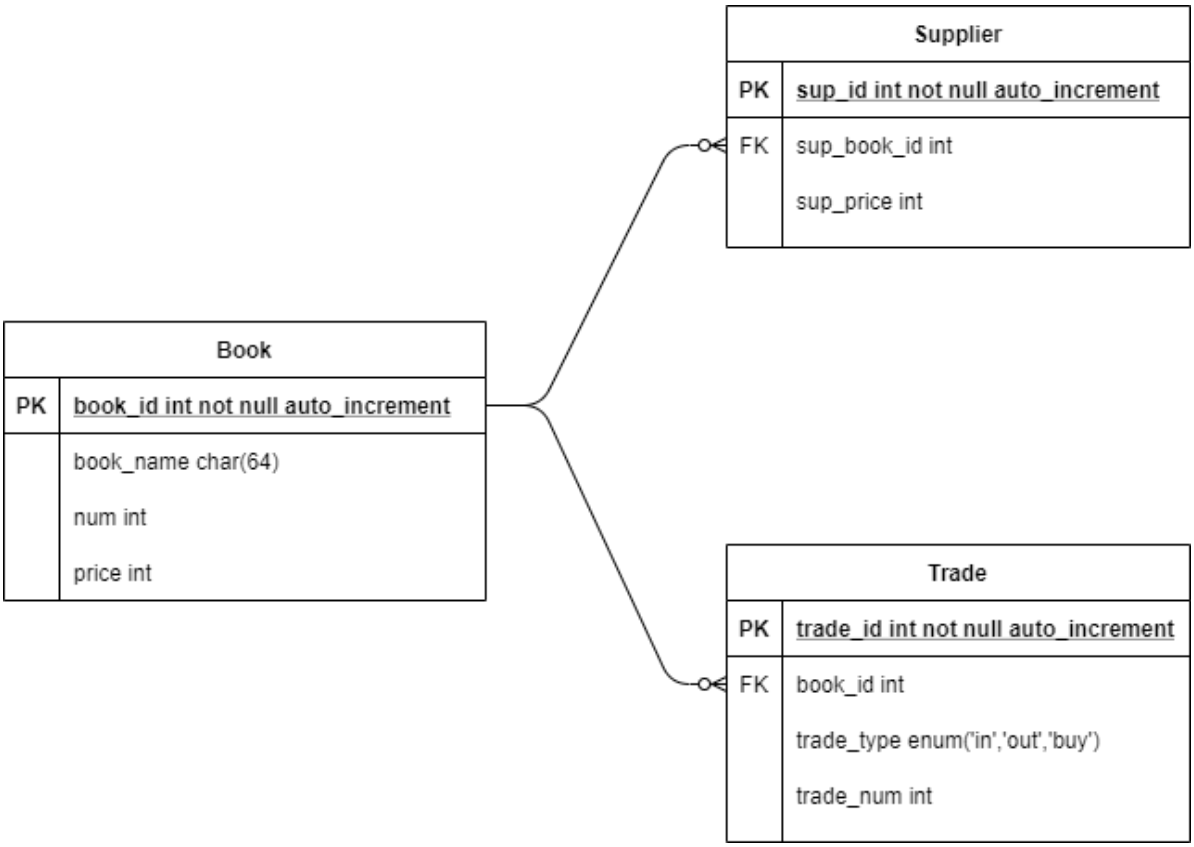
交易记录（**交易id**，交易书籍id，交易类型(进货，退货，出售)，交易数量）

出售（书籍id，数量）

退货（书籍id，数量）

进货（书籍id，供应商id，数量）

4.3数据库物理结构设计

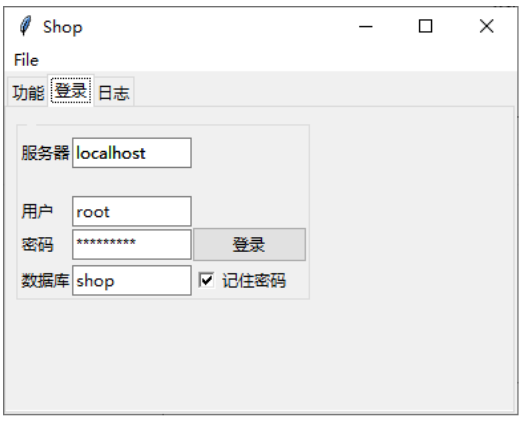


5.系统功能的实现

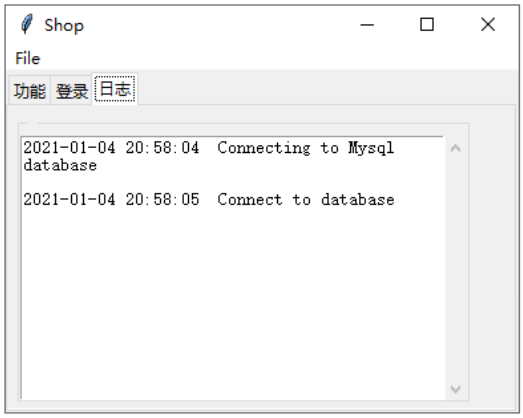
基础功能

连接数据库

显示登录界面，根据服务器，用户名，密码及需要连接的数据库名称连接到数据库。



登录界面

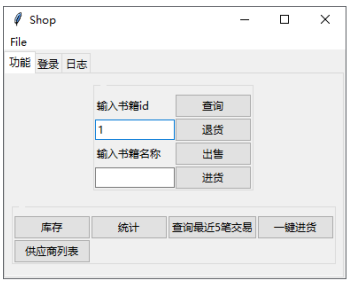


登录成功日志

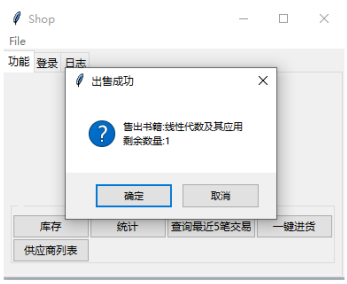
销售

根据输入的书籍ID出售一本该书籍，如果剩余书籍数量少于0，则显示出售失败。

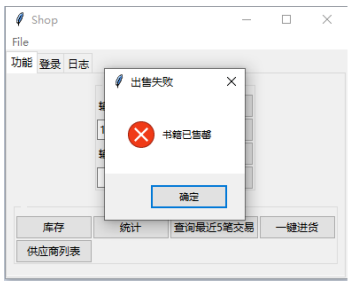
```
def __sale(self):
    _id = self.io_book_id.get()
    _name = self.io_book_name.get()
    if _id:
        sel_sql = 'select * from book where book_id={};'.format(_id)
    else:
        sel_sql = 'select * from book where book_name=\'{}\';'.format(_name)
    try:
        self.cursor.execute(sel_sql)
        _result = self.cursor.fetchone()
        if _result[2] == 0:
            raise ValueError
        _message = '售出书籍:{}\n剩余数量:{}'.format(_result[1], _result[2] - 1)
        exe_sql = 'insert into trade(book_id,trade_type,trade_num)
value({},\'out\',1);'.format(_result[0])
        self.cursor.execute(exe_sql)
        exe_sql = 'update book set num=num-1 where book_id=
{};'.format(_result[0])
        self.cursor.execute(exe_sql)
        ok = tkinter.messagebox.askokcancel(title='出售成功', message=_message)
    except TypeError as e:
        ok = tkinter.messagebox.showerror(title='出售失败', message='未查找到该书
籍')
    except AttributeError as e:
        ok = tkinter.messagebox.showerror(title='出售失败', message='未连接到数据
库')
    except ValueError as e:
        ok = tkinter.messagebox.showerror(title='出售失败', message='书籍已售罄')
    finally:
        self.io_book_id.set('')
        self.io_book_name.set('')
        self.conn.commit()
```



根据书籍ID出售书籍



出售成功

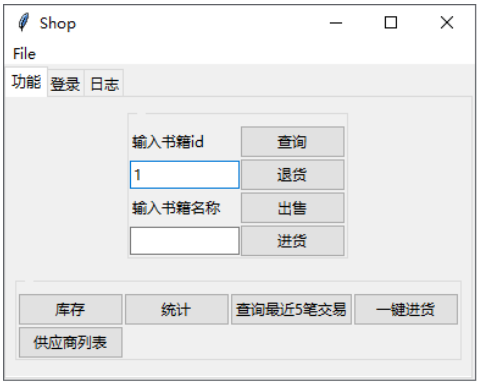


出售失败

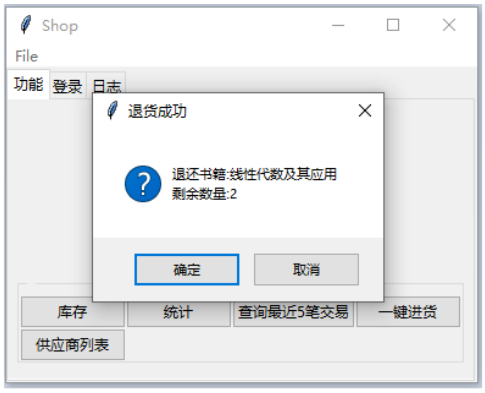
退货

根据输入的书籍ID退货。

```
def __ret(self):
    _id = self.io_book_id.get()
    _name = self.io_book_name.get()
    if _id:
        sel_sql = 'select * from book where book_id={};'.format(_id)
    else:
        sel_sql = 'select * from book where book_name=\'{}\';'.format(_name)
    try:
        self.cursor.execute(sel_sql)
        _result = self.cursor.fetchone()
        _message = '退还书籍:{}\n剩余数量:{}'.format(_result[1], _result[2] + 1)
        exe_sql = 'insert into trade(book_id,trade_type)
value({},\'in\');'.format(_result[0])
        self.cursor.execute(exe_sql)
        exe_sql = 'update book set num=num+1 where book_id=
{};'.format(_result[0])
        self.cursor.execute(exe_sql)
        ok = tkinter.messagebox.askokcancel(title='退货成功', message=_message)
    except TypeError as e:
        ok = tkinter.messagebox.showerror(title='退货失败', message='未查找到该书
籍')
    except AttributeError as e:
        ok = tkinter.messagebox.showerror(title='退货失败', message='未连接到数据
库')
    finally:
        self.io_book_id.set('')
        self.io_book_name.set('')
        self.conn.commit()
```



退货界面



退货成功

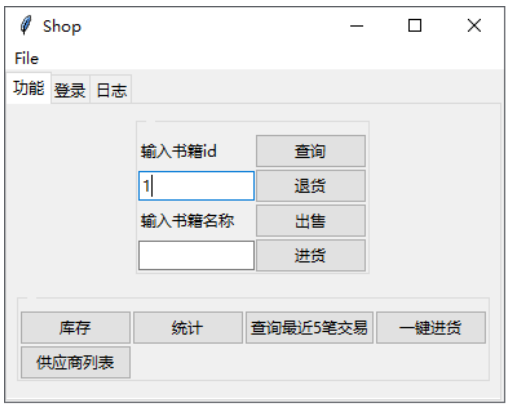
进货

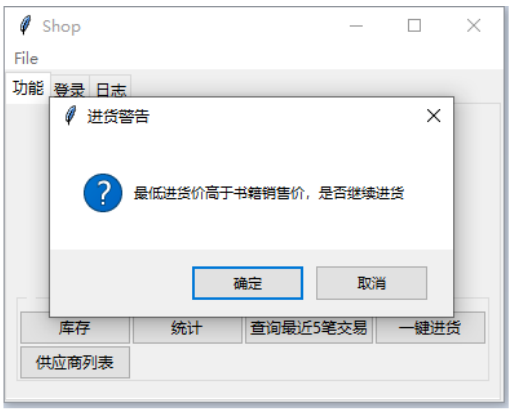
- 单次进货

  根据输入的书籍ID，选择价格最低的供应商进货。

  如果供应商最低价格高于书籍定价，即该书籍利润为负，弹出警告。

```
def __buy_one(self):
    try:
        _id = self.io_book_id.get()
        sel_sql = 'select * from book where book_id={};'.format(_id)
        self.cursor.execute(sel_sql)
        _result = self.cursor.fetchone()
        self.__buy_book(_id, _result[3], 1)
    except pymysql.err.ProgrammingError as e:
        tkinter.messagebox.showerror(title='进货失败', message='未查找到该书籍')
    except AttributeError as e:
        ok = tkinter.messagebox.showerror(title='进货失败', message='未连接到数
据库')
```





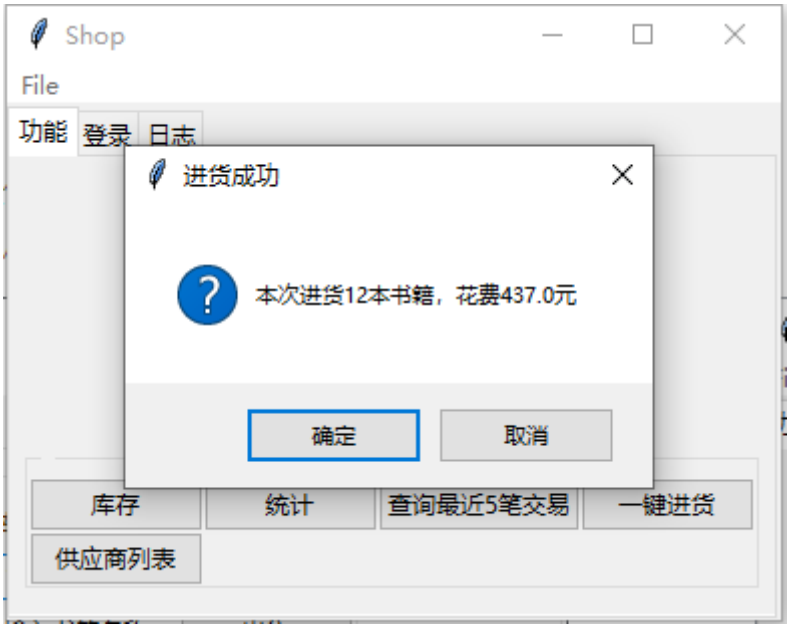
进货警告

- 批量进货

  统计历史销售量，如果目前书籍存量x小于历史销售量n，则进货$n-x$本。

```
def __buy(self):
    sel_sql = 'select * from trade'
    try:
        self.cursor.execute(sel_sql)
        _result = self.cursor.fetchall()
        total = dict()
        for _, bid, _type, __ in _result:
            if bid not in total:
                total.update({bid: 1})
            if _type == 'out':
                total[bid] += 1
            else:
                total[bid] -= 1
        t_num, t_price = 0, 0
        for bid in total:
            sel_sql = 'select * from book where book_id={};'.format(bid)
            self.cursor.execute(sel_sql)
```

```
            _result = self.cursor.fetchone()
            sub = total[bid] - _result[2] + 1
            if sub > 0:
                self.__buy_book(bid, _result[3], sub)
                t_num += sub
                t_price += sub * _result[3]
        ok = tkinter.messagebox.askokcancel(title='进货成功',
                                            message='本次进货{}本书籍，花费{}
元'.format(t_num, round(t_price / 1000, 3)))
    except AttributeError as e:
        ok = tkinter.messagebox.showerror(title='进货失败', message='未连接到数
据库')
    finally:
        self.conn.commit()
```



统计

根据历史销售量输出统计的报表，包括销售额，销售量排行榜。

```
def __statistic(self):
    sel_sql = 'select * from trade'
    try:
        self.cursor.execute(sel_sql)
        _result = self.cursor.fetchall()
        total = dict()
        for _, bid, _type, __ in _result:
            if bid not in total:
                total.update({bid: [0]})
            if _type == 'out':
                total[bid][0] += 1
            elif _type == 'in':
                total[bid][0] -= 1
        for bid in total:
            sel_sql = 'select * from book where book_id={};'.format(bid)
            self.cursor.execute(sel_sql)
            _result = self.cursor.fetchone()
            total[bid].append(total[bid][0] * _result[3])
            total[bid].append(_result[1])
```

```
        new_frame = tk.Toplevel()
        new_frame.title('报表')
        tree = ttk.Treeview(new_frame, show='headings')
        tree['columns'] = ('书籍id', '书籍名称', '销售总量', '销售总额')
        for c in tree['columns']:
            tree.column(c, width=150, anchor='center')
            tree.heading(c, text=c)
        i = 0
        for _, v in total.items():
            tree.insert('', i, values=(_, v[2], v[0], round(v[1] / 1000, 3)))
            i += 1
        for col in tree['columns']:
            tree.heading(col, text=col, command=lambda _col=col:
self.treeview_sort_column(tree, _col, False))
        tree.pack()
        new_frame.update()
        self.__set_win_center(new_frame)
    except AttributeError as e:
        ok = tkinter.messagebox.showerror(title='统计失败', message='未连接到数据
库')
```

按照销售总额排序

按照销售量排序

额外功能

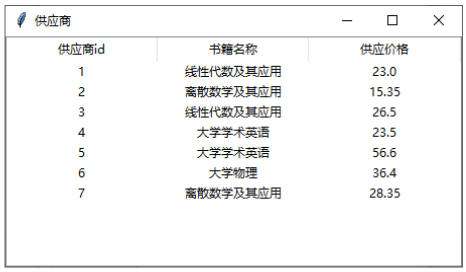
库存、供应商查询

查询目前库存已有书籍，显示书名、价格、存量；查询所有供应商。

```
def __get_supplier(self):
    sel_sql = 'select * from supplier'
    try:
        self.cursor.execute(sel_sql)
        _result = self.cursor.fetchall()
        new_frame = tk.Toplevel()
        new_frame.title('供应商')
        tree = ttk.Treeview(new_frame, show='headings')
        tree['columns'] = ('供应商id', '书籍名称', '供应价格')
        for c in tree['columns']:
            tree.column(c, width=150, anchor='center')
            tree.heading(c, text=c)
        i = 0
        cache = dict()
        for _id, _bid, price in _result:
            if _bid not in cache:
                sel_sql = 'select * from book where book_id={};'.format(_bid)
```

```
                self.cursor.execute(sel_sql)
                _result = self.cursor.fetchone()
                cache.update({_bid: _result[1]})
            name = cache[_bid]
            tree.insert('', i, values=(_id, name, round(price / 1000, 3)))
            i += 1
        tree.pack()
        new_frame.update()
        self.__set_win_center(new_frame)
    except AttributeError as e:
        ok = tkinter.messagebox.showerror(title='查询失败', message='未连接到数据
库')
```

库存



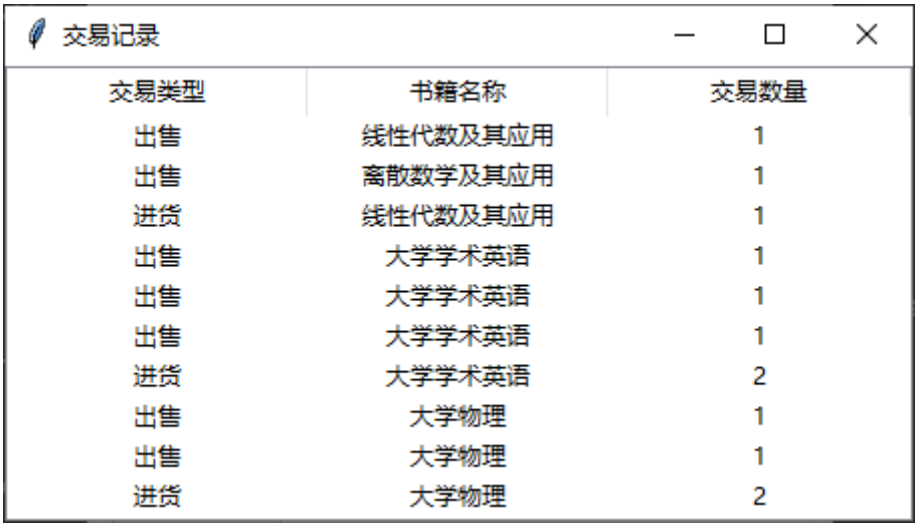
供应商列表

交易记录查询

查询最近的30笔交易记录

```
def __latest_k_trade(self, k):
    sel_sql = 'select * from trade'
    try:
        self.cursor.execute(sel_sql)
        _result = list(self.cursor.fetchall())
        _result = _result[-min(k, len(_result)):]
        _result = list(map(self.__trans, _result))
        new_frame = tk.Toplevel()
        new_frame.title('交易记录')
        tree = ttk.Treeview(new_frame, show='headings')
        tree['columns'] = ('交易类型', '书籍名称', '交易数量')
        for c in tree['columns']:
            tree.column(c, width=150, anchor='center')
            tree.heading(c, text=c)
        i = 0
        cache = dict()
        for _id, _type, num in _result:
            if _id not in cache:
                sel_sql = 'select * from book where book_id={};'.format(_id)
                self.cursor.execute(sel_sql)
                _result = self.cursor.fetchone()
                cache.update({_id: _result[1]})
            name = cache[_id]
            tree.insert('', i, values=(_type, name, num))
            i += 1
        tree.pack()
        new_frame.update()
        self.__set_win_center(new_frame)
```

```
    except AttributeError as e:
        ok = tkinter.messagebox.showerror(title='查询失败', message='未连接到数据
库')
```

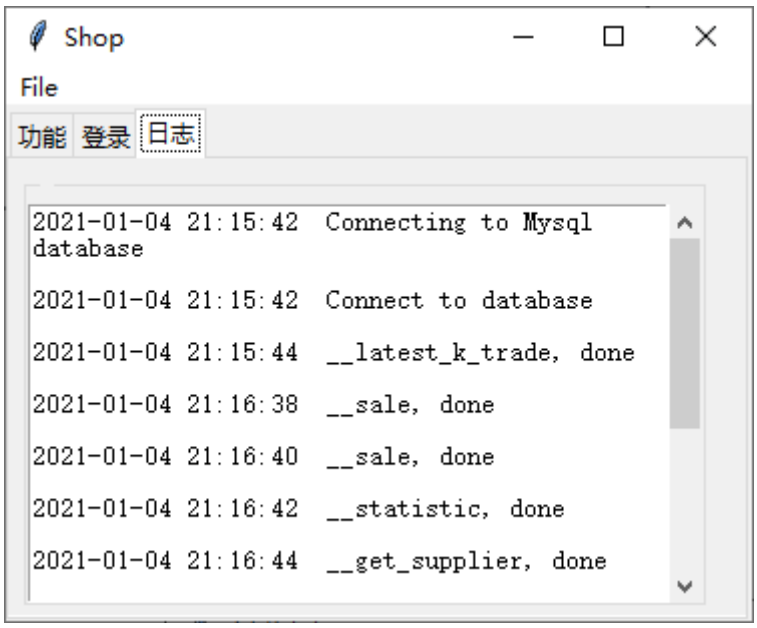


交易类型	书籍名称	交易数量
出售	线性代数及其应用	1
出售	离散数学及其应用	1
进货	线性代数及其应用	1
出售	大学学术英语	1
出售	大学学术英语	1
出售	大学学术英语	1
进货	大学学术英语	2
出售	大学物理	1
出售	大学物理	1
进货	大学物理	2

日志

每次操作写入日志，便于排查错误。

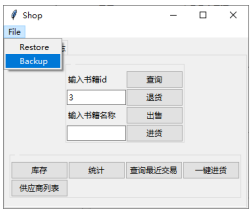
```
def write_log(func):
    def wrapper(self, *args, **kwargs):
        self.update_detail('{}, done'.format(func.__name__))
        return func(self, *args, **kwargs)

    return wrapper

def update_detail(self, s):
    self.detail.configure(state=tk.NORMAL)
    self.detail.insert(INSERT, str(time.strftime("%Y-%m-%d %H:%M:%S  ",
time.localtime())) + s + '\n\n')
    self.detail.see(END)
    self.detail.configure(state=tk.DISABLED)
```



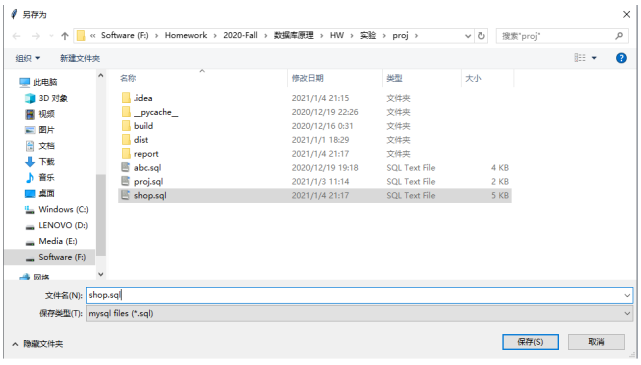
备份数据库

选定路径，将目前已连接的数据库导出备份。

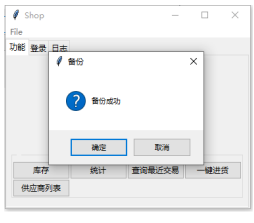
```
def __backup(self):
    try:
        id(self.cursor)
        backup_path = tkinter.filedialog.asksaveasfile(mode='w',
defaultextension=".sql",
                                                       filetypes=[('mysql
files', '.sql')])
        if not backup_path:
            return
        backup_path = backup_path.name
        _user = self.login_user_name.get()
        _pwd = self.login_pwd_name.get()
        _db = self.login_db_name.get()
        os.popen('mysqldump -u{} -p{} {} > "{}"'.format(_user, _pwd, _db,
backup_path))
        tkinter.messagebox.askokcancel(title='备份', message='备份成功')
        self.update_detail('backup sql')
    except AttributeError as e:
        self.update_detail(str(e))
        tkinter.messagebox.showerror(title='备份失败', message='未连接到数据库')
```



备份选项



选择路径

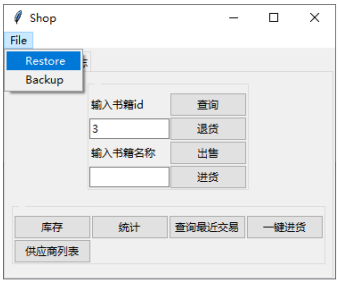


备份成功

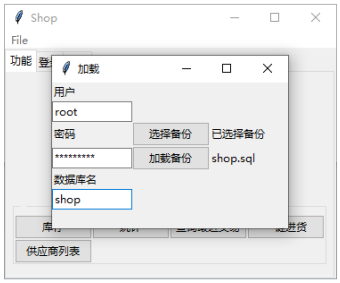
加载已备份数据库

选择数据库备份文件，将数据库导入到指定服务器，需要用户名及密码。

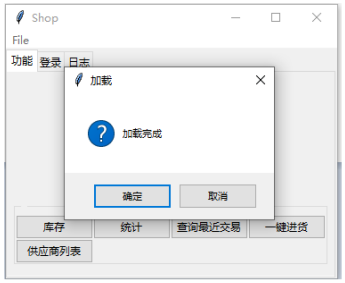
```
def __load_restore(self):
    _user = self.sql_user_name.get()
    _pwd = self.sql_pwd_name.get()
    _db = self.sql_db_name.get()
    os.popen('mysql -u{} -p{} --execute="drop database if exists
{};"'.format(_user, _pwd, _db))
    os.popen('mysql -u{} -p{} --execute="create database {};"'.format(_user,
_pwd, _db))
    os.popen('mysql -u{} -p{} {} < "{}"'.format(_user, _pwd, _db,
self.restorePath))
    self.update_detail('restore {}'.format(self.restorePath))
    tkinter.messagebox.askokcancel(title='加载', message='加载完成')
```



加载选项



加载界面，输入用户名密码。



加载成功

6.总结

1. 数据库管理系统（DataBase-Management System，DBMS）由一个互相关联的数据的集合和一组用以访问这些数据的程序组成。
2. 关系模型：关系模型用表的集合来表示数据和数据间的联系。每个表有多个列，每列有唯一的列名。关系模型是基于记录的模型的一种。
3. 实体-联系模型：实体-联系(ER)数据模型基于对现实世界的这样一种认识：现实世界由一组称作实体的基本对象以及这些对象间的联系构成。实体是现实世界中可区别于其他对象的一件“事情”或一个“物体”。
4. 域约束：每个属性都必须对应于一个所有可能的取值构成的域，比如这次设计中的交易类型的域就只有三种：进货、退货、出售。
5. 参照完整性：一个关系给定属性集上的取值也在另一关系的某一属性集的取值中出现。