Assignment 3: Library 项目报告

于凡奇 18307130182

项目链接： https://github.com/TwistsOfFate/FDU-lib

Assignment 3: Library 项目报告

- 1.整体设计
  - 理念
  - 设计要点
- 2.项目构成
- 3.数据组织
  - 数据库中的关系表
    - admins
    - users
    - books
    - records
  - fdulib中的结构体
    - Library
    - Book
    - UserHistory
- 4.功能介绍
  - 创建管理员/用户账户
  - 添加图书
  - 查询图书
  - 删除图书
  - 借书
  - 查询借书记录
  - 查询待还书目
  - 延期
  - 查询逾期书目
  - 还书
  - 更新用户状态
- 5.功能验证方法
- 6.SampleLib
  - 特点
- 7.致谢

1.整体设计

理念

鉴于此次Assignment具有一定的偏应用的倾向，目标是设计实现一个具有一定规模、解决现实需求的图书馆管理系统，我希望能够不局限于仅仅完整正确地实现功能，而是运用所学尽量做到**优雅**、**标准**地实现，同时也应使得系统易于维护、调整和升级。受到Unix设计思想的伟大影响，同时使用着理念较为先进的Go语言，我期待在本Library实现中能体现现代软件系统的一些优秀的特征：

- 一定程度上的"策略与机制分离"
- 接口在一定程度上的标准化
- 简洁、组合原则
- 完整的error handling

为此，在本Library中作了如下努力：

- 将直接与数据库进行交互的"机制"作为一个独立的模块`fdulib`，与用户使用时的"策略"相分离。`fdulib`把最小的必要的SQL访问操作封装成一系列Go函数接口，使得在未来通过不同的策略灵活地调用这些函数成为可能。模块`SampleLib`作为一个示例"策略"，体现了使用`fdulib`的一种可能的方式。
    - 当需求发生一定范围内的变化时（如添加一些组合功能），可以仅通过调整上层的"策略"模块来满足。另一方面，当数据库中数据组织方式发生一定范围内的变化时，也可以通过仅修改`fdulib`来适应。
- `fdulib`的众多查询类函数中，返回值均为两种类型：[]Book和[]UserHistory。因此，调用`fdulib`的上层模块只需提供对Book和UserHistory两种结构体的支持即可，模块对接变得更加容易。
- `fdulib`中删除图书等操作仅支持通过ID这一主键，这简化了函数的实现。但由于查询图书支持通过各字段检索，实际使用中只需在上层模块中将查询与删除操作进行组合便可实现通过各字段删除的功能。
- `fdulib`中提供的每一个函数都带有一个error返回值，这个函数内调用其它函数产生的任何错误都由此返回值传递出去；`SampleLib`中对大部分函数调用（特别是读入函数）都进行了error handling。
    - Library系统一方面需要与数据库进行通信，另一方面需要处理用户的（各种）输入，不再是那种简单的在只用和stdio打交道的本地运行的小程序，不受本身控制的因素极多，可能出现意外错误的环节很多。
    - `fdulib`在使用中须由别的上层模块调用，若无完整的error传递机制，底层出现问题时在用户层面完全无法确定问题所在。
    - 可借助数据库自身特性（如不允许插入两个主键相同的元组）检查部分操作合法性并直接显示由数据库提供的错误信息，简化了代码中判断操作是否合法的逻辑。

设计要点

- `fdulib`仅提供一系列功能，而不区分用户权限；具体的用户权限分配由上层模块灵活设置。
- Library系统应具有"记录"的功能，故在"删除"图书乃至用户账户时，不实际删除对应元组，而是将其标记为已删除，以便在未来能够回溯历史。
- 在进行需要记录时间的操作时，`fdulib`不直接使用当前时间，而是将时间作为函数的传入参数。因此在无法连接数据库时，一些操作可以由上层模块离线进行，同时正确地记录时间。
- 根据Analyzing Prepared Statement Performance，在提前准备SQL语句可能带来的好处不明显时，我们或许应该避免提前准备语句。由于database/sql的语句准备与连接管理的方式，一条语句可能被准备多次而未执行一次（准备好了但等太久连接关掉了，下次开了新连接又要重新准备一次）。为了避免这不必要的损耗，`fdulib`中的SQL语句均通过fmt.Sprintf等方法在Go中生成完整的填好参数的字符串再移送sqlx.DB执行，这样避免了准备语句。

2.项目构成

> - module fdulib
>   - main.go
>   - types.go
>   - fdulib_test.go
> - module SampleLib
>   - main.go

主要工作分为三部分：

1. `fdulib`**本体功能**的实现：包括fdulib/main.go和fdulib/types.go，后者定义了`fdulib`中使用的（同时也是从外部调用时需要用到的）结构体。
2. `fdulib`功能的**测试**：fdulib_test.go，通过一系列精心设计的测试验证了每个函数的功能。
3. `SampleLib`：一个调用`fdulib`的上层模块，通过对`fdulib`中函数的组合与包装相对完整地实现了各项功能，通过命令行与用户交互。

3.数据组织

这里主要介绍在数据库中的关系表的设计以及`fdulib`中的结构体。

数据库中的关系表

admins

```
CREATE TABLE IF NOT EXISTS admins(
    user VARCHAR(255) PRIMARY KEY,
    password VARCHAR(255)
)
```

- 用于储存管理员信息。

users

```
CREATE TABLE IF NOT EXISTS users(
   user VARCHAR(255) PRIMARY KEY,
   password VARCHAR(255),
   user_stat INT DEFAULT 0,
   overdue INT DEFAULT 0
)
```

- 用于储存普通用户信息。`user_stat`表示用户状态：0为正常，1为禁用（有3本以上的书逾期未还）。`overdue`记录用户当前逾期书本数目。

books

```
CREATE TABLE IF NOT EXISTS books(
    book_id VARCHAR(255) PRIMARY KEY,
    book_stat INT DEFAULT 0,
    title VARCHAR(255) NOT NULL,
    edition INT DEFAULT 1,
    author VARCHAR(255),
    ISBN VARCHAR(20),
    added_date DATE NOT NULL,
    added_by_admin VARCHAR(255) NOT NULL,
    expire_days INT NOT NULL,
    removed_date DATE,
    removed_by_admin VARCHAR(255),
    removed_msg VARCHAR(1023),
    FOREIGN KEY (added_by_admin) REFERENCES admins(user),
    FOREIGN KEY (removed_by_admin) REFERENCES admins(user)
)
```

- 包括了一本书的一系列基本信息，以及添加/删除时间、添加/删除者等管理信息。
- `book_stat`为0表示可用，1表示已借出，2表示已删除。
- 考虑到有一些书（如古籍）可能遗失了相关信息，这里大部分属性都是可NULL的。为了让`book_id`的取值更加灵活，如类似美国国会图书分类号的编码方式，将`book_id`类型设为字符串。

> 计算机专业学生耳熟能详的经典教材*Computer Systems: A Programmer's Perspective*的国会分类号为`QA76.5.B795 2016`，一串带有点和空格的乱七八糟字符。

- 为了让查询时匹配字段的方法更加方便、统一（利用MySQL丰富的模式匹配功能），大部分字段类型都设为字符串。这里用`expire_days`来表示每本书的归还期限，不同类型的书可以有长度不等的时限。

records

```
CREATE TABLE IF NOT EXISTS records(
    record_id VARCHAR(255) PRIMARY KEY,
    user VARCHAR(255) NOT NULL,
    book_id VARCHAR(255) NOT NULL,
    lent_date DATE NOT NULL,
    expire_date DATE NOT NULL,
    extended INT DEFAULT 0,
    returned_date DATE,
    FOREIGN KEY (user) REFERENCES users(user),
    FOREIGN KEY (book_id) REFERENCES books(book_id)
)
```

- 用于储存借书记录。由于一个用户可能多次借同一本书，将`(user, book_id)`作为主键亦不可靠，故另外设置一字符串`record_id`作为主键，上层模块可自由设计此id的编码方式。
- `extended`表示此次借书已经延长的次数。
- `expired_date`相对于`extended`实际上是冗余的：它可以由`lent_date`，`extended`和books中的`expire_days`计算得到；但每次查询到期日都如此般将两张表进行连接计算的性能开销与代码量也较大，故出于实际情况考虑保留一个计算好的`expired_date`方便查询。

fdulib中的结构体

Library

```
type Library struct {
    db *sqlx.DB
    //Optimally, each client should have a separate MySQL account e.g.'HGX' or
'H3'
    User string
    Password string
    DBName string
}
```

与原来不同的是把登录数据库所需信息移到了结构体中。为了更好地实现并发，每个客户端都可以拥有一个独立的数据库登录账户，如光华楼和三教各一个。

Book

```
type Book struct {
    Book_id          string
```

```
    Book_stat        int
    Title            string
    Edition          int
    Author           sql.NullString
    ISBN             sql.NullString
    Added_date       string
    Added_by_admin   string
    Expire_days      int
    Removed_date     sql.NullString
    Removed_by_admin sql.NullString
    Removed_msg      sql.NullString
}
```

在`fdulib`中进行添加或查询图书类操作时，将会以一个或一系列Book结构体作为参数，每一个结构体带有完整的一本书的信息（对应关系表books中的一条元组）。由于对应关系表中部分属性可NULL，这里使用NullString来处理。

UserHistory

```
type UserHistory struct {
    Record_id     string
    Lent_date     string
    Expire_date   string
    Returned_date sql.NullString
    Book_id       string
    Title         string
    Author        sql.NullString
}
```

在`fdulib`中进行查询用户借书历史之类操作时，将会返回一个或一系列UserHistory结构体，每一个结构体带有用户的一条借书历史，包含了基本的必要信息。

4.功能介绍

下面将逐一介绍Library系统的功能，同时进行相应的测试。

创建管理员/用户账户

fdulib

```
  func (lib *Library) CreateAdmin(user, password string) error
  func (lib *Library) GetAdminPassword(user string) (pw string, err error)
  func (lib *Library) CreateUser(user, password string) error
  func (lib *Library) GetUserPassword(user string) (pw string, err error)
```

用户名重复时由数据库报错。

fdulib_test

```
=== RUN   TestLibrary_CreateAdmin
    TestLibrary_CreateAdmin: fdulib_test.go:192: Error 1062: Duplicate entry
'admin2' for key 'admins.PRIMARY'
--- PASS: TestLibrary_CreateAdmin (0.01s)
=== RUN   TestLibrary_GetAdminPassword
--- PASS: TestLibrary_GetAdminPassword (0.00s)
=== RUN   TestLibrary_CreateUser
    TestLibrary_CreateUser: fdulib_test.go:216: Error 1062: Duplicate entry
'user2' for key 'users.PRIMARY'
--- PASS: TestLibrary_CreateUser (0.01s)
=== RUN   TestLibrary_GetUserPassword
--- PASS: TestLibrary_GetUserPassword (0.00s)
```

通过创建3个用户（其中有2个重复）并验证密码来测试，产生的2个`Error 1062`符合期望。

添加图书

fdulib

```
func (lib *Library) AddBook(book *Book) error
```

`book_id`重复时也由数据库报错，由于与上面类似，这里不再测试。

fdulib_test

```
=== RUN   TestLibrary_AddBook
--- PASS: TestLibrary_AddBook (0.03s)
```

向Library中添加6本图书。

查询图书

fdulib

```
func (lib *Library) QueryBook(field string, content string) ([]Book, error)
```

支持通过各字符串字段查找图书，支持局部关键字匹配，满足条件的图书信息以[]Book的形式返回。查询无结果时返回空的[]Book，不报错，相关提示信息可由上层模块提供，这里不必额外增加复杂性。当选择的字段非法时由数据库报错。

fdulib_test

```
=== RUN   TestLibrary_QueryBook
    TestLibrary_QueryBook: fdulib_test.go:254: Error 1054: Unknown column
'titled' in 'where clause'
--- PASS: TestLibrary_QueryBook (0.00s)
```

查询`title`字段包含"database"的图书，并核验查询结果（3本书）；查询`titled`字段包含"atabase"的图书，报出相应的错误。

删除图书

fdulib

```
func (lib *Library) RemoveBookByID(book_id, removed_date, removed_by_admin,
removed_msg string) error
```

`book_id`不存在或书已被移除时会报错。

fdulib_test

```
=== RUN   TestLibrary_RemoveBookByID
    TestLibrary_RemoveBookByID: fdulib_test.go:287: wrong book ID or book
already removed
    TestLibrary_RemoveBookByID: fdulib_test.go:293: wrong book ID or book
already removed
--- PASS: TestLibrary_RemoveBookByID (0.00s)
```

删除一本书；试图删除一本不存在的书；试图删除刚刚删除的那本书。后两个非法操作均报错。

借书

fdulib

```
func (lib *Library) LendBook(record_id, user, book_id, lent_date string) error
```

1. 更新并检查用户状态是否正常
2. 检查图书状态是否为"可用"并设置为"借出"
3. 在数据库中插入借书记录

fdulib_test

```
=== RUN   TestLibrary_LendBook
    TestLibrary_LendBook: fdulib_test.go:321: user account suspended, unable to
borrow book
    TestLibrary_LendBook: fdulib_test.go:328: book with id 'QA76.9.C643 P37
2014-1' is currently unavailable
--- PASS: TestLibrary_LendBook (0.02s)
```

正常地借出5本书；试图用一个被禁用的账户借书；试图借一本已经借出的书。后两个操作均报错。

查询借书记录

fdulib

```
func (lib *Library) QueryUserHistory(user string) ([]UserHistory, error)
```

查询一个用户的借书记录，按借书日期从早到晚排序。

fdulib_test

```
=== RUN   TestLibrary_QueryUserHistory
--- PASS: TestLibrary_QueryUserHistory (0.00s)
```

查询用户"user1"的借书记录并比对查询结果。

查询待还书目

fdulib

```
func (lib *Library) QueryPendingBook(user string) ([]UserHistory, error)
```

查询一个用户的待归还书目，按截至日期从早到晚、`title`字典序排序。

fdulib_test

```
=== RUN   TestLibrary_QueryUserHistory
--- PASS: TestLibrary_QueryUserHistory (0.00s)
```

查询用户"user1"待归还的书目，并核验返回的查询内容及顺序。

延期

fdulib

```
func (lib *Library) ExtendDeadline(user string, record_id string) error
```

检查用户状态并进行延期。遇到以下情况均会报错：

- 已经延期3次
- 用户已被禁用
- `record_id`不存在
- 图书实际已经归还

fdulib_test

```
=== RUN   TestLibrary_ExtendDeadline
    TestLibrary_ExtendDeadline: fdulib_test.go:366: book unavailable for further
extension
    TestLibrary_ExtendDeadline: fdulib_test.go:373: user account suspended,
unable to extend deadline
--- PASS: TestLibrary_ExtendDeadline (0.01s)
```

对正常用户"user2"的一本书进行4次延期，其中第4次报错；对已被禁用的用户"user1"的一本书进行延期，亦报错。

查询逾期书目

fdulib

```
func (lib *Library) QueryOverdue(user string) ([]UserHistory, error)
```

查询一个用户的逾期书目，按截至日期从早到晚排序。

fdulib_test

```
=== RUN   TestLibrary_QueryOverdue
--- PASS: TestLibrary_QueryOverdue (0.00s)
```

查询用户"user1"逾期的书目，并核验返回的查询内容。

还书

fdulib

```
func (lib *Library) ReturnBook(user, record_id, returned_date string) error
```

更新records表，更新图书状态，更新用户状态（看是否归还了一本逾期的书）。遇到以下情况报错：

- 该书已归还
- 用户未借书（借书记录不存在）

fdulib_test

```
=== RUN   TestLibrary_ReturnBook
    TestLibrary_ReturnBook: fdulib_test.go:407: book already returned or user
did not borrow book
    TestLibrary_ReturnBook: fdulib_test.go:414: book already returned or user
did not borrow book
--- PASS: TestLibrary_ReturnBook (0.08s)
```

归还"user1"借的4本书；试图归还一本已经归还的书；试图归还一本没有借的书。后两个操作均报错。

更新用户状态

fdulib

```
func (lib *Library) UpdateUserStatus(user string, date string) (user_stat int,
err error)
```

检查用户当前逾期图书的数目，并更新用户状态。实际使用中可以由上层模块定时调用。由于已被`LendBook()`调用（借书前检查用户状态）并测试，在此不再测试。

5.功能验证方法

- 安装`fdulib`
- 在fdulib_test.go的第12行，将访问MySQL所需账户信息按实际情况修改：

  ```
  var lib = Library{User: "fan", Password: "", DBName: "ass3"}
  ```

- 测试时会将`DBName`指定的数据库**清空**重置！请务必确保该数据库闲置可用。
- 运行`go test fdulib`

6.SampleLib

这是一个带有基础功能的用户界面，下层与`fdulib`对接，展示了与`fdulib`相对接的上层模块的一种可能实现。访问数据库所需登录信息在main.go的14~18行设置：

```
const (
    User        = "fan"
    Password    = ""
    DBName      = "ass3"
)
```

启动`SampleLib`时不会将数据库清空。

特点

- 相比于`fdulib`，其主要增加的功能在于区分了账户，admins和users可执行的操作不同，每个user也只能为自己执行操作而不能替别的用户操作。即便是调用`fdulib`中的同一个函数，admin和user看到的信息也可能不一样，如一些管理信息会对普通用户隐藏。已删除图书的相关信息对普通用户不可见而对管理员可见。
- 绝大部分使用的读入函数均使用了wrapper，大幅减少代码量的同时也不放过error handling。
- 为方便测试，在程序启动时自动创建管理员账户`root`，密码为`1905`。可由此账户开始浏览。

> 建议在`go test fdulib`之后再运行SampleLib，此时数据库有保留有test中导入的数据，比较方便测试。

7.致谢

感谢韩晓宇关于数据库建表的建议！

感谢胡志峰助教的支持与体谅！