图书管理系统

---18307130130 李睿琛

1. 数据库关系表设计：

关系表 **book** 包含五个属性值：title、ISBN、author、condp、reason
title：书名。
ISBN：书的编号。
author：作者。
condp：该书是否被移除。0 表示被移除，1 表示在架上。
reason：移除原因。未被移除为空。
主键：ISBN。

关系表 **lend** 包含五个属性值：ret、getdate、delay_cnt、book_id、student_id
ret：书籍状态。1 代表书籍已被归还；0 代表书籍未被归还
Getdate：书籍被借阅的日期
delay_cnt：书籍归还期限延期次数
book_id：外键，即 book 的主键
student_id：外键，即 student 的主键

关系表 **student** 包含两个属性值：
Ability：借阅书籍的权限。1 代表有权限，0 代表无权限
account：即学生账号。
主键：account

2. 各函数实现以及测试结果：

2.1 外部结构：

声明了 book 结构，方便书籍的批量输入。

```
type book struct {
    title  string
    ISBN   string
    author string
}
```

2.2TestMain：

功能：TestMain 是在所有 Test 之前进行的，一般用于初始化。

实现：用于连接 mysql 并创建数据库，同时向数据库中导入书籍。

测试：

```
func TestMain(m *testing.M) {
    books := []book{
        book{"B1","1","A1",},
        book{"B2","2","A2",},
        book{"B3","3","A3",},
        book{"B4","4","A4",},
        book{"B5","5","A5",},
        book{"B6","6","A6",},
        book{"B7","7","A7",},
        book{"B8","8","A8",},
        book{"B9","9","A9",},
        book{"B10","10","A10",},
    }
    lib.CreateDB()
    lib.ConnectDB()
    lib.CreateTables()
    lib.AddBooks("root", books)
    lib.AddBook("root", "B10", "A10", "10")
    m.Run()
}
```

测试结果：

```
flock@flock-Inspiron-5370:~/Desktop/2020-FD-library/boilerplate$ go test -v
create DB:ass3 success
connect DB:ass3 success
Createtables success
Add books by root
add books auto:
Add the book 1 B1 A1
Add the book 2 B2 A2
Add the book 3 B3 A3
Add the book 4 B4 A4
Add the book 5 B5 A5
Add the book 6 B6 A6
Add the book 7 B7 A7
Add the book 8 B8 A8
Add the book 9 B9 A9
Add the book 10 B10 A10
add books success
the book:10 already exits
,addbook invalid=== RUN   TestRemovebook
```

结果说明：

成功连接并创建数据库，成功添加十本书籍

最后的 invalid 操作是因为该书本已经在 library 中。

2.3 各功能实现：

2.3.1 ConnectDB()

实现：
调用 sqlx.open()连接 mysql

2.3.2CreateDB()：

实现：
调用 db.Exec()函数创建数据库 ass3

2.3.3CreateTables()

实现：
使用“CREATE TABLE”语句，调用 db.Exec()函数创建 book、student、lend 三个关系表。

```
CREATE TABLE book
(title VARCHAR(32) NOT NULL,
ISBN VARCHAR(32) NOT NULL,
author VARCHAR(32) NOT NULL,
condp bool NOT NULL,
reason VARCHAR(32) NOT NULL,
PRIMARY KEY(ISBN)
);
CREATE TABLE lend
(ret bool NOT NULL,
getdate datetime NOT NULL,
delay_cnt INT NOT NULL,
book_id VARCHAR(32) NOT NULL,
student_id VARCHAR(32) NOT NULL,
FOREIGN KEY (book_id) REFERENCES book(ISBN),
FOREIGN KEY (student_id) REFERENCES student(account)
);
CREATE TABLE student
(account VARCHAR(32) NOT NULL,
ability bool NOT NULL,
PRIMARY KEY(account)
);
```

2.3.4 AddBooks(admin string, books [ ]book)

功能：
这里实现了自动添加书籍和手动添加书籍两种方式。

实现：
当传入参数 admin 与 User 相等时，即为 administrator 时，才能进行有效操作，否则没有添加书籍权限。
而当传入参数 books 为空时，手动添加书籍；当传入参数 book 为结构体切片时，按结构体自动添加书籍。添加书籍通过调用 Addbook（）一本本添加。

2.3.5 AddBook(admin, title, author, ISBN string)

功能：
添加一本书籍

实现：
当传入参数 admin 与 User 相等时，即为 administrator 时，才能进行有效操作，否则没有添加书籍权限。
对于传入的 title、author、ISBN 参数，使用“INSERT”语句向 book 中插入元组，元组的 condp、reason 属性初始值分别为 1，“”。

```
`INSERT INTO book(title,ISBN,author,condp,reason)
                    VALUES("%s","%s","%s",1,"");`
```

2.3.6 RemoveBook(ISBN, reason string)

功能：
从 library 删除某本书籍。

实现：
首先通过 ISBN 和“SELECT”语句判断该书是否在 library 中，

```
err := lib.db.QueryRow("select ISBN from book where ISBN = ? AND condp=1",
ISBN).Scan(&iSBN)
```

如果在，“UPDATE”语句更新 book 中对应元组 condp 为 0，代表将书删除，reason 记录删除原因。
如果不在，即 err == sql.ErrNoRows，fmt.Printf 输出信息该书不在 library，删除操作无效。

测试：

```
func TestRemovebook(t *testing.T) {
    err := lib.RemoveBook("50", "the book is lost")
    if err != nil {
        t.Errorf("can't remove book")
    }
    err = lib.RemoveBook("2", "the book is lost")
    if err != nil {
        t.Errorf("can't remove book")
    }
    t.Logf("removebook success")
}
```

测试结果：

```
 ,addbook invalid=== RUN   TestRemovebook
 the book:50 doesn't exit,remove invalid
 remove the book 2 for the reason: the book is lost
     TestRemovebook: library_test.go:21: removebook success
 --- PASS: TestRemovebook (0.00s)
```

结果说明：

编号 50 的书籍不在 book 关系表中，remove 操作无效；

编号 2 的书籍成功被删除。

2.3.7 AddAccount(s string)

实现：

使用“INSERT”语句向 student 中插入对应元组。Ability 初始值为 1。

测试：

```
run test | debug test
func TestAddAccount(t *testing.T) {
    lib.AddAccount("s1")
    err := lib.AddAccount("s2")
    if err != nil {
        t.Errorf("can't Add Account")
    }
    t.Logf("AddAccount success")
}
```

测试结果

```
 === RUN   TestAddAccount
  Add account s1
  Add account s2
     TestAddAccount: library_test.go:33: AddAccount success
 --- PASS: TestAddAccount (0.00s)
```

添加成功！

2.3.8 QueryByTitle(s string)、QueryByAuthor(s string)、QueryByISBN(s string)

实现：

使用“SELECT”语句选择 book 中对应元组。

测试：

```
run test | debug test
func TestQueryByTitle(t *testing.T) {
    err := lib.QueryByTitle("B3")
    if err != nil {
        t.Errorf("can't Query By Title")
    }
    t.Logf("QueryByTitle success")
}

run test | debug test
func TestQueryByAuthor(t *testing.T) {
    err := lib.QueryByAuthor("A1")
    if err != nil {
        t.Errorf("can't Query By Author")
    }
    t.Logf("QueryByAuthor success")
}

run test | debug test
func TestQueryByISBN(t *testing.T) {
    err := lib.QueryByISBN("3")
    if err != nil {
        t.Errorf("can't Query By ISBN:2")
    }
    t.Logf("QueryByISBN success")
}
```

测试结果：

```
=== RUN   TestQueryByTitle
 Query By title:B3
title ISBN author condp reason
B3 3 A3 true
    TestQueryByTitle: library_test.go:33: QueryByTitle success
--- PASS: TestQueryByTitle (0.00s)
=== RUN   TestQueryByAuthor
 Query By author:A1
title ISBN author condp reason
B1 1 A1 true
    TestQueryByAuthor: library_test.go:41: QueryByAuthor success
--- PASS: TestQueryByAuthor (0.00s)
=== RUN   TestQueryByISBN
 Query By ISBN:3
title ISBN author condp reason
B3 3 A3 true
    TestQueryByISBN: library_test.go:49: QueryByISBN success
--- PASS: TestQueryByISBN (0.00s)
```

2.3.9 BorrowBook(account, ISBN string)

功能：

借书并判断该学生是否用权限继续借书。归还期限 30 天，延长一次延迟 7 天。

实现：

首先“SELECT”语句选择对应 account 的学生，获得其借书权限，

```
err := lib.db.QueryRow(s).Scan(&able)
```

如果该学生不存在 student 关系表中，即 err == sql.ErrNoRows 输出对应信息；
如果存在，再判断书是否存在 book 关系表中。

```
err := lib.db.QueryRow("SELECT condp FROM book WHERE ISBN=? AND condp=1",
ISBN).Scan(&exist)
```

如果不存在，即 err == sql.ErrNoRows 输出对应信息；
如果存在，判断学生是否有借书权限，如果没有，输出对应信息；如果有，再判断书是否已经被借出。

```
err := lib.db.QueryRow("SELECT book_id,student_id FROM lend WHERE book_id=? AND ret=0",
ISBN).Scan(&id, &sid)
```

如果被借出，输出对应信息；
如果没被借出，使用“INSERT”语句向 lend 关系表中插入相应元组。
lend(ret,getdate,delay_cnt,book_id,student_id)。
Ret、delay_cnt 初始值分别为 0，0
Getdate 的获得使用 time 包。

```
now := time.Now()
getdate := now.Format("2006-01-02 15:04:05")
```

同时每次借书成功后，通过“SELECT COUNT(*)”语句统计该学生借书总数，如果等于 3，则通过“UPDATE”更新 student 表中对应 ability 为 0 取消继续借书的权限

```
UPDATE student  SET ability=0 WHERE account='%s' `, account
```

测试：

```
run test | debug test
func TestBorrowBook(t *testing.T) {
    lib.BorrowBook("s3", "4")
    lib.BorrowBook("s1", "2")
    lib.BorrowBook("s1", "4")
    lib.BorrowBook("s1", "5")
    lib.BorrowBook("s1", "6")
    lib.BorrowBook("s2", "4")
    err := lib.BorrowBook("s1", "7")
    if err != nil {
        t.Errorf("can't Borrow Book")
    }
    t.Logf(" BorrowBook success")
}
```

测试结果：

```
=== RUN   TestBorrowBook
the student:s3 doesn't exit,borrow invalid
the book:2 doesn't in the library,borrow invalid
the student:s1 borrow the book: 4, please return in 30 days
the student:s1 borrow the book: 5, please return in 30 days
the student:s1 borrow the book: 6, please return in 30 days
the book:4 has been borrowed by student:s1, borrow invalid
the student:s1 shouldn't borrow more than three books, borrow invalid
    TestBorrowBook: library_test.go:63:  BorrowBook success
--- PASS: TestBorrowBook (0.03s)
```

结果说明：

学生 s3 不存在，书籍 2 已结被移除，输出对应信息。

学生 s1 成功借书 4、5、6。学生 s2 想借书 4，但改书已结被借出，输出对应信息。

学生 s1 无法再借书 7，因为借书不能超过 3 本。

2.3.10 QueryHistory(account string)

实现：

通过“SELECT *”语句：`SELECT * FROM lend WHERE student_id="%s" `, account

然后 Rows.Next（）循环打印出对应元组。

测试：

```
run test | debug test
func TestQueryHistory(t *testing.T) {
    err := lib.QueryHistory("s1")
    if err != nil {
        t.Errorf("can't Query History")
    }
    t.Logf("QueryHistory success")
}
```

测试结果：

```
=== RUN   TestQueryHistory
the borrow history of student s1:
ret getdate delay_cnt book_id student_id
false 2020-04-28 13:54:53 0 4 s1
false 2020-04-28 13:54:53 0 5 s1
false 2020-04-28 13:54:53 0 6 s1
    TestQueryHistory: library_test.go:97: QueryHistory success
--- PASS: TestQueryHistory (0.00s)
```

结果说明：

输出 s1 的借书记录。

2.3.11 QueryBookCon(account string)

实现：

通过“SELECT *”语句

```
` SELECT book_id FROM lend WHERE student_id='%s' AND ret=0 `, account
```

然后 Rows.Next（）打印出对应元组的 book_id。

测试：

```
run test | debug test
func TestQueryBookCon(t *testing.T) {
    err := lib.QueryBookCon("s1")
    if err != nil {
        t.Errorf("can't Query BookCon")
    }
    t.Logf("QueryBookCon success")
}
```

测试结果：

```
=== RUN   TestQueryBookCon
ISBN
the books student s1 has borrowed and not returned yet:
4
5
6
    TestQueryBookCon: library_test.go:108: QueryBookCon success
--- PASS: TestQueryBookCon (0.00s)
```

2.3.12 CheckDeadline(ISBN string) error

实现：

通过“SELECT *”语句获取对应书籍的 getdate 和 delay_cnt

Deadline 计算逻辑如下：使用 time.Parse（）等函数完成类型转换

```
getdate, _ := time.Parse("2006-01-02 15:04:05", data)
day, _ := time.ParseDuration("24h")
deadline := getdate.Add(day * 30).Format("2006-01-02 15:04:05")
if cnt == 0 {
} else if cnt == 1 {
    deadline = getdate.Add(day * 37).Format("2006-01-02 15:04:05")
} else if cnt == 2 {
    deadline = getdate.Add(day * 44).Format("2006-01-02 15:04:05")
} else {
    deadline = getdate.Add(day * 51).Format("2006-01-02 15:04:05")
}
```

测试：

```
run test | debug test
func TestCheckDeadline(t *testing.T) {
    err := lib.CheckDeadline("4")
    if err != nil {
        t.Errorf("can't Check Deadline")
    }
    t.Logf("CheckDeadline success")
}
```

测试结果：

```
=== RUN   TestCheckDeadline
the deadline of the borrowed book 4 is 2020-05-28 13:54:53
    TestCheckDeadline: library_test.go:119: CheckDeadline success
--- PASS: TestCheckDeadline (0.00s)
```

2.3.13 ExtendDeadline(ISBN string)

实现：

通过“SELECT *”语句获取对应书籍的 delay_cnt

如果 delay_cnt 为 3，则不能再延迟期限，输出对应信息。

如果小于 3，通过“UPDATE”语句更新 lend 表中对应元组 delay_cnt+=1

测试：

```
    lib.ExtendDeadline("4")
    lib.ExtendDeadline("4")
    lib.ExtendDeadline("4")
    err := lib.ExtendDeadline("4")
    if err != nil {
        t.Errorf("can't Extend Deadline")
    }
    t.Logf("ExtendDeadline success")
}
```

测试结果：

```
=== RUN   TestExtendDeadline
extend 7 days to return the book 4
extend success
extend 7 days to return the book 4
extend success
extend 7 days to return the book 4
extend success
extend 7 days to return the book 4
the book 4 couldn't been delayed 3 times
    TestExtendDeadline: library_test.go:133: ExtendDeadline success
--- PASS: TestExtendDeadline (0.01s)
```

结果说明：

Book 4 已经延期三次，不能再延期。

2.3.14 CheckDue(account string)

实现：

通过“SELECT”语句从 lend 关系表中获取对应学生借的书籍的 book_id、getdate、delay_cnt。

用 rows.Next()遍历，对于每本书籍，根据 getdate、delay_cnt 获取对应 deadline，再用 Before 方法与现在时间比较 Dead.Before(nowdate)，筛选出最终结果。

```
deadline := getdate.Add(day * 30).Format("2006-01-02 15:04:05")
if delay_cnt == 0 {
} else if delay_cnt == 1 {
    deadline = getdate.Add(day * 37).Format("2006-01-02 15:04:05")
} else if delay_cnt == 2 {
    deadline = getdate.Add(day * 44).Format("2006-01-02 15:04:05")
} else {
    deadline = getdate.Add(day * 51).Format("2006-01-02 15:04:05")
}
Dead, _ := time.Parse("2006-01-02 15:04:05", deadline)
if Dead.Before(nowdate) {
    fmt.Println(ISBN)
}
```

测试：

```
run test | debug test
func TestCheckDue(t *testing.T) {
    err := lib.CheckDue("s1")
    if err != nil {
        t.Errorf("can't Check Due")
    }
    t.Logf("CheckDue success")
}
```

测试结果：

```
=== RUN   TestCheckDue
the overdue books the student s1 has:
    TestCheckDue: library_test.go:144: CheckDue success
--- PASS: TestCheckDue (0.00s)
```

结果说明：

学生 s1 没有待还书籍，所以没有输出。

2.3.15 ReturnBook(ISBN, account string)

实现：

通过“SELECT”语句筛选 lend 关系表中对应元组，如果存在，则用“UPDATE”语句更新 ret 为 1，不存在则输出对应信息。

测试：

```
func TestReturnBook(t *testing.T) {
    err := lib.ReturnBook("4", "s1")
    err = lib.ReturnBook("8", "s1")
    if err != nil {
        t.Errorf("can't Return Book")
    }
    t.Logf("ReturnBook success")
}
```

测试结果：

```
=== RUN   TestReturnBook
the student s1 return the book 4
the student s1 doesn't borrow the book 8,return invalid
    TestReturnBook: library_test.go:115: ReturnBook success
--- PASS: TestReturnBook (0.00s)
PASS
ok      github.com/ichn-hu/IDBS-Spring20-Fudan/assignments/ass3/boilerplate     0.165s
```

3. 图书管理系统特点

1. 数据库存储数据尽可能少，减少了硬件资源的占用。如 deadline 可由 getdate 和 delay_cnt 推出，所以不进行额外进行存储。
2. 考虑查询等操作无效，即不符合实际情况，并提供对应信息帮助改错。虽然这只在几个函数如：BorrowBook、ReturnBook 函数中做了考虑，但其方法完全可以在其他函数中复制。
3. 实现所有要求功能；对于添加书籍，提供了手动添加和自动批量添加两种方法。

4. 方法整理:

4.1 Sql 包的使用:

Sql.open、 db.Exec、 db.Query、 db.QueryRow、 rows.Scan、 sql.ErrNoRows、 rows.Next。

4.2 Time 包的使用:

time.Parse 、 time.ParseDuration 、 getdate.Add 、 time.Now 、 now.Format 、 Dead.Before(nowdate)

4.3 Fmt 包的使用:

Fmt.Printf、fmt.Println、fmt.Sprintf

5. 远程仓库:

https://github.com/Flock-cloud1218/2020-FD-library