**实 验 报 告**

|  |  |  |  |
| --- | --- | --- | --- |
| **课程名称：** | 数据库系统实验 | | |
| **学 院：** | 计算机科学与工程学院 | | |
| **专 业：** | 计算机科学与技术 | **班级：** | 2018级 2 班 |
| **姓 名：** | 付子豪 | **学号：** | 201801060807 |

2020**年 12 月** 27 **日**

**山东科技大学教务处制**

**实 验 报 告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **组 别** |  | **姓 名** | 付子豪 | **同组实验者** |  |
| **实验项目名称** | 数据，元数据，用户和权限 | | | **实验日期** | 11月 19 日 |
| **教 师 评 语** |  | | | | |
| **实验成绩：** | | | **指导教师（签名）：**  2020年 月 日 | | |
| **一．实验目的**  设计特定的数据结构，实现表、视图、索引、用户和权限的存储  **二．实验内容**  建立数据字典，存储数据库的元数据（至少包括表、视图、索引、 用户和权限）  **三．实验过程**  1.利用python中pandas库的DataFrame格式对元数据进行存储  2.表：df\_all.to\_csv(db\_path+table\_name+'.csv')  视图：db.to\_csv('Data/view.csv')  索引：df.to\_csv('Data/index/'+index\_name+'.csv')  用户与权限：db.to\_csv('Data/user.csv')  **四．实验结果**  表：  视图：    索引：  用户与权限：  **五．遇到的主要问题及解决的办法**  Python中可保存为csv文件，可方便存储和读取。用户文件可建立一个独立的csv文件，每一个元组为一个用户和其所包含的权限，每个操作下有用户所能操作的表名和列名。  **六．思考与感想**  视图是虚表，存储时不需要把表存下来，只需要把子查询语句存下来，用时在进行查询。  哈希索引存储索引字段的哈希值和哈希链表，可加快索引。用户和权限保证了安全性。 | | | | | |

**实 验 报 告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **组 别** |  | **姓 名** | 付子豪 | **同组实验者** |  |
| **实验项目名称** | 建表 | | | **实验日期** | 11月26 日 |
| **教 师 评 语** |  | | | | |
| **实验成绩：** | | | **指导教师（签名）：**  2020年 月 日 | | |
| **一．实验目的**  解析、执行CREATE TABLE语句，实现建表功能  **二．实验内容**  执行CREATE table语句，创建关系表。解析CREATE table语句，做词法、语法、语义分析，及安全性检查，并实现建表功能。建表时需要包含主码、外码、唯一性约束、非空约束等完整性约束的定义  **三．实验过程**   1. 执行Create table语句，先解析语句，之后用safe函数检查用户的权限，之后将建立的表名和数据字典传给create\_table()函数 2. Create\_table()函数将会创建新的数据帧，在对数据字典遍历后根据是否包含主码、外码、唯一性约束、非空约束等完整性约束创建新的数据帧，合并所有数据帧，并用表名存储下来   语句分析代码：    创建表代码：    **四．实验结果**      **五．遇到的主要问题及解决的办法**  语句解析时，利用正则化提取括号里的内容，表中建立五个特殊元组表示属性类型和属性是否为主码，是否非空，是否唯一，是否有check约束，用0和1表示是或否  **六．思考与感想**  建表要建立其属性的各个约束条件，为之后的操作做基础 | | | | | |

**实 验 报 告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **组 别** |  | **姓 名** | 付子豪 | **同组实验者** |  |
| **实验项目名称** | 建立视图和索引 | | | **实验日期** | 11月 27 日 |
| **教 师 评 语** |  | | | | |
| **实验成绩：** | | | **指导教师（签名）：**  2020年 月 日 | | |
| **一．实验目的**  解析、执行CREATE VIEW实现创建试图功能，解析、执行CREATE INDEX语句，实现创建索引的功能  **二．实验内容**  执行CREATE view语句，创建视图。解析CREATE view语句，做词法、语法、语义分析，及安全性检查，并实现创建视图功能。  执行CREATE index语句，创建索引。解析CREATE index语句，做词法、语法、语义分析，及安全性检查，并实现创建索引功能  **三．实验过程**  1.先建立一个view.csv文件，每当解析完create view语句之后，  将视图名和子查询语句传到creat\_view函数  2.creat\_view函数建立一个新的数据帧，包含视图名和子查询  之后打开view.csv文件，将新的数据帧加入到view.csv中，把新的view.csv存储下来  分析语句代码：    创建视图代码：    1.对创建索引语句进行解析，安全性检查之后，将索引名，表名和索引字段传到create\_index函数中  2.create\_index函数先打开表文件，之后建立有关索引字段的哈希索引，并将哈希值和链表指针作为数据帧存储在一个csv文件中，以索引名为文件名存储此文件  语句解析代码：    创建索引代码：    **四．实验结果**        **五．遇到的主要问题及解决的办法**  先建立一个view的csv文件用来存储建立的视图名和子查询，对视图语句的解析包括视图名和子查询语句，都要进行存储。哈希索引不同的字段可能会出现一样的哈希值，要进行判断，哈希链表存储每一个元组。  **六．思考与感想**  视图和索引都能加快数据库的运算速度，一个是虚表，一个有索引文件 | | | | | |

**实 验 报 告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **组 别** |  | **姓 名** | 付子豪 | **同组实验者** |  |
| **实验项目名称** | 查询 | | | **实验日期** | 12月3 日 |
| **教 师 评 语** |  | | | | |
| **实验成绩：** | | | **指导教师（签名）：**  2020年 月 日 | | |
| **一．实验目的**  解析、执行SELECT语句，实现查询功能  **二．实验内容**  执行Select语句，从表存储文件中查询数据，并输出结果。  解析语句，做词法、语法、语义分析，及安全性检查，并实现查询单表、连接、嵌套和集合查询功能。在SELECT语句中需要支持GROUP BY、HAVING和ORDER BY子句，支持聚集函数；Where语句支持and、or、between and、in、like等。  **三．实验过程**  1.解析select语句，进行安全性检查后，判断是否有where条件，再判断是否有集合查询的条件，再解析group by，having，order by，最后将解析结果传到select函数中  2.select函数中先对表名进行判断，判断是否为连接查询，再判断是否含有where语句，若无则直接判断所输出的属性，之后判断是否有集合查询条件，若有则进行查询并进行运算，更新其所要打印的数据帧，再检查是否为插入选择，若 是则返回数据帧而不打印，若否在判断是否有group by，having，order by，然后进行打印输出；对于有where的查询，判断是where类型，分为between，in，like，and和or。分别利用where查询到其所要的数据帧，在进行打印。  语句解析代码：  elif operate=='select':  #"select \* from sc where id=001 and name='xiaohong'"  table\_list=sql\_word[3].split(',')  if sql\_word[1]=='\*':  columns=['\*']  else:  columns=sql\_word[1].split(',')  if not safe(sql\_word[3],operate,columns):  return False  if 'where' not in sql\_word:#没有where的简单  if 'union' in sql\_word:  where\_index=sql\_word.index('union')  jihe\_sql=sql\_word[where\_index:]  jihe\_sql=" ".join(jihe\_sql)  select(columns,table\_list,jihe\_sql=jihe\_sql)  elif 'intersect' in sql\_word:  where\_index=sql\_word.index('intersect')    jihe\_sql=sql\_word[where\_index:]  jihe\_sql=" ".join(jihe\_sql)  select(columns,table\_list,jihe\_sql=jihe\_sql)  elif 'except' in sql\_word:  where\_index=sql\_word.index('except')    jihe\_sql=sql\_word[where\_index:]  jihe\_sql=" ".join(jihe\_sql)  select(columns,table\_list,jihe\_sql=jihe\_sql)  else:  if 'order' in sql\_word:#有没有order by  where\_index=sql\_word.index('order')  order\_sql=sql\_word[where\_index:]  group\_sql=[]  having\_sql=[]  elif 'group' in sql\_word:#有没有分组  where\_index=sql\_word.index('group')  if 'having' in sql\_word:#有没有having  group\_index=sql\_word.index('having')  order\_sql=[]  group\_sql=sql\_word[where\_index:group\_index]  having\_sql=sql\_word[group\_index:]  else:  order\_sql=[]  group\_sql=sql\_word[where\_index:]  having\_sql=[]  else:  order\_sql=[]  group\_sql=[]  having\_sql=[]  select(columns,table\_list,'',tag,order\_sql,group\_sql,having\_sql,jihe\_sql='')  else:#有where,分割where\_sql  if 'union' in sql\_word:  where\_index=sql\_word.index('union')  where\_sql=sql\_word[5:where\_index]  jihe\_sql=sql\_word[where\_index:]  jihe\_sql=" ".join(jihe\_sql)  select(columns,table\_list,where\_sql,tag,jihe\_sql=jihe\_sql)  elif 'intersect' in sql\_word:  where\_index=sql\_word.index('intersect')  where\_sql=sql\_word[5:where\_index]  jihe\_sql=sql\_word[where\_index:]  jihe\_sql=" ".join(jihe\_sql)  select(columns,table\_list,where\_sql,tag,jihe\_sql=jihe\_sql)  elif 'except' in sql\_word:  where\_index=sql\_word.index('except')  where\_sql=sql\_word[5:where\_index]  jihe\_sql=sql\_word[where\_index:]  jihe\_sql=" ".join(jihe\_sql)  select(columns,table\_list,where\_sql,tag,jihe\_sql=jihe\_sql)  else:  if 'order' in sql\_word:  where\_index=sql\_word.index('order')  where\_sql=sql\_word[5:where\_index]  order\_sql=sql\_word[where\_index:]  elif 'group' in sql\_word:  where\_index=sql\_word.index('group')  if 'having' in sql\_word:  group\_index=sql\_word.index('having')  where\_sql=sql\_word[5:where\_index]  group\_sql=sql\_word[where\_index:group\_index]  having\_sql=sql\_word[group\_index:]  else:  where\_sql=sql\_word[5:where\_index]  group\_sql=sql\_word[where\_index:]  having\_sql=[]  else:  group\_sql=[]  having\_sql=[]  where\_sql=sql\_word[5:]  order\_sql=[]  select(columns,table\_list,where\_sql,tag,order\_sql,group\_sql,having\_sql,jihe\_sql='')  Select函数代码：  def select(columns,table\_list,where\_sql='',tag='',order\_sql='',group\_sql='',having\_sql='',jihe\_sql=''):  Betwenn：  if 'between' in where\_sql:  column,mins,maxs=where(where\_sql)  db=db.loc['tuple']  df=db[(db[column]<maxs)&(db[column]>mins)]#选择好数据帧  #进行返回或打印，重复  In：（包含嵌套查询）  elif 'in' in where\_sql:  db=db.loc['tuple']  column,values=where(where\_sql)  if 'select' in values:#嵌套查询之一in  columns\_1=values[1].split(',')  table\_list\_1=values[3].split(',')  where\_sql\_1=values[5:]  df= select(columns\_1,table\_list\_1,where\_sql\_1,tag='insert')  values=df[column].tolist()    df=pd.DataFrame()  for i in values:  df1=db[db[column]==i]  df=pd.concat([df,df1],axis=0)  Like：elif 'like' in where\_sql:  db=db.loc['tuple']  column,list\_f,list\_l=where(where\_sql)  df=pd.DataFrame()  list1=list(db[column])  for i in range(len(list1)):  list2\_f=list(list1[i])  a=list1[i]  b=a[::-1]  list2\_l=list(b)  flag=0  flag1=0  if '%' in list\_l:  if len(list\_f)!=len(list2\_f):  flag=1  for j in range(len(list\_f)):  if (list\_f[j]!=list2\_f[j])&(list\_f[j]!='\_'):  flag=1  break  if flag==0:  df1=db[db[column]==list1[i]]  df=pd.concat([df,df1],axis=0)  else:  for j in range(len(list\_f)):  if (list\_f[j]!=list2\_f[j])&(list\_f[j]!='\_'):  flag=1  break  if flag==0:  if len(list\_l)>0:  for k in range(len(list\_l)):  if (list\_l[k]!=list2\_l[k])&(list\_l[k]!='\_'):  flag1=1  break  if flag1==0:  df1=db[db[column]==list1[i]]  df=pd.concat([df,df1],axis=0)  else:  df1=db[db[column]==list1[i]]  df=pd.concat([df,df1],axis=0)  打印或输出：  if "\*" not in columns:  if len(jihe\_sql)!=0:  df\_jihe=jihe\_f(jihe\_sql)  if jihe\_sql[0]=='union':  df=pd.merge(df,df\_jihe, how='outer')  elif jihe\_sql[0]=='intersect':  df=pd.merge(df,df\_jihe)  elif jihe\_sql[0]=='except':  df = df.append(df\_jihe)  df = df.append(df\_jihe)  df = df.drop\_duplicates(keep=False)  if tag=='insert':  if len(order\_sql)!=0:  column=order\_sql[2]  if order\_sql[3]=='desc':  df.sort\_values(by=column,ascending=False,inplace=True)  elif order\_sql[3]=='asc':  df.sort\_values(by=column,inplace=True)  columns\_list=gather(columns,df)  return df[columns\_list]  else:  if len(order\_sql)!=0:  column=order\_sql[2]  if order\_sql[3]=='desc':  df.sort\_values(by=column,ascending=False,inplace=True)  elif order\_sql[3]=='asc':  df.sort\_values(by=column,inplace=True)  columns\_list=gather(columns,df)  if len(group\_sql)!=0:  column=group\_sql[2]  grouped = df.groupby(column)  for name,group in grouped:  if len(having\_sql)!=0:  limit=having\_sql[1]  dict\_1={}  if '>' in limit:  cols=limit.split('>')  dict\_1[cols[0]]=cols[1]  symbol='>'  elif '<' in limit:  cols=limit.split('<')  dict\_1[cols[0]]=cols[1]  symbol='<'  elif '=' in limit:  cols=limit.split('=')  dict\_1[cols[0]]=cols[1]  symbol='='  if '>' in limit:  if name>dict\_1[column]:  print(name)  display(group.head())  elif '<' in limit:  if name<dict\_1[column]:  print(name)  display(group.head())  elif '=' in limit:  if name<dict\_1[column]:  print(name)  display(group.head())  else:  print(name)  display(group.head())  else:  display(df[columns\_list])  else:  if len(jihe\_sql)!=0:  df\_jihe=jihe\_f(jihe\_sql)  if jihe\_sql[0]=='union':  df=pd.merge(df,df\_jihe, how='outer')  elif jihe\_sql[0]=='intersect':  df=pd.merge(df,df\_jihe)  elif jihe\_sql[0]=='except':  df = df.append(df\_jihe)  df = df.append(df\_jihe)  df = df.drop\_duplicates(keep=False)  if tag=='insert':  if len(order\_sql)!=0:  column=order\_sql[2]  if order\_sql[3]=='desc':  df.sort\_values(by=column,ascending=False,inplace=True)  elif order\_sql[3]=='asc':  df.sort\_values(by=column,inplace=True)  columns\_list=gather(columns,df)  return df  else:  if len(order\_sql)!=0:  column=order\_sql[2]  if order\_sql[3]=='desc':  df.sort\_values(by=column,ascending=False,inplace=True)  elif order\_sql[3]=='asc':  df.sort\_values(by=column,inplace=True)  columns\_list=gather(columns,df)  if len(group\_sql)!=0:  column=group\_sql[2]  grouped = df.groupby(column)  for name,group in grouped:  if len(having\_sql)!=0:  limit=having\_sql[1]  dict\_1={}  if '>' in limit:  cols=limit.split('>')  dict\_1[cols[0]]=cols[1]  symbol='>'  elif '<' in limit:  cols=limit.split('<')  dict\_1[cols[0]]=cols[1]  symbol='<'  elif '=' in limit:  cols=limit.split('=')  dict\_1[cols[0]]=cols[1]  symbol='='  if '>' in limit:  if name>dict\_1[column]:  print(name)  display(group.head())  elif '<' in limit:  if name<dict\_1[column]:  print(name)  display(group.head())  elif '=' in limit:  if name<dict\_1[column]:  print(name)  display(group.head())  else:  print(name)  display(group.head())  else:  display(df)  And和or：  else:#or或者and时  dfs=[]  link,limit,symbol=where(where\_sql)  for k,i in zip(limit,symbol):  if i=='=':  for i in k:  if '.' in i:  index=i.index('.')  j=i[(index+1):]  if '.' in k[i]:  df1=db  else:  df1=db[db[j]==k[i]]  else:  df1=db[db[i]==k[i]]  dfs.append(df1)  elif i=='<':  for i in k:  if '.' in i:  index=i.index('.')  j=i[(index+1):]  if '.' in k[i]:  df1=db  else:  df1=db[db[j]<k[i]]  else:  df1=db[db[i]<k[i]]  dfs.append(df1)  elif i=='>':  for i in k:  if '.' in i:  index=i.index('.')  j=i[(index+1):]  if '.' in k[i]:  df1=db  else:  df1=db[db[j]>k[i]]  else:  df1=db[db[i]>k[i]]  dfs.append(df1)  for i in range(len(dfs)):  if i<len(dfs)-1:  if link[i]=='and':  dfs[i+1]=pd.merge(dfs[i],dfs[i+1])  elif link[i]=='or':  dfs[i+1]=pd.merge(dfs[i],dfs[i+1],how='outer')  df=dfs[len(dfs)-1]  Where语句：  def where(where\_sql):  if 'between' in where\_sql:  column=where\_sql[0]  mins=where\_sql[2]  maxs=where\_sql[4]  return column,mins,maxs  elif ('in' in where\_sql)|(' = ' in where\_sql):  column=where\_sql[0]  if 'select' in where\_sql[2]:  select\_sql=' '.join(where\_sql)  values = re.findall('\((.\*)\)', select\_sql)[0].split(' ')  print(values)  else:  values = re.findall('\((.\*)\)', where\_sql[2])[0].split(',')  return column,values  elif 'like' in where\_sql:  list\_f=[]  list\_l=[]  column=where\_sql[0]  list0=list(where\_sql[2])  if '%' not in where\_sql[2]:  list\_f=list(where\_sql[2])  list\_l=['%']  else:  if list0[0]=='%':  list\_f=[]  \_list=where\_sql[2].split('%')  a=\_list[1]  b=a[::-1]  list\_l=list(b)  elif list0[len(list0)-1]=='%':  list\_l=[]  \_list=where\_sql[2].split('%')  a=\_list[0]  list\_f=list(a)  else:  list0=where\_sql[2].split('%')  list\_f=list(list0[0])  a=list0[1]  b=a[::-1]  list\_l=list(b)  return column,list\_f,list\_l  else:  link=[]  condi=[]  symbol=[]  limit=[]  for i in range(len(where\_sql)):  if where\_sql[i]=='and' or where\_sql[i]=='or':  link.append(where\_sql[i])  else:  condi.append(where\_sql[i])  for i in range(len(condi)):  if '=' in condi[i]:  a=condi[i].split('=')  limit\_dict={a[0]:a[1]}  limit.append(limit\_dict)  symbol.append('=')  elif '>' in condi[i]:  a=condi[i].split('>')  limit\_dict={a[0]:a[1]}  limit.append(limit\_dict)  symbol.append('>')  elif '<' in condi[i]:  a=condi[i].split('<')  limit\_dict={a[0]:a[1]}  limit.append(limit\_dict)  symbol.append('<')  return link,limit,symbol  Jihe\_f函数：  def jihe\_f(jihe\_sql):  sql\_word=jihe\_sql.split(' ')  sql\_word=sql\_word[1:]  table\_list=sql\_word[3].split(',')  if sql\_word[1]=='\*':  columns=['\*']  else:  columns=sql\_word[1].split(',')  if 'where' not in sql\_word:#没有where的简单  df=select(columns,table\_name,'',tag='insert')  return df  else:#有where  where\_sql=sql\_word[5:]    df=select(columns,table\_list,where\_sql,tag='insert')  return df  聚集函数：  def gather(columns,db):#聚集函数，影响select的输入，返回应该输入的属性列表  columns\_list=[]  for i in range(len(columns)):  if '(' in columns[i]:  if 'count' in columns[i]:  column= re.findall('\((.\*)\)', columns[i])[0].split(',')#属性  print(db.shape[0])  elif 'sum' in columns[i]:  column= re.findall('\((.\*)\)', columns[i])[0].split(',')#属性  column=column[0]  values=list(db[column])  All=0  for i in values:  i=float(i)  All+=i  print(All)  elif 'avg' in columns[i]:  column= re.findall('\((.\*)\)', columns[i])[0].split(',')#属性  column=column[0]  values=list(db[column])  l=len(values)  All=0  for i in values:  i=float(i)  All+=i  print(All/l)  elif 'max' in columns[i]:  column= re.findall('\((.\*)\)', columns[i])[0].split(',')#属性  column=column[0]  values=list(db[column])  print(max(values))  elif 'min' in columns[i]:  column= re.findall('\((.\*)\)', columns[i])[0].split(',')#属性  column=column[0]  values=list(db[column])  print(min(values))  else:  columns\_list.append(columns[i])  return columns\_list  **四．实验结果**  查询的表信息：    And和or查询：      Between查询：    In查询：    Like查询：        集合查询：    嵌套查询：    连接查询：    Group by和having语句查询：      Order by语句查询：      聚集函数：        **五．遇到的主要问题及解决的办法**  查询语句分类多种，代码冗杂，每一种分类都应该囊括，耗费了大量时间和精力，问题也层出不穷，我先在程序之外单独实现了各个功能，再加到程序主体上，再进行特例调试，一步步调试出结果  **六．思考与感想**  经过查询的编程，让我对sql语句对查询的各个情况有了了解，对书本上的知识有了实践，收获很深，也获得了得出正确结果时的喜悦。 | | | | | |

**实 验 报 告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **组 别** |  | **姓 名** | 付子豪 | **同组实验者** |  |
| **实验项目名称** | 插入，删除，更新 | | | **实验日期** | 12月10 日 |
| **教 师 评 语** |  | | | | |
| **实验成绩：** | | | **指导教师（签名）：**  2020年 月 日 | | |
| **一．实验目的**  *解析、执行SELECT语句，实现查询功能。解析、执行INSERT、UPDATE和DELETE语句，实现表中数据的增删改功能，检查完整性约束*  **二．实验内容**  执行INSERT、DELETE和UPDATE语句，更新数据表的内容。  解析语句，做词法、语法、语义分析，做完整性及安全性检查，并实现更新功能。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。  **三．实验过程**  插入：1.对插入语句进行解析，安全性检查后，分为单个原则查询和子查询插入，将表名和数字字典以及子查询传入insert  2.insert函数中对于元组插入，只需创建一个关于数据字典的数据帧插入到原表中；对于元组集合子查询，要先获得select语句的数据原则，再插入  删除：1.对删除语句进行解析，安全性检查后，分为带有where语句和全表删除，分别把表名和where语句传到delete函数中  2.delete函数进行where判断，有则进行指定删除，无则进行全表删除，之后进行非空检查，无误后保存下来  更新：1.对更新语句进行解析，安全性检查后，分为全表更新和where更新，将表名和更新的数据字典和where语句传到update函数中  2. update函数对所选的数据元组进行更新，  实验代码：  插入：    删除：    **四．实验**  更新：      **四.实验结果**  插入：      删除：    删除前表元组：    删除后表元组：    更新：    更新前：    更新后：    **五．遇到的主要问题及解决的办法**  对于插入，分为单个原则插入和元组集合的插入，对于元组集合的插入，要先利用之后的select语句得到要插入的元组，再进行插入。对于删除和更新，都要where语句，先利用where函数解析where之后的条件，再进行删除和更新操作。Where所包含的查询在之后select中逐一体现。  **六．思考与感想**  插入，更新和删除是对数据表的基本操作，其中也有复杂的情况，比如子查询插入和带有where的更新和删除，都需要先得到所要进行操作的元组集合，所以select语句分析和实现格外关键。 | | | | | |

**实 验 报 告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **组 别** |  | **姓 名** | 付子豪 | **同组实验者** |  |  |
| **实验项目名称** | 权限 | | | **实验日期** | 12月 17 日 |  |
| **教 师 评 语** |  | | | | |  |
| **实验成绩：** | | | **指导教师（签名）：**  2020年 月 日 | | |  |
| **一．实验目的**  解析、执行GRANT和REVOKE语句，实现数据库用户权限的授予和回收。   1. **实验内容**   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  **三．实验过程**  1.先对grant和revoke进行语句解析，将所授权的操作，表名和用户名传到各自函数中  2.grant函数先检查用户是否有grant功能，在将对应的表名和列名插入到对应操作下  Revoke函数先检查用户是否有revoke功能，在将对应的表名删除  Grant：  def grant(columns,table\_name,username):  db=pd.read\_csv('Data/user.csv',header=0,index\_col=0)  if 'all' in db[db['user']==user]['grant'].values:  if username in db['user'].values:    for i in range(len(columns)):  if columns[i]=='all privileges':  db.replace(db[db['user']==username],'all',inplace=True)  elif columns[i]=='create':  db[columns[i]].replace(db[db['user']==username][columns[i]],'all'  ,inplace=True)  else:  if "(" in columns[i]:  index=columns[i].index('(')  index\_1=columns[i].index(')')  privileges=columns[i][:index]  a=columns[i][index:index\_1+1]  else:  privileges=columns[i]  a=''  db[privileges].replace(db[db['user']==username  ][privileges],db[db['user']==username][privileges]+','+table\_name+a,inplace=True)  else:  print("没有该用户，请先建立用户！")  else:  print("用户权限不够！")  db.to\_csv('Data/user.csv')  Revoke：  def revoke(columns,table\_name,username):  db=pd.read\_csv('Data/user.csv',header=0,index\_col=0)  if 'all' in db[db['user']==user]['revoke'].values:  if username in db['user'].values:  for i in range(len(columns)):  a=str(db[db['user']==username][columns[i]])  cols=a.split(' ')  cols  b=cols[4]  b=b.replace('\nName:','')  if table\_name in b:  b=b.replace(','+table\_name,'')  db[columns[i]].replace(db[db['user']==user  name][columns[i]],b,inplace=True)  db.to\_csv('Data/user.csv')  else:  print("该用户没有该权限")  else:  print("没有该用户，请先建立用户！")  else:  print("用户权限不够")  **四．实验结果**            **五．遇到的主要问题及解决的办法**  语句分析比较明确，权限名称带列名时要特殊处理，把列名带括号加到表名后面，安全性检查  可对列名分析。  **六．思考与感想**  安全性分析要对user表中的用户进行权限查找，只有找到才可以进行相关操作 | | | | | |  |

**实 验 报 告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **组 别** |  | **姓 名** | 付子豪 | **同组实验者** |  |
| **实验项目名称** | 帮助信息 | | | **实验日期** | 12月18 日 |
| **教 师 评 语** |  | | | | |
| **实验成绩：** | | | **指导教师（签名）：**  2020年 月 日 | | |
| **一．实验目的**  *实现“HELP DATABASE”、“HELP TABLE 表名”、“HELP VIEW 视图名”、“HELP INDEX 索引名” 等命令。*  **二．实验内容**  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。  **三．实验过程**  1.对help语句进行分析，将help语句传到help函数中  2.help函数对语句进行解析，并作出相应处理  代码：  def help(command):  command\_word=command.split(' ')  if command\_word[1]=='database':  print('数据表——————————————')  for table\_name in table\_list:  print(table\_name+':')  db=pd.read\_csv('Data/'+table\_name+'.csv',index\_col=0)  df=db.loc['type']  print(df)  print('索引———————————————')  for i in range(len(index\_list)):  print('索引名：'+index\_list[i][0])  print('索引表名：'+index\_list[i][1])  print('索引字段：'+str(index\_list[i][2]))  print('视图———————————————')  db=pd.read\_csv('Data/'+'view'+'.csv',index\_col=0)  print(db)  elif command\_word[1]=='table':  table\_name=command\_word[2]  print('数据表——————————————')  print(table\_name+':')  db=pd.read\_csv('Data/'+table\_name+'.csv',index\_col=0)  df=db.loc[['type','primary key','unique','not null','foreign key','check']]  print(df)  elif command\_word[1]=='view':  view\_name=command\_word[2]  db=pd.read\_csv('Data/'+'view'+'.csv',index\_col=0)  print(db[db['view\_name']==view\_name])  elif command\_word[1]=='index':  index\_name=command\_word[2]  for i in range(len(index\_list)):  if index\_name in index\_list[i]:  print('索引名：'+index\_list[i][0])  print('索引表名：'+index\_list[i][1])  print('索引字段：'+str(index\_list[i][2]))  **四．实验结果**    **五．遇到的主要问题及解决的办法**  先建立全局列表存储每一个表名，索引名，然后根据命令进行输出  **六．思考与感想**  Help函数输出有关表，视图和索引的信息，能更清晰的了解数据库结构 | | | | | |

**实 验 报 告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **组 别** |  | **姓 名** | 付子豪 | **同组实验者** |  |
| **实验项目名称** | 原型系统 | | | **实验日期** | 12月 24 日 |
| **教 师 评 语** |  | | | | |
| **实验成绩：** | | | **指导教师（签名）：**  2020年 月 日 | | |
| **一．实验目的**  *实现DBMS原型系统，完成实验报告*  **二．实验配置环境**  实验环境为Python 3.6.5 |Anaconda，利用python语句完成py文件  **三．实验过程**  1.py文件中有主运行函数run函数，run函数是整个程序的开始与窗口  2.run函数在获得指令后分析指令，判断指令是不是sql语句，是则传到query函数中进行进一步解析  **四．实验结构**  run函数模块  Query模块  Grant/Revoke模块  Update模块  用户登录模块  Insert模块  Help模块  Delete模块  Select模块  Create模块  索引建立和更新模块  **登录代码：**  **def login():**  **global user**  **print ("Please Login:")**  **username = input("username: ")**  **df=pd.read\_csv('Data/user.csv',index\_col=0,header=0)**  **if username in df['user'].values:**  **print ("Login Success!Welcome {}! ".format(username))**  **user=username**  **else:**  **a=[[username,username,username,username,username,username,username,username]]**  **db=pd.DataFrame**  **(a,columns={'user':'','create':'','select':'','update':'','delete':'','insert':'','grant':'','revoke':''},**  **index=['tuple'])**  **df=pd.concat([df,db],axis=0)**  **df.to\_csv('Data/user.csv')**  **print ("Login Success!Welcome {}! ".format(username))**  **user = username**  **运行代码：**  **def run():**    **welcome()**  **login()**  **i=1**  **while i:**  **command = get\_command()**  **#print command**  **if command == 'quit' or command == 'exit':**  **print(" Thanks for using L-DBMS. Bye~~")**  **i=0**  **elif 'help'in command:**  **help(command)**  **elif command=='exit user':**  **user=''**  **login()**  **else:**  **query(command)**  **Check约束实现：**  **def check(df):**  **df=pd.read\_csv('Data/sc.csv',index\_col=0)**  **a=list(df.columns)**  **df**  **for i in a:**  **if df.loc['check'][i]!='无':**  **check=df.loc['check'][i]**  **limit= re.findall('\((.\*)\)', check)[0].split(' ')#属性**  **print(limit)**  **if '>' in limit[0]:**  **col=limit[0].split('>')**  **db=df.loc['tuple']**  **values=list(db[i])**  **for i in range(len(values)):**  **if not values[i]>col[1]:**  **print('元组不满足check约束')**  **return False**  **return True**  **主码唯一约束：**  **def check\_Constraint(df):**  **columns=list(df.columns)**  **for k in columns:**  **if(df.loc['primary key'][k]=='1'):**  **try:**  **db=df.loc['tuple']**  **countdata=db[k].value\_counts().reset\_index()**  **a=list(countdata[k])**  **for i in range(len(a)):**  **if a[i]!=1:**  **print('不允许重复值')**  **return False**  **except:**  **continue**  **elif df.loc['unique'][k]=='1':**  **try:**  **db=df.loc['tuple']**  **countdata=db[k].value\_counts().reset\_index()**  **a=list(countdata[k])**  **for i in range(len(a)):**  **if a[i]!=1:**  **print('不允许重复值')**  **return False**  **except:**  **continue**  **return True**  **非空约束：**  **def check\_null(db):**  **for k in db.columns:**  **if(db.loc['primary key'][k]=='1'):**  **if db[k].isna().sum()>0:**  **print(k+"属性不允许有空值")**  **return False**  **elif db.loc['not null'][k]=='1':**  **if db[k].isna().sum()>0:**  **print(k+"属性不允许有空值")**  **return False**  **return True**  **安全性检查：**  **def safe(table\_name,root\_name,columns=''):**  **db=pd.read\_csv('data/user.csv',index\_col=0)**  **column\_list=[]**  **if len(columns)!=0:**  **try:**  **for k in columns:**  **column\_list.append(k)**  **except:**  **column\_list=columns**  **if 'all' in db[db['user']==user][root\_name].values:**  **return True**  **elif table\_name in str(db[db['user']==user][root\_name]):**  **for i in column\_list:**  **if i not in str(db[db['user']==user][root\_name]):**  **print('没有该列的权限！')**  **return False**  **return True**  **else :**  **print('没有该权限！')**  **return False**  **五．反思**  通过python实现一个简单的数据库，熟悉了pandas库的各种命令，也对书本上的知识有了实践，对之后的复习和考试有了一定的促进作用，实验中遇到了各种各样的问题，要有足够的耐心解决各种问题，要充分利用书本和网络上的资源，解决自己在编程中遇到的问题。 | | | | | |