|  |
| --- |
| **课 程 设 计 任 务 书**   1. 课程设计题目：　　　　　　　数据库系统课程设计 2. 设计原始资料：　　　　　　　《数据库系统概论》       设计应解决下列各主要问题：　　1、设计特定的数据结构，用于存储数据表、视图、索引等数据库对象的信息，即建 立数据库系统的数据字典；  　　2、设计特定的数据结构，用于存储数据表中的数据  　　3、设计特定的数据结构，分别用于存储用户和访问权限的信息；  　　4、输入“help database”命令，输出所有数据表、视图和索引的信息，同时显示 其对象类型；输入“help table 表名”命令，输出数据表中所有属性的详细信息；输入 “help view 视图名”命令，输出视图的定义语句；输入“help index 索引名”命令， 输出索引的详细信息；  5、解析 CREATE、SELECT、INSERT、DELETE、UPDATE 等 SQL 语句的内容；  6、检查 SQL 语句中的语法错误和语义错误；  7、执行 CREATE 语句，创建数据表、视图、索引等数据库对象；创建数据表时需要 包含主码、外码、唯一性约束、非空约束等完整性约束的定义；  8、执行 GRANT 语句，为用户授予 SELECT、INSERT、DELETE、UPDATE 权限；执行 REVOKE 语句，收回上述权限；  9、用户登录时，需要输入用户名；如果用户没有被授权，则拒绝执行用户查询或更 新操作，并给出提示信息；  10、执行 SELECT 语句，从自主设计的数据表中查询数据，并输出结果；在 SELECT 语句中需要支持 GROUP BY、HAVING 和 ORDER BY 子句，需要支持 5 种聚集函数；  11、执行 INSERT、DELETE 和 UPDATE 语句，更新数据表的内容；更新过程中需要检查更新后的数据表是否会违反参照完整性约束。如果是，则提示违反哪一条完整性约束，并拒绝执行更新操作；如果否，提示数据表更新成功，并说明插入、删除或修改了几个元组。   1. 设计说明书应附有下列图纸：　　　　模块调用图        1. 小组分工说明：           六、命题发出日期：　 2018-10-01　　　设计完成日期：　 2018-11-29 |
| **指导教师评语**  **成绩：**  **指导教师（签章）：**  **年　　月　　日** |

[1.需求分析 3](#_Toc531626690)

[1.1概述 3](#_Toc531626691)

[1.2基本功能 3](#_Toc531626692)

[1.3 系统需求 4](#_Toc531626693)

[2. 概要设计说明 4](#_Toc531626694)

[2.1模块功能及描述 4](#_Toc531626695)

[2.2 模块调用图 5](#_Toc531626696)

[3.详细设计说明 6](#_Toc531626697)

[3.1 环境依赖 6](#_Toc531626698)

[3.2 数据存储 6](#_Toc531626699)

[3.3 run函数的功能 6](#_Toc531626700)

[3.3 初始化函数的功能 7](#_Toc531626701)

[3.4 用户验证及登录模块 7](#_Toc531626702)

[3.5 创建表模块 9](#_Toc531626703)

[3.6 创建视图模块 10](#_Toc531626704)

[3.7 插入模块 11](#_Toc531626705)

[3.8 更新数据 13](#_Toc531626706)

[3.9 语句处理函数 14](#_Toc531626707)

[3.10 删除模块 17](#_Toc531626708)

[3.11 权限检查模块 17](#_Toc531626709)

[3.12 权限的赋予和回收模块 18](#_Toc531626710)

[3.13 约束检查模块 19](#_Toc531626711)

[3.14 查询模块 20](#_Toc531626712)

[3.15 索引模块 23](#_Toc531626713)

[3.16 help xxx模块 26](#_Toc531626714)

[3.17 数据库选择和创建模块 27](#_Toc531626715)

[4.用户使用说明 27](#_Toc531626716)

[5.课程设计感想 31](#_Toc531626717)

1.需求分析

# 1.1概述

从底层做起，实现数据库的组织、存储、检索、更新和索引等功能。

# 1.2基本功能

1. 设计特定的数据结构，用于存储数据表、视图、索引三种数据库对象的元数据信息，建立数据库系统的数据字典
2. 设计特定的数据结构，用于存储数据表中的数据；
3. 设计特定的数据结构，用于存储索引数据；
4. 设计特定的数据结构，分别用于存储用户和访问权限的信息；
5. 输入“help database”命令，输出所有数据表、视图和索引的信息，同时显示其对象类型；输入“help table 表名”命令，输出数据表中所有属性的详细信息；输入“help view 视图名”命令，输出视图的定义语句；输入“help index 索引名”命令，输出索引的详细信息；
6. 解析CREATE、SELECT、INSERT、DELETE、UPDATE等SQL语句的内容；检查SQL语句中的语法错误和语义错误；
7. 执行CREATE语句，创建数据表、视图、索引三种数据库对象；创建数据表时需要包含主码外码、唯一性约束、非空约束等完整性约束的定义；
8. 执行SELECT语句，从自主设计的数据表中查询数据，并输出结果；在SELECT语句中需要支持GROUP BY、HAVING和ORDER BY子句，需要支持5种聚集函数；
9. 执行INSERT、DELETE和UPDATE语句，更新数据表的内容；更新过程中需要检查更新后的数据表是否会违反参照完整性约束。如果是，则提示违反哪一条完整性约束，并拒绝执行更新操作；如果否，提示数据表更新成功，并说明插入、删除或修改了几个元组。
10. 当数据表的内容更新后，根据索引的定义，自动更新索引表的内容；
11. 在有索引的数据表上执行查询语句时，首先利用索引找到满足条件的元组指针，然后通过指针到数据表中取出相应的元组；
12. 执行GRANT语句，为用户授予对某数据库对象的SELECT、INSERT、DELETE、UPDATE等权限；执行REVOKE语句，收回上述权限；
13. 用户登录时，需要输入用户名；如果用户没有被授权，则拒绝执行用户查询或更新等操作，并给出提示信息；
14. 将SELECT语句转化为关系代数表达式，再利用查询优化算法对关系代数表达式进行优化，输出优化后的关系代数表达式或SELECT语句

# 1.3 系统需求

开发语言：Python

开发工具：Pycharm

2. 概要设计说明

# 2.1 数据文件存储

设置system库，table\_information库，view库为系统初始化后建立的初始数据库。

文件使用excel存储。

# 2.2 模块功能及描述

**2.1.1 run函数模块**

run函数模块是整个系统的入口，本模块的主要功能是初始化用户，调用各模块，处理用户输入，实现数据库管理功能。

**2.1.2用户验证及登录模块**

本模块功能是进行用户的登录认证。

**2.1.3创建表模块**

根据标准 SQL 语言将输入语句进行分割，获得表名，各个属性名，属性类型，约束条件等内容，再进行表的创建。

**2.1.4 创建视图模块**

本模块通过 CREATE VIEW VIEW\_NAME AS SELECT …语句来创建视图，将存储视图和对应的语句。

**2.1.5 插入数据模块**

将用户输入的数据存入相应的表中并进行约束检查。

**2.1.6 更新数据模块**

根据用户的 WHERE 条件更新符合条件的元组并进行约束检查。

**2.1.7 查询数据模块**

根据用户的查询条件进行查询。

**2.1.8 删除数据模块**

根据用户的 WHERE 条件删除符合条件的元组。

**2.1.9 帮助模块**

HELP DATABASE 可查看当前数据库下的所有的表，视图信息。

help table 表名可输出数据表中所有属性的详细信息；

输入“help view 视图名”命令，输出视图的定义语句；

输入“help index 索引名”命令，输出索引的详细信息；

**2.1.10 授权与权限收回模块**

通过grant和revork实现用户对表和视图的 CREATE、DELETE、UPDATE、INSERT 四种操作的权限授予与收回。

**2.1.11 索引建立和更新模块**

使用create语句建立索引，在内容更新后更新索引。

# 2.3 模块调用图

用户验证及登录模块

Grant/Revoke模块

Update模块

run函数模块

Insert模块

Helpdatabase模块mo

Delete模块

Select模块

Create模块

索引建立和更新模块

3.详细设计说明

# 3.1 环境依赖

本程序用到的库：

from openpyxl import \*

import os

import re

from index import \*

from prettytable import PrettyTable

import hashlib

import bisect

import Queue

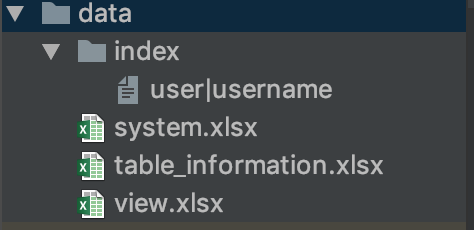
# 3.2 数据存储

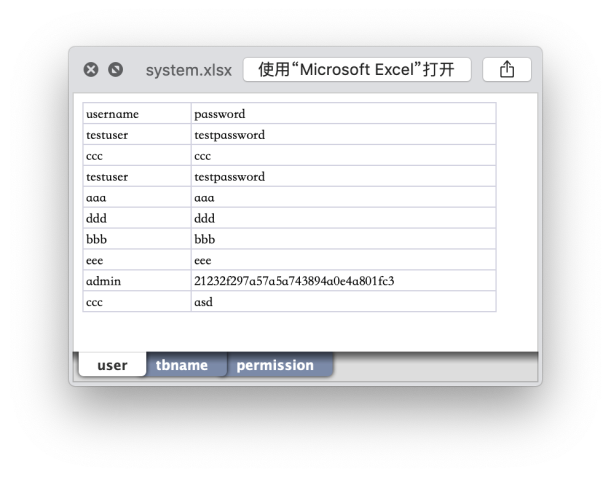
数据库的数据使用xlsx格式存储，每一个文件对应一个数据库，工作簿对应库中的表，表结构和文件结构对应。

索引使用文本存储dict。

使用openpyxl库进行文件内容的操作。

数据库示例：





# 3.3 run函数的功能

\* 程序初始化

\* 打印欢迎语句

\* 处理用户登录与认证

\* 处理帮助命令和退出命令

def run():

Initialization()

welcome()

login()

while True:

command = get\_command()

#print command

if command == 'quit' or command == 'exit':

print("[🍻] Thanks for using L-DBMS. Bye~~")

exit(0)

elif command == 'help':

help()

else:

query(command)

# 3.3 初始化函数的功能

\* 创建数据存储目录

\* 创建系统数据库文件

\* 创建系统用户

\* 赋予系统用户权限

def Initialization():

if not os.path.exists(db\_path):

os.mkdir(db\_path)

if not os.path.exists("data/table\_information.xlsx"):

Workbook().save("data/table\_information.xlsx")

if os.path.exists("data/system.xlsx"):

print "Initializating......"

else:

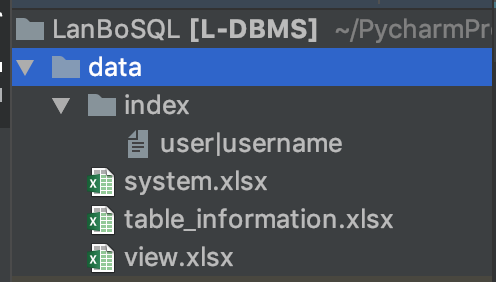
creat\_db('system')

db = load\_workbook("data/system.xlsx")

permission\_tb\_col = ['database char[50] pk unique','select char','insert char','delete char','update char']

creat\_table('permission', db, 'system',permission\_tb\_col)

目录结构：



# 3.4 用户验证及登录模块

登陆处理：

通过raw\_input()函数获取用户输入的用户名和密码，交给check\_login()函数来验证是否正确。如果check通过，输出欢迎界面并将全局变量user赋值。

def login():

global user

print "Please Login:"

username = raw\_input("username: ")

password = raw\_input("password: ")

if check\_login(username,password):

print "Login Success!Welcome {}! 😊".format(username)

user = username

else:

print "user not exist or password is wrong!😣 Try again."

login()

check\_login函数通过输入的username查询数据库中对应的密码，将查询结果和用户输入的值的md5加密值进行比较，如果一致，认为成功登陆。

如果密码不正确或无此用户，都输出error并要求用户再一次输入

def check\_login(username,password):

db = load\_workbook("data/system.xlsx")

#right\_pswd = select(password,user,{'username':username})

table = db['user']

col\_list = list(iter\_cols(table))

try:

pos = col\_list[0].index(username)

except:

return False

right\_pswd = col\_list[1][pos]

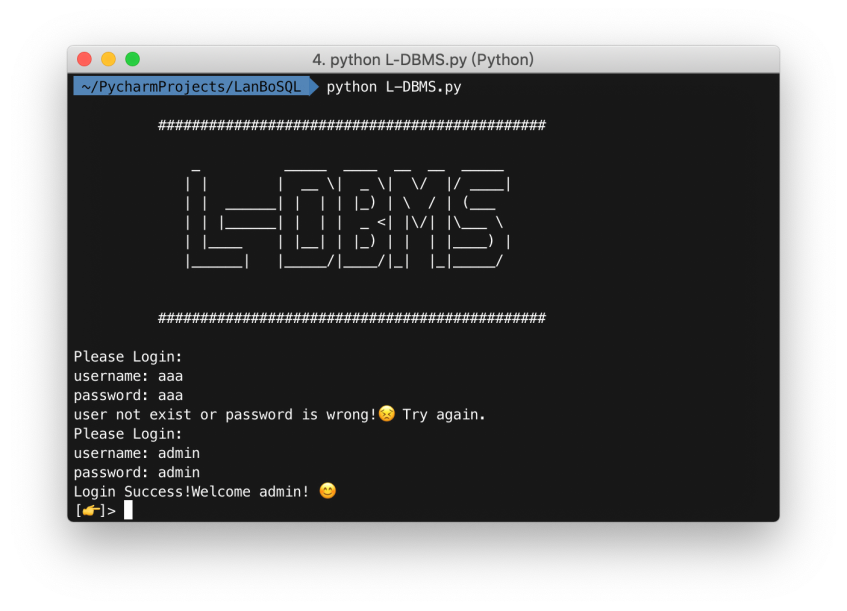
if hashlib.md5(password).hexdigest() == right\_pswd:

return True

else:

return False

登陆效果：



# 3.5 创建表模块

语句：

create table tbname (id int PK null,user char[10] )

在创建时将约束写入table\_information表中。

def creat\_table(table\_name,current\_database,current\_dbname,columns\_list):

# create table

if table\_name not in current\_database.sheetnames:

table = current\_database.create\_sheet(table\_name)

else:

print (u"数据表已存在,请重新输入.")

return

if current\_database.worksheets[0].title == 'Sheet':

del current\_database['Sheet']

#表创建完成，开始创建列

length = len(columns\_list)

#print length

tbinfo = load\_workbook("data/table\_information.xlsx")

tbinfo\_tb = tbinfo[current\_dbname]

tbinfo\_rows = tbinfo\_tb.max\_row

column\_names = []

for i in range(length): #将字段的属性写到table\_information库中

column = columns\_list[i].split(' ')

tbinfo\_tb.cell(row=tbinfo\_rows+1+i,column=1).value = table\_name

tbinfo\_tb.cell(row=tbinfo\_rows+1+i, column=2).value = column[0]

tbinfo\_tb.cell(row=tbinfo\_rows+1+i, column=3).value = column[1]

for key in column[2:]:

if key == 'null':

tbinfo\_tb.cell(row=tbinfo\_rows + 1 + i, column=4).value = '1'

elif key == 'not\_null':

tbinfo\_tb.cell(row=tbinfo\_rows + 1 + i, column=4).value = '0'

elif key == 'unique':

tbinfo\_tb.cell(row=tbinfo\_rows + 1 + i, column=5).value = '1'

elif key == 'pk':

tbinfo\_tb.cell(row=tbinfo\_rows + 1 + i, column=6).value = '1'

elif key == 'fk':

tbinfo\_tb.cell(row=tbinfo\_rows + 1 + i, column=7).value = '1'

column\_names.append(column[0])

for j in range(1, 8):

if tbinfo\_tb.cell(row=tbinfo\_rows + 1 + i, column=j).value is None:

tbinfo\_tb.cell(row=tbinfo\_rows + 1 + i, column=j).value = 'NULL'

tbinfo.save("data/table\_information.xlsx")

for i in range(length):

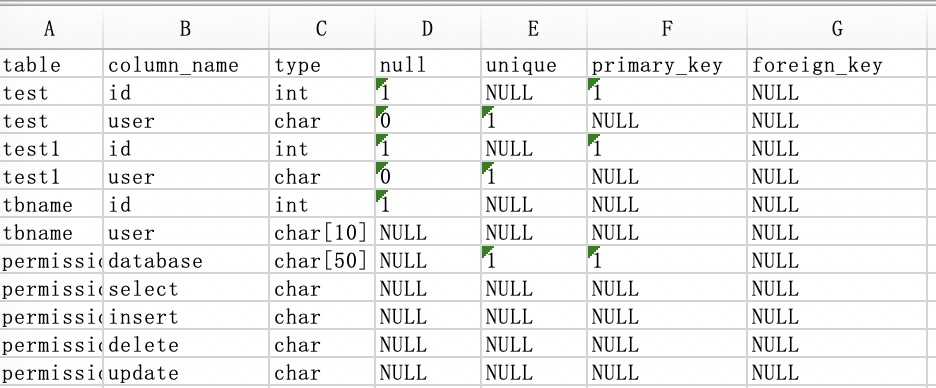
table.cell(row=1,column=i+1).value = column\_names[i] #表第一行是列名

current\_dbname = db\_path + current\_dbname + '.xlsx'

current\_database.save(current\_dbname)

print (u"数据表创建完成。")

table\_information表（测试数据）



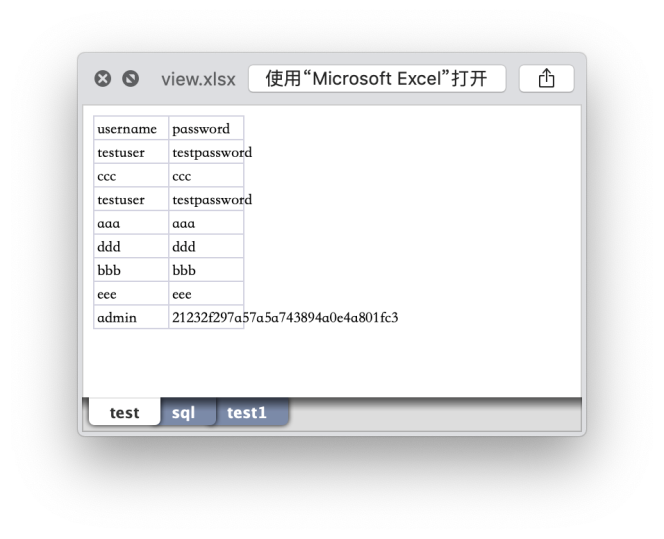
# 3.6 创建视图模块

语句：creat view view\_name as select xx from xx

解析查询语句，将结果存在view库中

每一个视图已视图名作为表名写入view库中

在view库中设置一个sql表原来存储视图和对应的sql语句



def view(viewname,sql):

"""

file = view\_path + viewname

view = query(sql,'view')

f = open(file, "w")

f.write(str(view))

f.close()

print "Success"

"""

db = load\_workbook("data/view.xlsx")

if viewname not in db.sheetnames:

table = db.create\_sheet(viewname)

else:

print ("view is exist.")

return

if db.worksheets[0].title == 'Sheet':

del db['Sheet']

sql\_table = db['sql']

maxrow = sql\_table.max\_row #在sql表中存view名和对应的sql语句

sql\_table.cell(row=maxrow + 1, column = 1).value = viewname

sql\_table.cell(row=maxrow + 1, column = 2).value = sql

table = db[viewname]

views = query(sql, 'view')

for i in range(len(views)):

for j in range(len(views[i])):

table.cell(row=i+1, column=j+1).value = views[i][j]

db.save("data/view.xlsx")

# 3.7 插入模块

支持单组和多组的插入

单组的插入处理就是将sql语句中的要插入的数据处理成字典，再交给insert函数处理。

多组的话将每组字典加入数组再进行处理。

在插入数据前通过check\_ Constraint函数进行约束检查

注释部分是用来处理嵌套语句的，逻辑是通过正则取出其中的子查询语句，带上tag参数交给query函数处理，将结果以数组形式返回。因为对表的操作也是先转化成数组，这样直接处理数组就ok

Query处理语句的部分：



def insert(table\_name, current\_database, current\_dbname, columns\_list):

if not check\_Constraint(columns\_list,table\_name): #columns应为[dict]

print "Constraint Error"

return False

table = current\_database[table\_name]

for columns in columns\_list:

table\_rows = table.max\_row

table\_columns = table.max\_column

length = len(columns)

# print length

for i in range(length):

column = re.search('\((.\*?)\)', columns[i], re.S).group(1)

column\_list = column.split(',')

chk\_len = len(column\_list)

if chk\_len != table\_columns:

print ('插入失败，请检查输入的数据数量是否与列数量对应。')

return

else:

for j in range(chk\_len):

table.cell(row=table\_rows + i + 1, column=j + 1).value = column\_list[j]

current\_dbname = db\_path + current\_dbname + '.xlsx'

current\_database.save(current\_dbname)

print ("数据插入完成。")

def check\_Constraint(columns,tablename): #columns={'a':'xx'}

db = load\_workbook("system/table\_information.xlsx")

table = db[using\_dbname]

rows = []

rows\_list = list(iter\_rows(table)) #所有行

cols\_list = list(iter\_cols(table))

for col in columns:

value = columns[col]

for i in range(len(cols\_list[0])): #table对应的行

if cols\_list[0][i] == tablename:

rows.append(i)

for line in rows:

if rows\_list[line][1] == col:

typee, is\_null, unique, pk, fk = rows\_list[line][2:]

if is\_null == '0':

if value == '':

return False

if unique == '1':

if not check\_unique(tablename,col,value):

return False

if pk == '1':

if not check\_unique(tablename,col,value) or value == '':

return False

if '[' in typee:

typee, maxlen = re.findall('(\w\*)\[(\d\*)\]',type) #int[10] => int,10

else:

maxlen = 1000

if len(value) > maxlen:

return False

if typee == 'int':

if type(value) != type(1):

return False

if typee == 'char':

if type(value) != type('c'):

return False

# 3.8 更新数据

支持单组和多组的更新

单组的更新处理就是将sql语句中的要插入的数据处理成字典，再交给insert函数处理。

多组的话将每组字典加入数组再进行处理。

在更新数据前通过check\_ Constraint函数进行约束检查

其他功能和insert的逻辑一样

语句：

UPDATE table\_name SET column1=value1,column2=value2,... WHERE some\_column=some\_value;

def update(table\_name,current\_database,current\_dbname,columns\_list,update\_columns\_list):

if not check\_Constraint(update\_columns\_list,table\_name): #columns应为dict

print "Constraint Error"

return False

table = current\_database[table\_name]

table\_rows = table.max\_row # 行

table\_columns = table.max\_column # 列

length = len(columns\_list)

update\_row\_num = [x for x in range(2,table\_rows+1)]

columns\_name = []

for cell in list(table.rows)[0]:

columns\_name.append(cell.value)

for key in columns\_list:

flag = 0

for i in range(len(columns\_name)): # 判断colmuns\_list 是否有 not in colmus中的

if columns\_name[i] == key:

flag = 1

if flag == 0: # 输入的列名不存在

print("Unknown column '{}' in 'where clause'".format(key))

return

for key in columns\_list:

column\_num = columns\_name.index(key)

for i in update\_row\_num[::-1]: #倒着来

if table.cell(row=i, column=column\_num+1).value != columns\_list[key]:

update\_row\_num.remove(i)

if len(update\_row\_num) > 0:

for i in update\_row\_num[::-1]:

for j in range(1,table\_columns+1):

clu\_name = table.cell(row=1, column=j).value

table.cell(row=i, column=j).value = update\_columns\_list[clu\_name]

else:

print("find 0 to update.")

current\_database.save(db\_path + current\_dbname + '.xlsx')

print("更新完成，影响{}行".format(len(update\_row\_num)))

# 3.9 语句处理函数

\* ql语句的处理与解析。

\* 通过切词和正则来转化数据和调用各函数。

\* 通过split将语句按照空格切成数组，先根据首词判断操作再细分。

\* tag参数是为insert，view等函数需要用到查询结果但不输出是的标识，如果带着该参数调用select函数，不会打印结果而是将结果以数组返回。

\* select操作的谓词通过predicate和symbol参数来标识，带着这两个参数调用select函数，具体参见select模块

\* 如果都没匹配最后会输出错误

def query(sql,tag=''):

sql\_word = sql.split(" ")

if len(sql\_word) < 2:

print "[!] Wrong query!"

return

operate = sql\_word[0].lower()

if operate == 'use':

if sql\_word[1] == 'database':

try:

use\_db(sql\_word[2])

except:

print "[!]Error"

else:

print "[!]Syntax Error.\neg:>use database dbname"

elif operate == 'create':

if sql\_word[1] == 'database':

try:

creat\_db(sql\_word[2])

except:

print "[!]Create Error"

elif sql\_word[1] == 'table':

columns\_list = re.findall('\((.\*)\)', sql)[0].split(',')

print columns\_list, using\_dbname

try:

creat\_table(sql\_word[2], using\_db, using\_dbname, columns\_list)

except:

print "[!]Error"

elif sql\_word[1] == 'view': #creat view test1 as select \* from user

viewname = sql\_word[2]

sql = ' '.join(sql\_word[4:])

view(viewname,sql)

elif sql\_word[1] == 'index':

return

else:

print "[!]Syntax Error."

elif operate == 'select':

pos = 0

for i in range(len(sql\_word)):

if '(' in sql\_word[i] and 'select' in sql\_word[i]:

pos = i

if pos == 3:

sql2 = sql\_word[3][1:-1]

query(sql2,tag='nesting')

sql\_word[3] = 'tmp'

sql = ' '.join(sql\_word)

columns = sql\_word[1]

table\_name = sql\_word[3]

if len(sql\_word) > 4:

#try:

limit = sql\_word[5].split()

predicate = 'and'

symbol = '='

if ',' in sql\_word[5]:

limit = sql\_word[5].split(',')

predicate = 'and'

elif '|' in sql\_word[5]:

limit = sql\_word[5].split('|')

predicate = 'or'

elif '>' in sql\_word[5]:

#limit = sql\_word[5].split()

symbol = '>'

elif '<' in sql\_word[5]:

#limit = sql\_word[5].split()

symbol = '<'

elif len(sql\_word) > 6:

if sql\_word[6] == 'in':

limit = [sql\_word[5] + '=' + sql\_word[7]]

predicate = 'in'

if sql\_word[6] == 'like':

limit = [sql\_word[5] + '=' + sql\_word[7]]

predicate = 'like'

#except:

#limit = [].append(sql\_word[5])

#print limit

for i in range(len(limit)):

limit[i] = limit[i].split(symbol)

limit = dict(limit)

return select(columns, table\_name, limit, predicate=predicate, symbol=symbol, tag=tag)

else: #没where的情况

return select(columns, table\_name, tag=tag)

elif operate == 'grant':

set\_permission(sql\_word[5], sql\_word[3], sql\_word[1])

elif operate == 'revoke':

del\_permission(sql\_word[5], sql\_word[3], sql\_word[1])

elif operate == 'insert': #INSERT INTO table\_name col1=val1,col2=val2&col3=val3,col4=val4

table\_name = sql\_word[2]

columns\_list = []

if '&' in sql:

cols = sql\_word[3].split('&') #[{xx},{xx}] 多组

for p in range(len(cols)):

col = cols[p]

c = col.split(',')

for i in range(len(c)):

c[i] = c[i].split('=')

cols[p] = dict(c)

columns\_list = cols

else:

cols = sql\_word[3].split(',')

for i in range(len(cols)):

cols[i] = cols[i].split('=')

columns\_list.append(dict(cols))

insert(table\_name,using\_db,using\_dbname,columns\_list)

elif operate == 'update':

return

elif operate == 'help':

if sql\_word[1] == 'database':

show\_db()

if sql\_word[1] == 'table':

usdbnm = using\_dbname

use\_db('table\_information')

tbname = sql\_word[2]

select('\*',usdbnm,{'table':tbname})

if sql\_word[1] == 'view':

view\_name = sql\_word[2]

use\_db('view')

select('sql','sql',{'viewnamw':view\_name})

if sql\_word[1] == 'index':

print "All Index:"

indexs = os.listdir('data/index/') # 第二种方法，从保存数据库信息的库中查询

for index in indexs:

if '.DS' not in index:

print "[\*] " + index[:-5]

else:

print "[!]Syntax Error."

# 3.10 删除模块

会先check用户权限再进行操作。

其他逻辑类似insert函数

def delect(table\_name,current\_database,current\_dbname,columns\_list): #columns\_list={'name1':'value1','name2':'value2'}

table = current\_database[table\_name]

table\_rows = table.max\_row #行

table\_columns = table.max\_column #列

length = len(columns\_list)

delect\_row\_num = [x for x in range(2,table\_rows+1)]

columns\_name=[]

for cell in list(table.rows)[0]:

columns\_name.append(cell.value)

for key in columns\_list:

flag = 0

for i in range(len(columns\_name)): #判断colmuns\_list 是否有 not in colmus中的

if columns\_name[i] == key:

flag = 1

if flag == 0: #输入的列名不存在

print("Unknown column '{}' in 'where clause'".format(key))

return

for key in columns\_list:

column\_num = columns\_name.index(key)

for i in delect\_row\_num[::-1]: #倒着来

if table.cell(row=i, column=column\_num+1).value != columns\_list[key]:

delect\_row\_num.remove(i)

if len(delect\_row\_num) > 0:

for i in delect\_row\_num[::-1]:

#print i,table\_rows

table.delete\_rows(int(i))

else:

print("find 0 to delect.")

current\_database.save(db\_path + current\_dbname + '.xlsx')

print("删除完成，影响{}行".format(len(delect\_row\_num)))

# 3.11 权限检查模块

在用户对某对象进行操作之前确定该用户有没有操作权限。

用户对对象的操作权限存储在system库中的permission表

在用户进行相关操作时先去查询该用户有没有该操作的权限

def check\_permission(user,database,action):

table = load\_workbook("data/system.xlsx")['permission']

db\_list = list(iter\_cols(table))[0][1:]

row = db\_list.index(database)+2

action\_list = list(iter\_rows(table))[0]

col = action\_list.index(action)+1

allow\_user = table.cell(row=row, column=col).value.split(',')

if user in allow\_user:

return True

else:

print "Permission not allowed"

return False

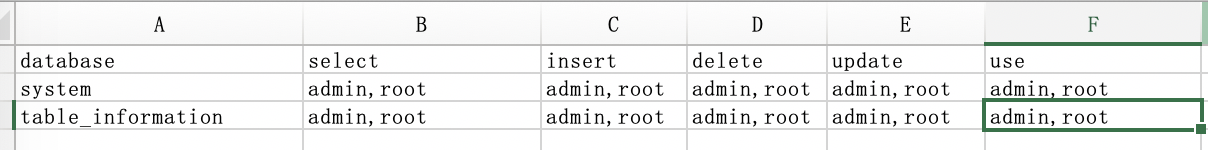
# 3.12 权限的赋予和回收模块

使用grant和revoke关键字赋予权限和回收权限

实质是对permision表中数据进行更新

grant语句

grant select on test\_tb for testuser



（这里第一列是具有权限的对象，不只是数据库，可以是table,view,index，在函数对应函数处理时将数据库名变量换成其他对象就OK）

def set\_permission(user,database,action):

db = load\_workbook("data/system.xlsx")

table = db['permission']

db\_list = list(iter\_cols(table))[0][1:]

row = db\_list.index(database) + 2

action\_list = list(iter\_rows(table))[0]

col = action\_list.index(action) + 1

allow\_user = table.cell(row=row, column=col).value.split(',')

if user in allow\_user:

print "user have this permission"

else:

table.cell(row=row, column=col).value = table.cell(row=row, column=col).value + ',' + user

db.save("data/system.xlsx")

revoke语句

revoke select on test\_tb for testuser

def del\_permission(user,database,action):

db = load\_workbook("data/system.xlsx")

table = db['permission']

db\_list = list(iter\_cols(table))[0][1:]

row = db\_list.index(database) + 2

action\_list = list(iter\_rows(table))[0]

col = action\_list.index(action) + 1

allow\_user = table.cell(row=row, column=col).value.split(',')

if user in allow\_user:

if allow\_user.index(user) == 0:

table.cell(row=row, column=col).value = table.cell(row=row, column=col).value.replace(user, '')

else:

table.cell(row=row, column=col).value = table.cell(row=row, column=col).value.replace(',' + user, '')

db.save("data/system.xlsx")

else:

print "user didn't have this permission"

# 3.13 约束检查模块

检查主码、外码、唯一性约束、非空约束等完整性约束。

单独的约束检查函数，在进行insert,update等操作时直接调用就OK

def check\_Constraint(columns,tablename): #columns={'a':'xx'}

db = load\_workbook("system/table\_information.xlsx")

table = db[using\_dbname]

rows = []

rows\_list = list(iter\_rows(table)) #所有行

cols\_list = list(iter\_cols(table))

for col in columns:

value = columns[col]

for i in range(len(cols\_list[0])): #table对应的行

if cols\_list[0][i] == tablename:

rows.append(i)

for line in rows:

if rows\_list[line][1] == col:

typee, is\_null, unique, pk, fk = rows\_list[line][2:]

if is\_null == '0':

if value == '':

return False

if unique == '1':

if not check\_unique(tablename,col,value):

return False

if pk == '1':

if not check\_unique(tablename,col,value) or value == '':

return False

if '[' in typee:

typee, maxlen = re.findall('(\w\*)\[(\d\*)\]',type) #int[10] => int,10

else:

maxlen = 1000

if len(value) > maxlen:

return False

if typee == 'int':

if type(value) != type(1):

return False

if typee == 'char':

if type(value) != type('c'):

return False

def check\_unique(tablename,column,value):

table = using\_db[tablename]

col\_pos = list(iter\_rows(table))[0].index(column) #第几列

cols\_list = list(iter\_cols(table))[col\_pos][1:]

if cols\_list.count(value) > 1: #该列中该值数量

return False

else:

return True

# 3.14 查询模块

\* 支持嵌套查询，and,or,in,like谓词。

\* 通过eval函数处理比较运算和数学运算

\* 通过select等关键字个数判断子查询，子查询通过tag参数调用query函数获得查询结果数组

\* query函数中语句的处理很关键，通过一些参数告诉select函数要返回什么样的值。提前讲语句处理成dict或list讲谓词等转换成符号方便select函数的处理。

\* 查询时转换成数组方便操作

语句处理：

elif operate == 'select':

pos = 0

for i in range(len(sql\_word)):

if '(' in sql\_word[i] and 'select' in sql\_word[i]:

pos = i

if pos == 3:

sql2 = sql\_word[3][1:-1]

query(sql2,tag='nesting')

sql\_word[3] = 'tmp'

sql = ' '.join(sql\_word)

columns = sql\_word[1]

table\_name = sql\_word[3]

if len(sql\_word) > 4:

#try:

limit = sql\_word[5].split()

predicate = 'and'

symbol = '='

if ',' in sql\_word[5]:

limit = sql\_word[5].split(',')

predicate = 'and'

elif '|' in sql\_word[5]:

limit = sql\_word[5].split('|')

predicate = 'or'

elif '>' in sql\_word[5]:

#limit = sql\_word[5].split()

symbol = '>'

elif '<' in sql\_word[5]:

#limit = sql\_word[5].split()

symbol = '<'

elif len(sql\_word) > 6:

if sql\_word[6] == 'in':

limit = [sql\_word[5] + '=' + sql\_word[7]]

predicate = 'in'

if sql\_word[6] == 'like':

limit = [sql\_word[5] + '=' + sql\_word[7]]

predicate = 'like'

#except:

#limit = [].append(sql\_word[5])

#print limit

for i in range(len(limit)):

limit[i] = limit[i].split(symbol)

limit = dict(limit)

return select(columns, table\_name, limit, predicate=predicate, symbol=symbol, tag=tag)

else: #没where的情况

return select(columns, table\_name, tag=tag)

select函数，不仅仅提供查询的功能，还用来处理view的一部分数据,通过tag参数来标识数据如何处理。

查询函数：

def select(columns,table\_name,limit={},predicate='and', symbol='=', tag=''): #{'c':'x','d':'x'}

if using\_dbname == '':

print "please choose databse!"

return

table = using\_db[table\_name]

#print columns

if columns == '\*' and len(limit) == 0:

columns\_name = list(iter\_rows(table))[0]

table\_print = PrettyTable(columns\_name)

for i in range(1,len(list(iter\_rows(table)))):

table\_print.add\_row(list(iter\_rows(table))[i])

table\_print.reversesort = True

if tag == 'view':

print table\_print

return list(iter\_rows(table)) #view

else:

print(table\_print)

else:

sel\_cols = columns.split(',') #\*的情况

rows\_list = list(iter\_rows(table)) #所有的行

cols = rows\_list[0]

col\_pos = []

limit\_pos = []

print\_row = []

limit\_cols = list(limit)

symbol = '==' if symbol == '=' else symbol

if columns[0] != '\*':

for i in range(len(sel\_cols)):

col\_pos.append(cols.index(sel\_cols[i])) #要查的列的列号

else:

sel\_cols = list(iter\_rows(table))[0]

col\_pos = range(len(cols))

for i in range(len(limit)):

limit\_pos.append(cols.index(limit\_cols[i])) #where的列

for i in range(1, len(rows\_list)):

match = 0

if predicate == 'in':

match\_list = limit[limit\_cols[0]]

for j in len(match\_list):

if rows\_list[i][limit\_pos[0]] == match\_list[j]:

print\_row.append(i)

if predicate == 'like':

like\_word = re.findall('(.\*)\%',limit[limit\_cols[0]])

if like\_word in rows\_list[i][limit\_pos[0]]:

print\_row.append(i)

else:

for j in range(len(limit\_pos)): #通过eval实现比较运算

if eval("'" + rows\_list[i][limit\_pos[j]] + "'" + symbol + "'" + limit[limit\_cols[j]] + "'"):

match += 1

if predicate == None:

print\_row.append(i)

if predicate == 'and' and match == len(limit\_pos): #and时要全部匹配

print\_row.append(i) #符合条件的行号

if predicate == 'or' and match > 0: #or时至少一个匹配

print\_row.append(i)

table\_print = PrettyTable(sel\_cols)

for i in range(len(print\_row)):

add\_rows = []

for x in col\_pos:

add\_rows.append(rows\_list[print\_row[i]][x])

table\_print.add\_row(add\_rows)

table\_print.reversesort = True

if tag == 'view':

return table\_print

elif tag == 'nesting':

tmpdb = using\_db

table = tmpdb['tmp']

for i in range(len(sel\_cols)):

table.cell(row=0,column=i+1).value = sel\_cols[i]

for i in range(len(print\_row)):

add\_rows = []

for x in col\_pos:

add\_rows.append(rows\_list[print\_row[i]][x])

for j in range(len(add\_rows)):

table.cell(row=i+2,column=j+1).value = add\_rows[j]

tmpdb.save("data/" + using\_dbname + ".xlsx")

else:

#table\_print.reversesort = True

print(table\_print)

# 3.15 索引模块

B+树的索引，将数据先处理成数组，每一组数据包含数据值和数据在表中的位置（起到指针的作用）。B+树的处理单独写了一个类，方便调用处理。

def index(current\_database,table\_name,column\_name):

table = current\_database[table\_name]

table\_columns = table.max\_column

table\_rows = table.max\_row

column\_num = 0

column\_value = []

column\_position = []

for i in range(1,table\_columns+1):

if table.cell(row=1,column=i).value == column\_name:

column\_num = i

if column\_num == 0:

print "no this column"

return

else:

for i in range(2,table\_rows+1):

column\_value.append(str(table.cell(row=i,column=column\_num).value))

column\_position.append('<{},{}>'.format(i,column\_num))

column\_value.sort()

for i in range(len(column\_value)):

tmp = [column\_value[i],column\_position[i]] #like [1,aaa|<2,1>]

column\_value[i] = tuple(tmp) #like [(1,aaa|<2,1>)]

#print column\_value[0]

bt = test\_BPTree(column\_value)

indexname = table\_name + '|' +column\_name

save\_index(str(bt), indexname)

def save\_index(bt,indexname):

line = re.findall(r'\[.\*?\]', bt)

for i in range(len(line)):

line[i] = line[i][2:-2].replace('|', '')

file = open('data/index/' + indexname,'w')

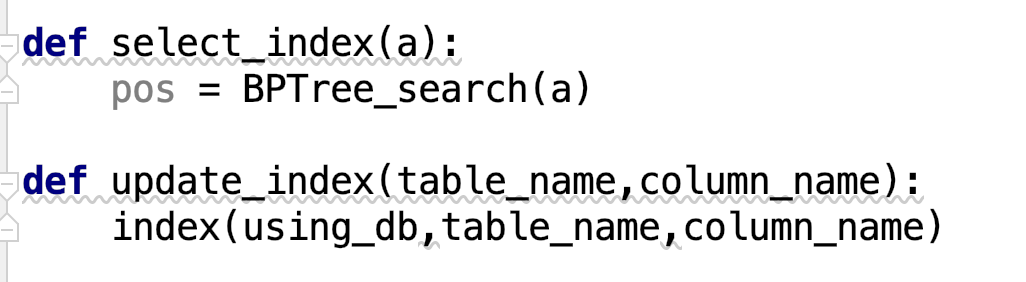
for i in range(len(line)):

file.writelines(line[i] + '\n')

file.close()

索引的存储，存储生成的b+树，每一行是B+树的一层。

存储数据类型是字典。



B+树的部分代码：

class BPTree(object):

…

def search(self, node, key):

i = bisect.bisect\_left(node.keys, key)

if i < len(node.keys) and key == node.keys[i]:

if node.is\_leaf():

return (node, i)

else:

return self.search(node.children[i + 1], key)

if node.is\_leaf():

return (None, None)

else:

# self.disk\_read(node.children[i])

return self.search(node.children[i], key)

…

def insert(self, key, value):

if len(self.root.keys) == self.\_maxkeys:

oldroot = self.root

self.root = BPNode()

self.root.children.append(oldroot)

self.split\_child(self.root, 0, oldroot)

self.insert\_nonfull(self.root, key, value)

else:

self.insert\_nonfull(self.root, key, value)

…

def levels(self):

leveldict = {}

for level, node in self.bft(self.root):

leveldict.setdefault(level, []).append(node)

return leveldict

def pprint(self, width=80):

leveldict = self.levels()

keys = leveldict.keys()

for k in keys:

print ' '.join(str(e) for e in leveldict[k]).center(width)

return leveldict

…

def min(self):

node = self.root

while node.children:

node = node.children[0]

return node.keys[0]

def max(self):

node = self.root

while node.children:

node = node.children[-1]

return node.keys[-1]

def bft(self, node, level=1):

"""Breadth first traversal."""

q = Queue.Queue()

level = level

q.put((level, node))

while not q.empty():

level, node = q.get()

yield (level, node)

for e in node.children:

q.put((level + 1, e))

…

def ceiling(self, node, key):

i = bisect.bisect(node.keys, key)

if i < len(node.keys) and key == node.keys[i]:

if node.is\_leaf():

return key

else:

return self.ceiling(node.children[i + 1], key)

if node.is\_leaf():

if i == len(node.keys):

kp = node.keys[-1]

if node.keys[-1] < key:

if len(node.next.keys) > 0:

return node.next.keys[0]

else:

return kp

return node.keys[i]

else:

return self.ceiling(node.children[i], key)

def split\_child(self, x, i, y):

z = BPNode()

z.keys = y.keys[self.degree:]

z.values = y.values[self.degree:]

if not y.is\_leaf():

z.children = y.children[self.degree:]

y.next = None

else:

z.keys.insert(0, y.keys[self.degree - 1])

z.values.insert(0, y.values[self.degree - 1])

z.next = y.next

y.next = z

x.children.insert(i + 1, z)

x.keys.insert(i, y.keys[self.degree - 1])

# x.values.insert(i, y.values[self.degree-1])

y.keys = y.keys[:self.degree - 1]

y.values = y.values[:self.degree - 1]

y.children = y.children[:self.degree]

# 3.16 help xxx模块

输入“help database”命令，输出所有数据表、视图和索引的信息，同时显示其对象类型；输入“help table 表名”命令，输出数据表中所有属性的详细信息；

输入“help view 视图名”命令，输出视图的定义语句；

输入“help index 索引名”命令，输出索引的详细信息；

elif operate == 'help':

if sql\_word[1] == 'database':

show\_db()

if sql\_word[1] == 'table':

usdbnm = using\_dbname

use\_db('table\_information')

tbname = sql\_word[2]

select('\*',usdbnm,{'table':tbname})

if sql\_word[1] == 'view':

view\_name = sql\_word[2]

use\_db('view')

select('sql','sql',{'viewnamw':view\_name})

if sql\_word[1] == 'index':

print "All Index:"

indexs = os.listdir('data/index/') # 第二种方法，从保存数据库信息的库中查询

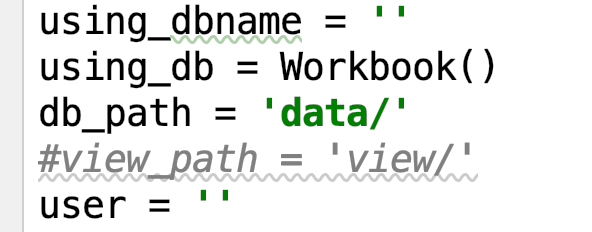
for index in indexs:

if '.DS' not in index:

print "[\*] " + index[:-5]

# 3.17 数据库选择和创建模块

定义来一些全局变量：



def use\_db(dbname):

global using\_dbname

global using\_db

if check\_permission(user, dbname, 'use'):

using\_dbname = dbname

using\_db = load\_workbook(db\_path+dbname+'.xlsx')

print("Database changed.")

def creat\_db(dbname):

dbpath = 'data/' + dbname + '.xlsx'

database = Workbook()

database.save(dbpath)

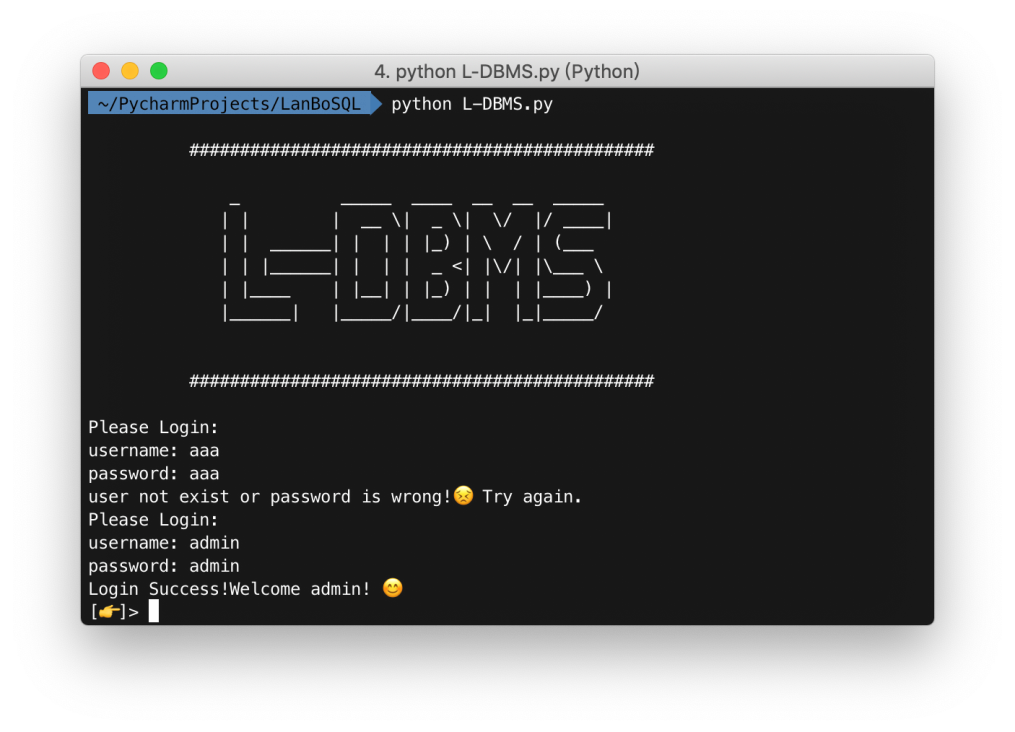
create\_tb\_in\_tbinfo(dbname)

print(u"数据库创建操作执行成功")

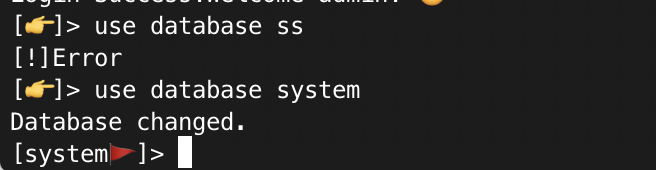
4.用户使用说明

> python L-DBMS.py 运行程序

进入用户登陆界面，输入用户名和密码登录。



选择数据库

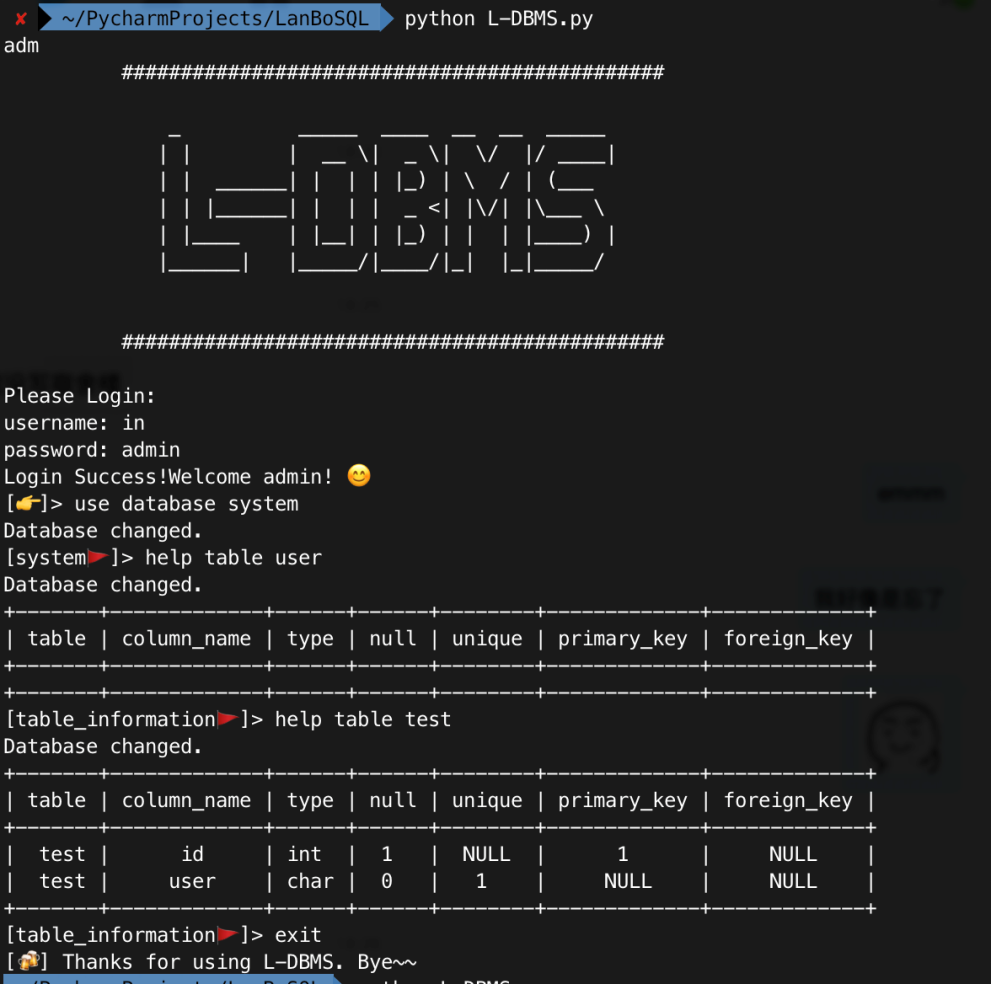


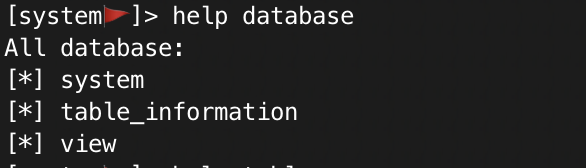
查询



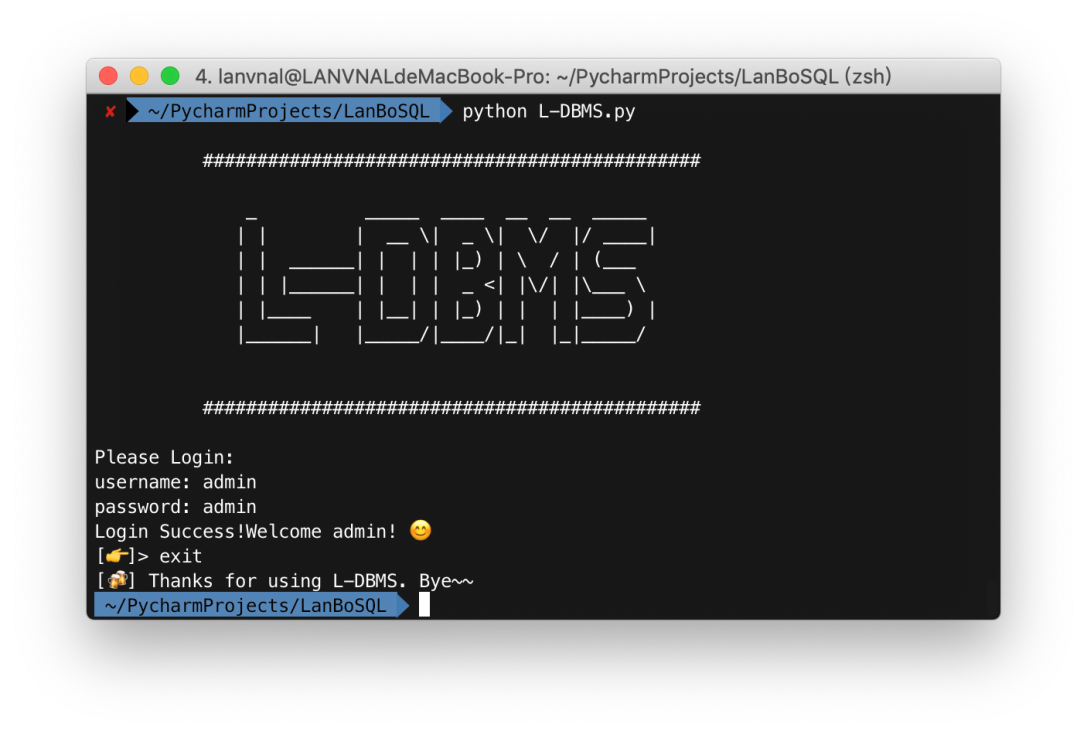


Help xxx





Exit or quit退出



5.课程设计感想

对sql语句的处理可以说是最麻烦的一部分了，在处理sql语句方面感觉做的还不够优雅，而且功能没做完善。在写代码过程中也是随着遇到的问题学习到了许多。而且认识到了再开始要规划好代码逻辑和数据结构，不然写到一半再改很麻烦。

功能并不完善，奈何时间有限，回去看自己的代码感觉并不优美，一些功能的实现方式甚至用很麻烦的方法解决，还需要努力啊。