**数据库系统project报告**

|  |  |
| --- | --- |
| 数据库系统project任务书 | |
| 名称 | 数据库SQL执行器设计与模拟实现 |
| 类型 | □验证性 □设计性 综合性 |
| 内容 | 针对关系型数据库SQL语句执行，进行需求分析，并设计一个简单的数据库系统SQL执行器模块，根据设计模拟实现SQL基本操作等，主要实现：数据库创建、表格创建、数据添加、删除、更新、查询等操作。  模拟实现采用：python或者Java实现具体功能，设计中若有数据的存储可以使用文本文件或者excel文件，SQL语句执行采用函数实现。 |
| 要求 | （1）设计方案要合理；  （2）能基于该SQL执行器模块完成SQL的模拟实现；  （3）设计方案有一定的效率分析。 |
| 任务时间 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 小组成员 | | | |
| 项目评分表 | | | |
| 序号 | 评分项 | 分值 | 得分 |
| 1 | 需求分析 | 3分 |  |
| 2 | 综合设计与实现 | 4分 |  |
| 3 | 团队协作 | 3分 |  |
| 项目总得分： | | | |

课程项目评分标准（总分10分）

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 评分项目 | 完成情况 | 得分 |
| 1 | 需求分析 | 分析合理 | 3分 |
| 分析较合理 | 2分 |
| 分析不合理 | 1分 |
| 分析完全错误 | 0分 |
| 2 | 综合设计与实现 | 设计完整，设计合理，工具使用熟练 | 4分 |
| 设计较完整，设计合理，工具使用较熟练 | 3分 |
| 设计较完整，设计较合理，工具使用较熟练 | 2分 |
| 设计较完整，设计不合理，工具使用不熟练 | 1分 |
| 抄袭、被抄袭 | 0分 |
| 3 | 团队协作 | 有团队，分工合理，密切协作 | 3分 |
| 有团队，分工合理，有一定协作 | 2分 |
| 有团队，分工不合理，无协作 | 1分 |
| 无团队，无协作 | 0分 |

1. 需求分析
2. 实现方式

实验要求采用Python或Java实现具体功能,设计中数据的存储使用文本(.txt)文件或者Excel(.csv)文件。出于可读性考虑,本次实验采用Excel文件对数据进行存储。由于Python在Excel文件的读写方面功能更为完善,且文件读写并非本次实验的重点,因此经过比较,小组成员决定采用Python来实现所需功能。

1. 语法分析

本次实验需要设计一个简单的数据库系统SQL执行器模块,并模拟实现以下SQL基本操作:

1)数据库创建(Create Database)

2)表格创建(Create Table)

3)数据添加(Insert)

4)数据删除(Drop & Delete)

5)数据更新(Update)

6)数据查询(Select)

由于SQL语句主要通过控制台(Console)以字符串(String)方式读入,因此首先需要对指令进行预处理和关键词(Keyword)检索,并通过关键词检索的结果将指令进行初步分解,以确定所要执行的SQL操作(以Python函数方式实现)和操作的数据对象(函数的输入)。

在接收到上述输入后,函数将根据操作类型(Create,Drop等)和关键词(From,Where等)将SQL语句彻底分解,并执行数据的筛选和处理。

1. 数据筛选

在本次实验拟实现的SQL操作(Delete,Update,Select等)中,通过暴力方式(如遍历)进行数据筛选的时间复杂度和空间复杂度是难以接受且现实意义较弱的。同时,项目要求设计方案具有一定效率分析。因此,经过综合考量,小组成员选择采用实际数据库中常见的B+树(BPlustTree)结构作为索引(Index),以简化数据筛选的时空复杂度。

1. 数据处理

在本次实验中,主要的数据处理操作包括:

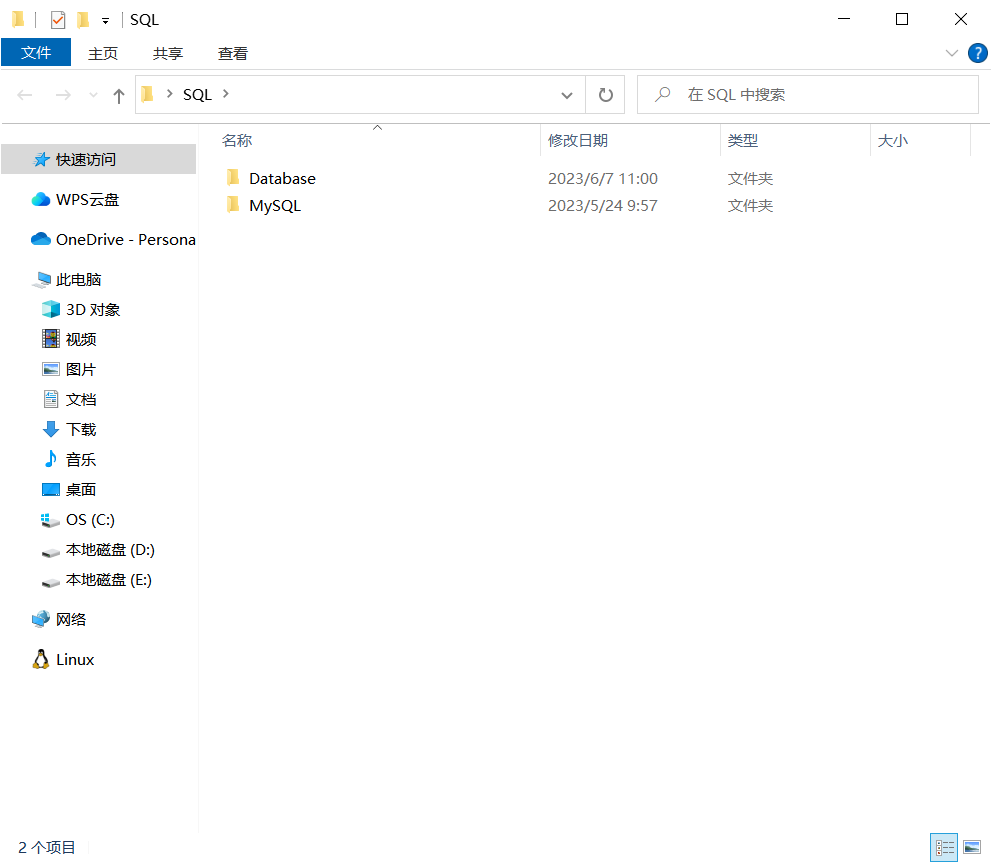
1)文件夹和Excel等文件的创建和删除

2)读取和重写Excel等文件中的数据

3)将程序运行的结果(关系模式,元组等)永久性保存在Excel等文件中

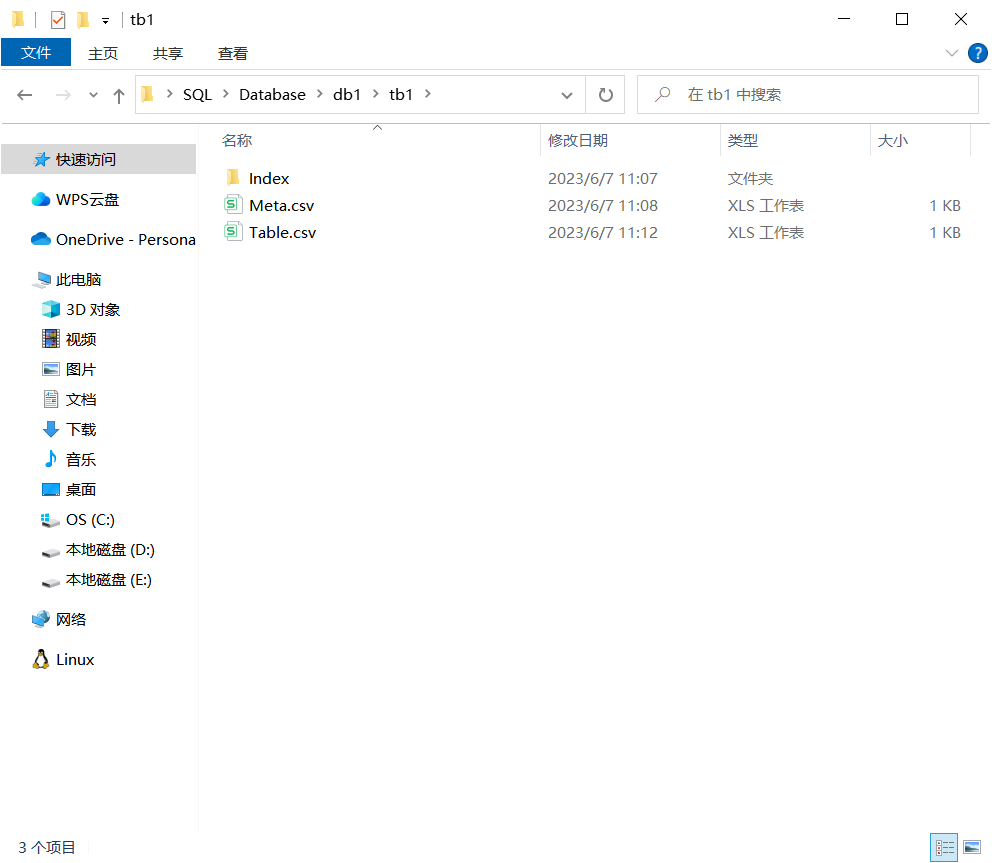
1. 综合设计与实现
2. 项目结构

项目共包含Database和MySQL两个目录,其中Database目录在程序执行时将依据存在性自动生成,是数据库对象的存储路径。



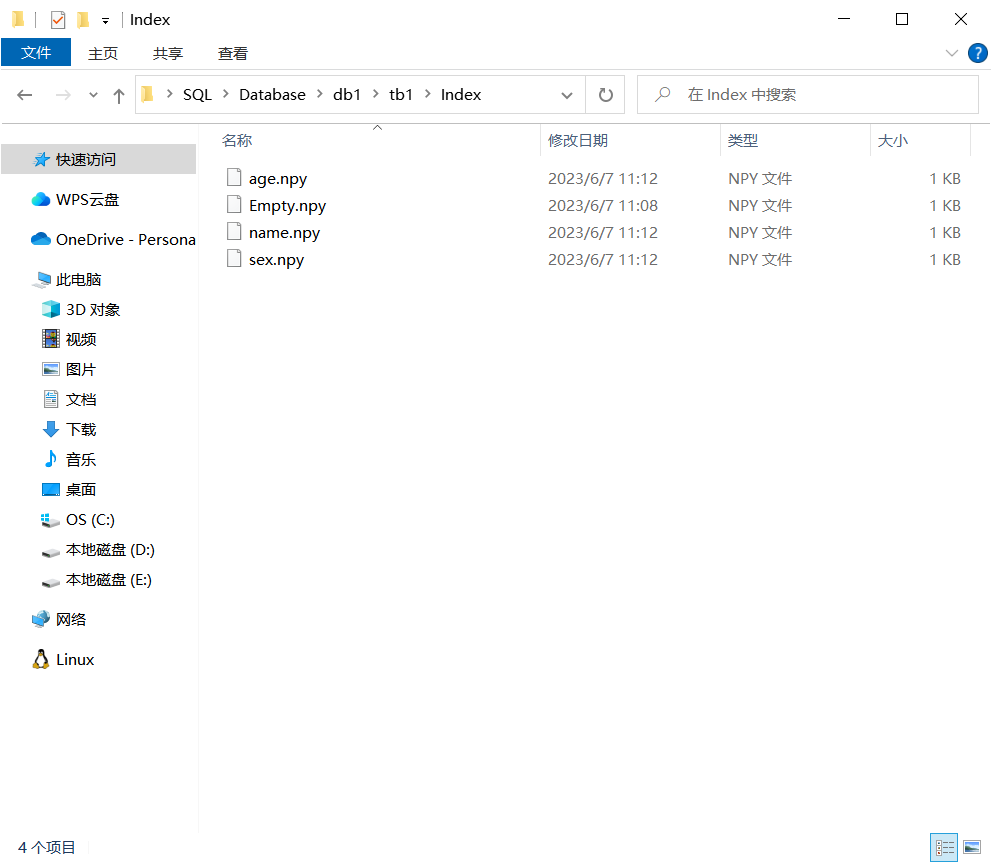
**图1.1 顶层结构**

图1.2所示的”SQL/Database/db1/tb1”路径表示的意思为数据库目录中,名为”db1”的数据库中,名为”tb1”的关系模式的相关文件,包括索引(Index)目录,Meta(元数据)文件和Table(关系模式)文件。



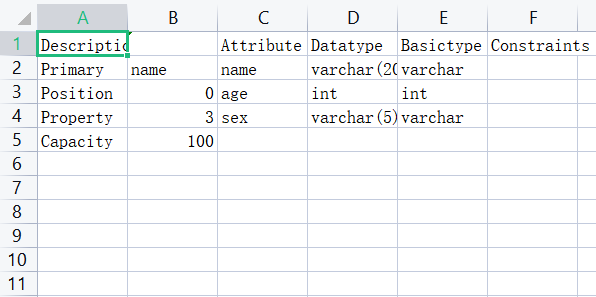
**图1.2 Database目录**

图1.3所示 Index目录下保存有在该关系模式的各个属性上建立的B+树索引,以.npy格式储存(Python的Numpy模块所提供的保留指定变量的生成文件)



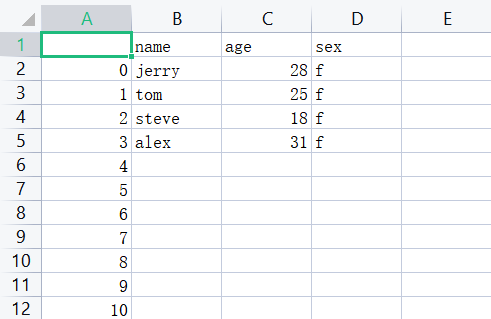
**图1.3 Index目录**

图1.4所示Meta.csv文件中保存的是对该关系模式的基本描述,包含主码(Primary Key),主码属性在所有属性中的位置(Position),属性数(Property),容量(Capacity)和各个属性的名称(Attribute),高层数据类型(Datatype),基本数据类型(Basictype)和属性上的约束(Constraints)



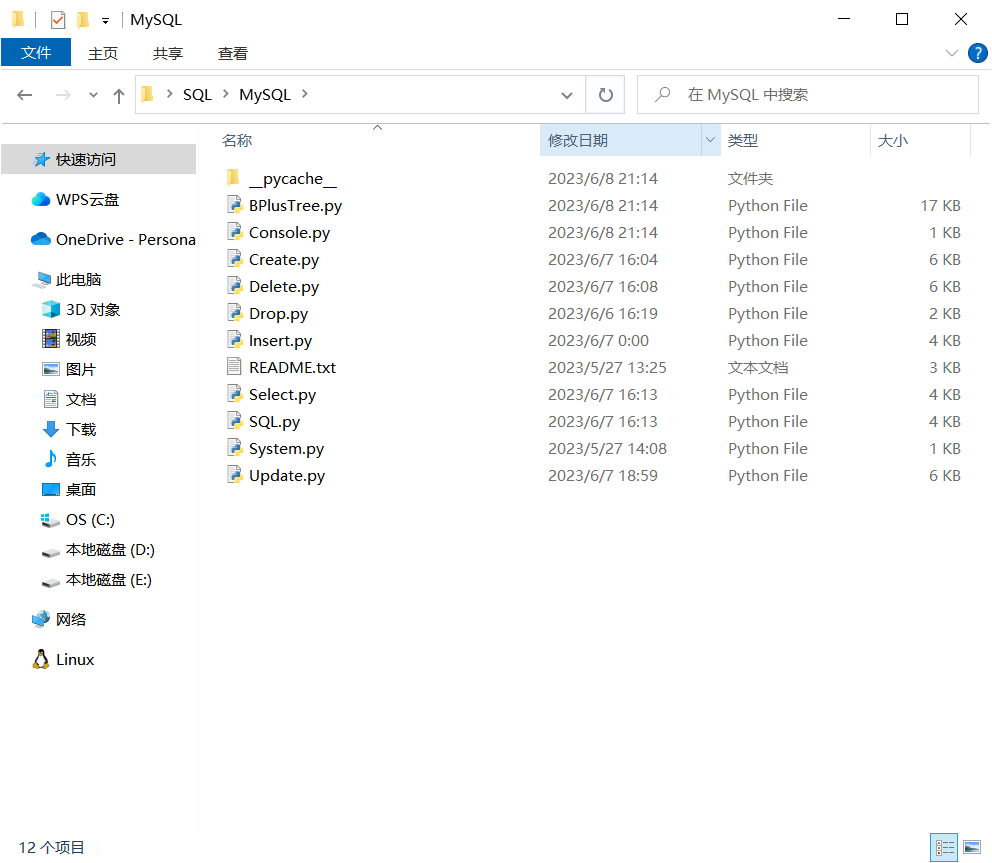
**图1.4 Meta文件**

图1.5所示Table.csv文件中保存的是该关系模式下存储的元组,用元组的行坐标模拟数据在硬件中的存储地址,最大地址由容量指定。当插入操作发生时,程序将读取Empty.npy文件中所记录的未存储有效数据(Not valid)的地址,并将新的元组存储在该地址所对应的行坐标位置,以模拟数据在硬件中的存储。当删除操作发生时,Empty.npy文件中将同步更新地址集,以简单模拟操作系统的行为



**图1.5 Table文件**

图1.6所示MySQL目录下保存的是实现上述功能的程序代码,其中Console文件为可执行文件,README文件为使用手册



**图1.6 MySQL目录**

1. 设计实现
2. System类

#该文件定义和实现了一些通用的系统函数

import os

def pardir(path):#返回指定目录的父目录,默认返回当前目录的父目录

    return os.path.abspath(os.path.join(path,".."))

def match(str):#检查字符串的括号是否合法

    s=[]

    for i in range(len(str)):

        if(str[i]=='('):

            s.append('(')

        elif(str[i]==')'):

            if(len(s)==0):

                return False

            s.pop()

    return (len(s)==0)

1. SQL类

#该文件定义了SQL指令的读取和部分处理操作

import os

import re

import Create,Drop,Insert,Delete,Update,Select

from System import pardir

class MySQL:

    def \_\_init\_\_(self,capacity=100):

        self.capacity=capacity

        self.root=os.path.join(pardir(os.getcwd()),"Database")#数据库根目录

        #print(os.path.abspath(self.root))

        if(not os.path.exists(self.root)):#根目录不存在时则自动新建

            os.mkdir(self.root)

        self.path=self.root#当前路径,默认为根路径

    def cksys(self,cmd):#检查是否是系统指令

        cmd=str.lower(cmd)

        if(cmd=="list"):

            print(os.listdir(self.path))

            return True

        elif(cmd=="cd."):

            if(self.path!=self.root):

                self.path=pardir(self.path)

            return True

        elif(cmd[0:3]=="cd "):

            if(self.path!=self.root):

                print('System Error:No access to the target file')

                return True

            if(len(cmd)<3):

                print('System Error:No valid expression after "cd".')

                return True

            db\_name=cmd[3:]

            target=os.path.join(self.path,db\_name)

            if(os.path.exists(target)):

                self.path=target

            else:print('System Error:No such Database called "'+db\_name+'",try "Create Database" first.')

            return True

        #elif(cmd=="help"):

        return False

    def Run(self):

        while(True):

            sql=self.Get()

            if (sql=="exit"):

                print("Exited")

                break

            if(sql=="sys"):

                continue

            self.Analyze(sql)

    def Get(self):

        print(self.path+":")

        sql=input()

        if(sql.lower().\_\_contains\_\_("exit")):

            return "exit"

        if(self.cksys(sql)):

            return "sys"

        sql=sql.rstrip()

        while(not sql.endswith(';')):

            sql+=' '+input()

            sql=sql.rstrip()

        return sql

    def Analyze(self,sql):

        #预处理

        sql=sql.lstrip()#删除前导空格

        sql=re.sub("\\s{2,}"," ",sql)#将所有多空格转换为单空格

        sql=re.sub(" ;",";",sql)#删除;前的空格

        sql=sql.lower()#转换为小写

        sql=sql.replace(";","")#删除分号

        mat=re.match("[a-z]+ ",sql)

        if(mat):

            key=mat.group()

            sql=sql.replace(key,"")#根据首个关键词执行对应SQL操作

            if(key=="create "):#Create

                self.path=Create.excute(self.root,self.path,self.capacity,sql)

            elif(key=="drop "):#Drop

                Drop.excute(self.root,self.path,sql)

            elif(self.path==self.root):

                print("System Error:Current SQL sentences must be excuted under certain database!")

            elif(key=="insert "):#Insert

                Insert.excute(self.path,sql)

            elif(key=="delete "):#Delete

                Delete.excute(self.path,self.capacity,sql)

            elif(key=="update "):#Update

                Update.excute(self.path,self.capacity,sql)

            elif(key=="select "):#Select

                Select.excute(self.path,self.capacity,sql)

            else:print("Syntax Error:SQL keyword not found.")

        else:print("Syntax Error:Invalid SQL expression.")

1. BPlusTree类

#该文件定义和实现复合搜索码(value),B+树(BPlusTree)及其节点(Node)

#具体的技术原理见"数据库系统概念"-"索引"-"B+树"部分,此处不作赘述

import queue as que

class value:#复合搜索码(非唯一性搜索码)

    def \_\_init\_\_(self,v,extra):

        self.v=v#原始搜索码值

        self.extra=extra#唯一化属性

    #运算符重载

    def \_\_lt\_\_(self,t):#小于

        if(type(t)!=value):

            return self.v<t

        if(self.v==t.v):

            return self.extra<t.extra

        return self.v<t.v

    def \_\_le\_\_(self,t):#小于等于

        if(type(t)!=value):

            return self.v<=t

        if(self.v==t.v):

            return self.extra<=t.extra

        return self.v<t.v

    def \_\_gt\_\_(self,t):#大于

        if(type(t)!=value):

            return self.v>t

        if(self.v==t.v):

            return self.extra>t.extra

        return self.v>t.v

    def \_\_ge\_\_(self,t):#大于等于

        if(type(t)!=value):

            return self.v>=t

        if(self.v==t.v):

            return self.extra>=t.extra

        return self.v>t.v

    def \_\_eq\_\_(self,t):#等于

        if(type(t)!=value):

            return self.v==t

        return self.v==t.v and self.extra==t.extra

    def \_\_ne\_\_(self,t):#不等于

        if(type(t)!=value):

            return self.v!=t

        return self.v!=t.v or self.extra!=t.extra

class Node:#B+树节点

    def \_\_init\_\_(self,K,P,size=0,isleaf=False):

        self.K=K#搜索码值

        self.P=P#指针

        self.size=size#搜索码数量

        self.isleaf=isleaf#是否是叶子结点

        self.par=None#父节点

        self.parp=None#父节点指向该节点的指针的下标

    def set(self,K,P,size):#设置节点属性

        self.K=K

        self.P=P

        self.size=size

class BPlusTree:#B+树

    def \_\_init\_\_(self,n=4):

        self.root=None#根节点

        self.n=n#B+树节点的最大孩子数

    def lower\_bound(self,it,v):#二分搜索,返回当前节点中大于等于当前搜素码的最小值下标;不存在时返回None

        l=0

        r=it.size-1

        if(it.K[r]<v):

            return None

        while(l<r):

            mid=(l+r)//2

            if(it.K[mid]>=v):

                r=mid

            else:l=mid+1

        return l

    def draw(self):#以BFS方式打印当前B+树

        if(self.root==None):

            print([])

            return

        dep=0

        q=que.Queue()

        q.put([self.root,1])

        while(not q.empty()):

            [it,d]=q.get()

            if(type(it.K[0])==value):#对于复合搜索码,只打印原始搜索码值

                kv=[k.v for k in it.K]

            else:kv=it.K

            if(d==dep):

                print(kv,end="")

            else:

                if(it==self.root):

                    print(kv,end="")

                else:

                    print()

                    print(kv,end="")

                dep+=1

            if(not it.isleaf):

                for p in it.P:

                    q.put([p,d+1])

        print()

    def findfirst(self,it,v):#返回B+树叶子层中含有搜索码值v的最左侧叶子节点

        if(it==None):

            return None

        if(it.isleaf):

            i=self.lower\_bound(it,v)

            if(i==None):

                return None

            return it

        i=self.lower\_bound(it,v)

        if(i==None):

            return self.findfirst(it.P[it.size],v)

        attempt=self.findfirst(it.P[i],v)

        if(attempt!=None):

            return attempt

        return self.findfirst(it.P[i+1],v)

    def find(self,v):#查询属性值等于指定搜索码值的元组的存储地址

        it=self.findfirst(self.root,v)

        if(it==None):

            return None

        result=[]

        i=self.lower\_bound(it,v)

        done=False

        while(not done):

            if(i<it.size and it.K[i]==v):

                result.append(it.P[i])

                i+=1

            elif(i==it.size and it.P[it.size]!=None):

                it=it.P[it.size]

                i=0

            else:done=True

        if(not len(result)==0):

            return result

        return None

    def findRange(self,lb,ub):#查询属性值在指定搜索码值区间(lb<=v<=ub)的元组的存储地址

        it=self.findfirst(self.root,lb)

        if(it==None):

            return None

        result=[]

        i=self.lower\_bound(it,lb)

        done=False

        while(not done):

            if(i<it.size and it.K[i]<=ub):

                result.append(it.P[i])

                i+=1

            elif(i<it.size and it.K[i]>ub):

                done=True

            elif(i==it.size and it.P[it.size]!=None):

                it=it.P[it.size]

                i=0

            else:done=True

        if(not len(result)==0):

            return result

        return None

    def findlt(self,ub):#查询属性值在指定搜索码值小于ub的元组的存储地址

        it=self.root

        if(it==None):

            return None

        while(not it.isleaf):

            it=it.P[0]

        result=[]

        i=0

        done=False

        while(not done):

            if(i<it.size and it.K[i]<ub):

                result.append(it.P[i])

                i+=1

            elif(i<it.size and it.K[i]>=ub):

                done=True

            elif(i==it.size and it.P[it.size]!=None):

                it=it.P[it.size]

                i=0

            else:done=True

        if(not len(result)==0):

            return result

        return None

    def findle(self,ub):#查询属性值在指定搜索码值小于等于ub的元组的存储地址

        it=self.root

        if(it==None):

            return None

        while(not it.isleaf):

            it=it.P[0]

        result=[]

        i=0

        done=False

        while(not done):

            if(i<it.size and it.K[i]<=ub):

                result.append(it.P[i])

                i+=1

            elif(i<it.size and it.K[i]>ub):

                done=True

            elif(i==it.size and it.P[it.size]!=None):

                it=it.P[it.size]

                i=0

            else:done=True

        if(not len(result)==0):

            return result

        return None

    def findgt(self,lb):#查询属性值在指定搜索码值大于lb的元组的存储地址

        it=self.findfirst(self.root,lb)

        if(it==None):

            return None

        result=[]

        i=self.lower\_bound(it,lb)

        if(it.K[i]==lb):

            i=i+1

        done=False

        while(not done):

            if(i<it.size):

                result.append(it.P[i])

                i+=1

            elif(i==it.size and it.P[it.size]!=None):

                it=it.P[it.size]

                i=0

            else:done=True

        if(not len(result)==0):

            return result

        return None

    def findge(self,lb):#查询属性值在指定搜索码值大于等于lb的元组的存储地址

        it=self.findfirst(self.root,lb)

        if(it==None):

            return None

        result=[]

        i=self.lower\_bound(it,lb)

        done=False

        while(not done):

            if(i<it.size):

                result.append(it.P[i])

                i+=1

            elif(i==it.size and it.P[it.size]!=None):

                it=it.P[it.size]

                i=0

            else:done=True

        if(not len(result)==0):

            return result

        return None

    def findne(self,v):#查询属性值在指定搜索码值不等于v的元组的存储地址

        it=self.root

        if(it==None):

            return None

        while(not it.isleaf):

            it=it.P[0]

        result=[]

        i=0

        done=False

        while(not done):

            if(i<it.size and it.K[i]!=v):

                result.append(it.P[i])

                i+=1

            elif(i<it.size and it.K[i]==v):

                i+=1

            elif(i==it.size and it.P[it.size]!=None):

                it=it.P[it.size]

                i=0

            else:done=True

        if(not len(result)==0):

            return result

        return None

    def insert(self,k,p):#向B+树索引中插入搜索码值为k,对应元组存储地址为p的键值对

        if(self.root==None):

            it=self.root=Node([],[None],isleaf=True)

            #很令人迷惑的特性:如果类的某个属性的默认参数是[],当新建一个对象时,python不会真的拿[]去初始化该属性,

            #而是使用上一次实例化这类对象的对应属性去初始化该属性。以Node类为例,不能写作def \_\_init\_\_(self,K=[]),it=Node()

            #而只能写作def \_\_init\_\_(self,K),并在调用时写作it=Node(K=[])

        else:

            it=self.root

            while(not it.isleaf):

                i=self.lower\_bound(it,k)

                if(i==None):

                    it.P[it.size].par=it

                    it=it.P[it.size]

                else:

                    it.P[i].par=it

                    it=it.P[i]

        if(it.size<self.n-1):

            self.insert\_in\_leaf(it,k,p)

        else:

            t=Node(it.K,it.P[:it.size+1],it.size)

            self.insert\_in\_leaf(t,k,p)

            that=Node(t.K[(t.size+1)//2:t.size],t.P[(t.size+1)//2:t.size+1],t.size-(t.size+1)//2,True)

            it.set(t.K[:(t.size+1)//2],t.P[:(t.size+1)//2]+[that],(t.size+1)//2)

            self.insert\_in\_parent(it,that.K[0],that)

    def insert\_in\_leaf(self,it,k,p):#在插入过程中更新B+树索引的叶子层,由insert函数自行调用

        if(it.size==0):

            it.K.append(k)

            it.P.insert(0,p)

        elif(it.K[0]>k):

            it.K.insert(0,k)

            it.P.insert(0,p)

        else:

            i=self.lower\_bound(it,k)

            if(i==None):

                it.K.append(k)

                it.P.insert(it.size,p)

            else:

                it.K.insert(i,k)

                it.P.insert(i,p)

        it.size+=1

    def insert\_in\_parent(self,it,k,that):#在插入过程中更新B+树索引的内部结点,由insert函数自行调用

        if(it==self.root):

            self.root=Node([k],[it,that],1)

            return

        par=it.par

        if(par.size+1<self.n):

            i=self.lower\_bound(par,k)

            if(i==None):

                par.K.append(k)

                par.P.append(that)

            else:

                par.K.insert(i,k)

                par.P.insert(i+1,that)

            par.size+=1

        else:

            t=Node(par.K,par.P,par.size)

            i=self.lower\_bound(t,k)

            if(i==None):

                t.K.append(k)

                t.P.append(that)

            else:

                t.K.insert(i,k)

                t.P.insert(i+1,that)

            t.size+=1

            par.set(t.K[:(t.size+1)//2],t.P[:(t.size+1)//2+1],(t.size+1)//2)

            k=t.K[(t.size+1)//2]

            part=Node(t.K[(t.size+1)//2+1:t.size],t.P[(t.size+1)//2+1:t.size+1],t.size-(t.size+1)//2-1)

            self.insert\_in\_parent(par,k,part)

    def delete(self,k):#删除B+树索引中搜索码值为k的键值对(此时的搜索码是唯一性搜索码)

        it=self.root

        while(not it.isleaf):

            i=self.lower\_bound(it,k)

            if(i==None):

                it.P[it.size].par=it

                it.P[it.size].parp=it.size

                it=it.P[it.size]

            elif(it.K[i]==k):

                it.P[i+1].par=it

                it.P[i+1].parp=i+1

                it=it.P[i+1]

            else:

                it.P[i].par=it

                it.P[i].parp=i

                it=it.P[i]

        self.delete\_entry(it,k)

    def delete\_entry(self,it,k,offset=0):#在删除过程中更新B+树索引的内部节点,由delete函数自行调用

        i=self.lower\_bound(it,k)

        del it.K[i]

        del it.P[i+offset]

        it.size-=1

        if(it==self.root and it.size==0):

            self.root=it.P[0]

            del it

            return

        if((it.isleaf and it.size<self.n//2) or (not it.isleaf and it.size<(self.n-1)//2)):

            if(it==self.root):

                return

            par=it.par

            i=it.parp

            if(i==par.size):

                left=par.P[i-1]

                right=None

            elif(i==0):

                left=None

                right=par.P[i+1]

            else:

                left=par.P[i-1]

                right=par.P[i+1]

            if(left!=None and left.size+it.size<self.n):

                k=par.K[i-1]

                if(not it.isleaf):

                    left.set(left.K+[k]+it.K,left.P+it.P,left.size+it.size+1)

                else:left.set(left.K+it.K,left.P[:left.size]+it.P,left.size+it.size)

                self.delete\_entry(par,k,1)

                del it

            elif(right!=None and it.size+right.size<self.n):

                k=par.K[i]

                if(not right.isleaf):

                    it.set(it.K+[k]+right.K,it.P+right.P,it.size+right.size+1)

                else:it.set(it.K+right.K,it.P[:it.size]+right.P,it.size+right.size)

                self.delete\_entry(par,k,1)

                del right

            else:

                if(left!=None):

                    k=par.K[i-1]

                    left.size-=1

                    lastk=left.K.pop()

                    if(not it.isleaf):

                        lastp=left.P.pop()

                        it.K=[k]+it.K

                    else:

                        lastp=left.P.pop(left.size-1)

                        it.K=[lastk]+it.K

                    it.P=[lastp]+it.P

                    par.K[i-1]=lastk

                else:

                    k=par.K[i]

                    right.size-=1

                    firstk=right.K.pop(0)

                    firstp=right.P.pop(0)

                    if(not it.isleaf):

                        it.K=it.K+[k]

                        it.P.append(firstp)

                    else:

                        it.K=it.K+[firstk]

                        it.P.insert(it.size,firstp)

                    par.K[i]=firstk

                it.size+=1

1. Create类

#该文件定义和实现数据库和关系模式的创建

import re

import os

import pandas as pd

import numpy as np

from System import pardir,match

from BPlusTree import BPlusTree as BPT

def excute(root,path,capacity,sql):

    mat=re.match("[a-z]+ ",sql)#匹配"Create"关键词后的第一个英文单词

    if(mat):

        key=mat.group()

        sql=sql.replace(key,"")

        if(key=="database "):#如果是"Database",则创建数据库

            if(path==root):

                return createDB(path,sql)

            else:

                print("Create Error:Already in database!")

        elif(key=="table "):#如果是"Table",则创建关系模式

            if(pardir(path)==root):

                createTB(path,capacity,sql)

            else:print("Create Error:Table could only be created under database!")

        else:print('Syntax Error:Keyword "'+key+'"after "Create" out of SQL criteria,try "Database" or "Table".')

    else:

        print('Syntax Error:Invalid expression after "Create".')

    return path

def createDB(path,db\_name):#创建数据库

    target=os.path.join(path,db\_name)

    if(not os.path.exists(target)):#如果指定数据库不存在,则创建

        os.mkdir(target)

        print('Database "'+db\_name+'" has been created.')

        return target

    else:

        print('Database "'+db\_name+'" already exists,you could input "cd '+db\_name+'" to re-enter it.')

        return path

def createTB(path,capacity,sql):

    idx=sql.find('(')#查询语法中是否含有括号,以及括号数量是否匹配

    if(idx==-1):

        print("Syntax Error:No left bracket after Table.")

        return

    if(not match(sql)):

        print("Syntax Error:Invalid brackets after Table.")

        return

    tb\_name=sql[:idx]#获取关系模式名称

    tb\_name=tb\_name.replace(" ","")

    target=os.path.join(path,tb\_name)#获取即将创建上述关系模式的文件夹路径

    if(os.path.exists(target)):#如果关系模式已存在

        print('Create Error:Table "'+tb\_name+'" already exists.')

        return

    primary\_key=""#主码

    name=[]#属性

    datatype=[]#数据类型

    types=[]#基础数据类型

    constraints=[]#属性上的约束

    sql=sql[idx+1:-1]

    phrases=sql.split(',')#以','为分隔符获取各个属性或约束的描述

    for phrase in phrases:

        if(phrase[:11]=="primary key"):#如果描述的是主码约束

            if(primary\_key==""):#如果主码不存在,则获取主码

                if(phrase[12]=='('):

                    primary\_key=phrase[13:-1]

                else:primary\_key=phrase[12:-1]

            else:print("Constraint Error:Mutiple primary key.")

        else:

            sp1=phrase.find(' ')#以' '为分隔符获取各个属性描述的细节

            sp2=phrase.find(' ',max(sp1+1,0))

            if(sp1==-1):

                print("Syntax Error:Invalid attribute expression "+phrase+".")

                return

            name.append(phrase[:sp1])#获取第一个细节:属性名

            if(sp2==-1):

                if(phrase[sp1+1:]=="int"):

                    types.append("int")

                elif(phrase[sp1+1:].\_\_contains\_\_("varchar")):

                    types.append("varchar")

                else:

                    print('Type Error:Datatype could only be "int" or "varchar",instead of "'+phrase[sp1+1:]+'" .')

                    return

                datatype.append(phrase[sp1+1:])#获取第二个细节:属性的数据类型

                constraints.append(None)#获取第三个细节:属性上的约束(optional)

            else:

                if(phrase[sp1+1:sp2]=="int"):

                    types.append("int")

                elif(phrase[sp1+1:sp2].\_\_contains\_\_("varchar")):

                    types.append("varchar")

                else:

                    print('Type Error:Datatype could only be "int" or "varchar",instead of "'+phrase[sp1+1:sp2]+'" .')

                    return

                datatype.append(phrase[sp1+1:sp2])

                constraints.append(phrase[sp2:])

    if(primary\_key not in name):#主码描述的属性不存在

        print('Create Error:Primary key "'+primary\_key+'" not found in attributes.')

        return

    if(primary\_key==""):#未指定主码

        print('Create Error:Keyword "primary key" not found.')

        return

    pos=name.index(primary\_key)#记录主码是关系模式中的第pos个属性

    os.mkdir(target)#以文件夹形式在当前数据库文件夹下创建关系模式

    table=pd.DataFrame([["" for attr in name] for i in range(capacity)],columns=name)

    table.to\_csv(os.path.join(target,"Table.csv"))#在关系模式文件夹下以.csv格式创建空的关系模式表,用行坐标模拟存储地址

    target\_index=os.path.join(target,"Index")

    os.mkdir(target\_index)#在关系模式文件夹下创建索引文件夹

    np.save(os.path.join(target\_index,"Empty.npy"),set(i for i in range(capacity)))#在索引文件夹下记录不含有效记录(元组)的存储地址

    for attr in name:

        np.save(os.path.join(target\_index,attr+".npy"),BPT())#为每个属性创建空的B+树索引

    #记录元数据:主码(primary),主码下标(position),属性数(Property),最大存储容量(Capacity)

    meta=pd.concat([pd.DataFrame({"Description":["Primary","Position","Property","Capacity"]}),pd.DataFrame({" ":[primary\_key,pos,len(name),capacity]})],axis=1)

    #属性名(Attribute),数据类型(Datatype),基本数据类型(Basictype),属性上的约束(Constraints)

    meta=pd.concat([meta,pd.DataFrame({"Attribute":name}),pd.DataFrame({"Datatype":datatype}),pd.DataFrame({"Basictype":types}),pd.DataFrame({"Constraints":constraints})],axis=1)

    meta.to\_csv(os.path.join(target,"Meta.csv"),index=False)#以.csv格式在关系模式文件夹下存储元数据

    print('Table "'+tb\_name+'" has been created.')

1. Drop类

#该文件定义和实现数据库和关系模式的删除

import os

import shutil

import re

from System import pardir

def excute(root,path,sql):

    mat=re.match("[a-z]+ ?",sql)#匹配"Drop"关键词后的第一个英文单词

    if(mat):

        key=mat.group()

        sql=sql.replace(key,"")

        if(key=="database "):#如果是"Database",则删除数据库

            if(path==root):

                dropDB(path,sql)

            else:

                print("Drop Error:Exit the current database first.")

        elif(key=="table "):#如果是"Table",则创建关系模式

            if(pardir(path)==root):

                dropTB(path,sql)

            else:print("Drop Error:Table could only be dropped under database!")

        else:print('Syntax Error:Keyword "'+key+'"after "Drop" out of SQL criteria,try "Database" or "Table".')

    else:

        print('Syntax Error:Invalid expression after "Drop".')

def dropDB(path,db\_name):

    db\_name=db\_name.replace(" ","")

    target=os.path.join(path,db\_name)

    if(os.path.exists(target)):#数据库存在

        shutil.rmtree(target)

        print('Database "'+db\_name+'" has been removed.')

    else:print('Database "'+db\_name+'" does not exist.')

def dropTB(path,tb\_name):

    tb\_name=tb\_name.replace(" ","")

    target=os.path.join(path,tb\_name)

    if(os.path.exists(target)):#关系模式存在

        shutil.rmtree(target)

        print('Table "'+tb\_name+'" has been removed.')

    else:print('Table "'+tb\_name+'" does not exist.')

1. Insert类

#该文件定义和实现了向指定关系模式的元组插入操作

import os

import pandas as pd

import numpy as np

from BPlusTree import value

def excute(path,sql):

    if(sql[0:5]!="into "):#检查关键词

        print('Syntax Error:Invalid expression after "Insert".')

        return

    if(not sql.\_\_contains\_\_(" values")):

        print('Syntax Error:Keyword "values" not found after "Insert into".')

        return

    sql=sql.replace("'","")#排列顺序一致时单引号或双引号不是必要的

    sql=sql.replace('"','')

    idx=sql.find(" values")

    idx1=sql.find('(')

    idx2=sql.find(')')

    tb\_name=sql[5:idx]#获取关系模式名称

    tb\_name=tb\_name.replace(" ","")

    values=sql[idx1+1:idx2].replace(" ","").split(',')#以','为分隔符获取各个属性值

    if(len(values)==0):#无有效属性值

        print('Syntax Error:Values equal to zero.')

    target=os.path.join(path,tb\_name)#获取指定关系模式的文件夹路径

    if(not os.path.exists(target)):#如果指定关系模式不存在

        print('Insert Error:Table "'+tb\_name+'" does not exist.')

        return

    meta=pd.read\_csv(os.path.join(target,"Meta.csv"))#读取元数据

    size=int(meta[" "].iloc[2])#属性数

    attr=[]

    for a in meta["Attribute"]:#属性名

        if(type(a)!=float):

            attr.append(a)

        else:break

    basic=meta["Basictype"]#数据类型

    primary=meta[" "].iloc[0]#主码

    pos=int(meta[" "].iloc[1])#主码下标

    if(len(values)>size):#如果插入的属性数大于关系模式属性数

        print("Insert Error:Values number ",len(values)," larger than attribute numbers ",size,"!")

        return

    table=pd.read\_csv(os.path.join(target,"Table.csv"),index\_col=0)#读取关系模式

    target\_index=os.path.join(target,"Index")#获取指定关系模式的索引文件夹路径

    index={attr[i]:np.load(os.path.join(target\_index,attr[i]+".npy"),allow\_pickle=True).item() for i in range(len(attr))}#读取索引

    Empty=np.load(os.path.join(target\_index,"Empty.npy"),allow\_pickle=True).item()#读取空闲的存储地址

    if(len(Empty)==0):#如果没有空闲的存储地址

        print("Insert Error:The storage capacity has reached the maximum limit.")

        return

    idx=Empty.pop()#模拟操作系统,以伪随机的方式获取当前元组被分配的存储地址

    for i in range(len(values)-1):

        if(basic.iloc[i]=="int"):

            values[i]=int(values[i])#将属性值的数据类型转变为真实数据类型(str->Any)

    table.iloc[idx]=values#将元组写入关系模式表

    if(index[primary].find(values[pos])!=None):#如果主码对应属性的属性值已记录

        print("Insert Error:Violation of primary key constraints!")

        return

    for i in range(len(values)):

        index[attr[i]].insert(value(values[i],values[pos]),idx)#插入复合搜索码

    meta.to\_csv(os.path.join(target,"Meta.csv"),index=False)#写回元数据

    table.to\_csv(os.path.join(target,"Table.csv"))#写回关系模式

    np.save(os.path.join(target\_index,"Empty.npy"),Empty)#写回空闲地址

    for i in range(len(attr)):

        np.save(os.path.join(target\_index,attr[i]+".npy"),index[attr[i]])#写回索引

    print("Insert excuted.")

1. Delete类

#该文件定义和实现了从指定关系模式中删除符合特定条件的元组的操作

import os

import shutil

import pandas as pd

import numpy as np

from BPlusTree import BPlusTree as BPT

from BPlusTree import value

def excute(path,capacity,sql):

    if(sql[0:5]!="from "):#检查关键词

        print('Syntax Error:Invalid expression after "Delete".')

        return

    idx=sql.find(" where ")

    if(idx==-1):

        tb\_name=sql[5:]

    else:tb\_name=sql[5:idx]

    tb\_name=tb\_name.replace(" ","")

    target=os.path.join(path,tb\_name)#获取指定关系模式的文件夹路径

    if(not os.path.exists(target)):#如果指定关系模式不存在

        print('Insert Error:Table "'+tb\_name+'" does not exist.')

        return

    meta=pd.read\_csv(os.path.join(target,"Meta.csv"))#读取元数据

    attributes=[]

    for a in meta["Attribute"]:#属性名

        if(type(a)!=float):

            attributes.append(a)

        else:break

    target\_index=os.path.join(target,"Index")

    if(idx==-1):#如果不含'WHERE'关键词

        os.remove(os.path.join(target,"Table.csv"))

        table=pd.DataFrame([["" for j in range(len(attributes))] for i in range(capacity)],columns=attributes)

        table.to\_csv(os.path.join(target,"Table.csv"))#在关系模式文件夹下以.csv格式重新创建空的关系模式表

        shutil.rmtree(target\_index)#完全删除所有索引

        os.mkdir(target\_index)#在关系模式文件夹下创建索引文件夹

        np.save(os.path.join(target\_index,"Empty.npy"),set(i for i in range(capacity)))#在索引文件夹下重新初始化不含有效记录(元组)的存储地址

        for attr in attributes:

            np.save(os.path.join(target\_index,attr+".npy"),BPT())#为每个属性重建空的B+树索引

        print("Delete excuted.")

        return

    sql=sql[idx+7:]

    Empty=np.load(os.path.join(target\_index,"Empty.npy"),allow\_pickle=True).item()#读取空闲的存储地址

    index={attributes[i]:np.load(os.path.join(target\_index,attributes[i]+".npy"),allow\_pickle=True).item() for i in range(len(attributes))}#读取索引

    result=set(i for i in range(capacity))#初始化结果集合

    search=sql.split(" and ")

    for sear in search:

        sear=sear.replace(" ","")

        if(sear.\_\_contains\_\_('>=')):#对每个谓词检查需求

            cmp='>='

            key\_value=sear.split('>=')

        elif(sear.\_\_contains\_\_('<=')):

            cmp='<='

            key\_value=sear.split('<=')

        elif(sear.\_\_contains\_\_('!=')):

            cmp='!='

            key\_value=sear.split('!=')

        elif(sear.\_\_contains\_\_('>')):

            cmp='>'

            key\_value=sear.split('>')

        elif(sear.\_\_contains\_\_('<')):

            cmp='<'

            key\_value=sear.split('<')

        elif(sear.\_\_contains\_\_('<>')):

            cmp='<>'

            key\_value=sear.split('<>')

        elif(sear.\_\_contains\_\_('=')):

            cmp='='

            key\_value=sear.split('=')

        else:

            print('Delete Error:No comparison operator like ">",">=","<","<=","=","!=" or "<>" found in "'+sear+'"!')

            return

        key=key\_value[0]#分割属性和属性值

        if(key not in attributes):

            print('Delete Error:Attribute "'+key+'" not contained in "'+tb\_name+'".')

            return

        val=key\_value[1]

        if((val[0]=="'" and val[-1]=="'") or (val[0]=='"' and val[-1]=='"')):#判断数据类型

            val=val[1:-1]

        else:val=int(val)

        if(cmp=='>'):

            res=index[key].findgt(val)

        elif(cmp=='>='):

            res=index[key].findge(val)

        elif(cmp=='<'):

            res=index[key].findlt(val)

        elif(cmp=='<='):

            res=index[key].findle(val)

        elif(cmp=='='):

            res=index[key].find(val)

        else:

            res=index[key].findne(val)

        if(res==None):

            result=None

            break

        result=result & set(res)#求检索结果的交集

    if(result==None):#集合为空则不进行任何删除

        print("Delete excuted.")

        return

    primary=meta[" "].iloc[0]#主码

    pos=int(meta[" "].iloc[1])#主码下标

    basic=meta["Basictype"]#数据类型

    table=pd.read\_csv(os.path.join(target,"Table.csv"),index\_col=0)#读取关系模式

    Empty=Empty | result#更新可存储地址

    for res in result:#更新索引

        if(basic.iloc[pos]=="int"):

            extra=int(table[primary].iloc[res])

        else:extra=table[primary].iloc[res]#获取主码

        for i in range(len(attributes)):

            attr=attributes[i]

            if(basic.iloc[i]=="int"):

                v=int(table[attr].iloc[res])

            else:v=table[attr].iloc[res]

            index[attr].delete(value(v,extra))#组合为唯一性复合搜索码,执行索引删除操作

    for res in result:

        table.iloc[res]=["" for attr in attr]#从关系模式中删除数据

    table.to\_csv(os.path.join(target,"Table.csv"))#写回关系模式

    np.save(os.path.join(target\_index,"Empty.npy"),Empty)#写回空闲地址

    for attr in attributes:

        np.save(os.path.join(target\_index,attr+".npy"),index[attr])#写回索引

    print("Delete excuted.")

1. Update类

#该文件定义和实现了在指定关系模式上更新符合特定条件的元组的操作

import os

import numpy as np

import pandas as pd

from BPlusTree import value

def excute(path,capacity,sql):

    idx1=sql.find(" set ")

    if(idx1==-1):

        print('Syntax Error:Keyword "Set" not found after "Update".')

        return

    tb\_name=sql[:idx1]

    tb\_name=tb\_name.replace(" ","")

    target=os.path.join(path,tb\_name)#获取指定关系模式的文件夹路径

    if(not os.path.exists(target)):#如果指定关系模式不存在

        print('Update Error:Table "'+tb\_name+'" does not exist.')

        return

    idx2=sql.find(" where ")#检索where关键词

    if(idx2==-1):

        reset=sql[idx1+5:]

    else:

        reset=sql[idx1+5:idx2]

        sql=sql[idx2+7:]

    meta=pd.read\_csv(os.path.join(target,"Meta.csv"))#读取元数据

    attributes=[]

    for a in meta["Attribute"]:#属性名

        if(type(a)!=float):

            attributes.append(a)

        else:break

    target\_index=os.path.join(target,"Index")

    index={attributes[i]:np.load(os.path.join(target\_index,attributes[i]+".npy"),allow\_pickle=True).item() for i in range(len(attributes))}#读取索引

    result=set(i for i in range(capacity))#初始化结果集

    if(idx2!=-1):

        search=sql.split(" and ")

        for sear in search:

            sear=sear.replace(" ","")

            if(sear.\_\_contains\_\_('>=')):

                cmp='>='

                key\_value=sear.split('>=')

            elif(sear.\_\_contains\_\_('<=')):

                cmp='<='

                key\_value=sear.split('<=')

            elif(sear.\_\_contains\_\_('!=')):

                cmp='!='

                key\_value=sear.split('!=')

            elif(sear.\_\_contains\_\_('>')):

                cmp='>'

                key\_value=sear.split('>')

            elif(sear.\_\_contains\_\_('<')):

                cmp='<'

                key\_value=sear.split('<')

            elif(sear.\_\_contains\_\_('<>')):

                cmp='<>'

                key\_value=sear.split('<>')

            elif(sear.\_\_contains\_\_('=')):

                cmp='='

                key\_value=sear.split('=')

            else:

                print('Update Error:No comparison operator like ">",">=","<","<=","=","!=" or "<>" found in "'+sear+'"!')

                return

            key=key\_value[0]

            if(key not in attributes):

                print('Update Error:Attribute "'+key+'" not contained in "'+tb\_name+'".')

                return

            val=key\_value[1]

            if((val[0]=="'" and val[-1]=="'") or (val[0]=='"' and val[-1]=='"')):

                val=val[1:-1]

            else:val=int(val)

            if(cmp=='>'):

                res=index[key].findgt(val)

            elif(cmp=='>='):

                res=index[key].findge(val)

            elif(cmp=='<'):

                res=index[key].findlt(val)

            elif(cmp=='<='):

                res=index[key].findle(val)

            elif(cmp=='='):

                res=index[key].find(val)

            else:

                res=index[key].findne(val)

            if(res==None):

                result=None

                print("Update excuted.")

                break

            result=result & set(res)

    if(result==None):#集合为空则不进行任何更新

        print("Update excuted.")

        return

    reset=reset.replace(" ","")

    key\_value=reset.split('=')

    key=key\_value[0]

    val=key\_value[1]

    idx=attributes.index(key)#更新属性下标

    if(idx==-1):

        print('Update Error:Attribute "'+key+'" not contained in the table "'+tb\_name+'".')

        return

    primary=meta[" "].iloc[0]#主码

    pos=int(meta[" "].iloc[1])#主码下标

    basic=meta["Basictype"]#数据类型

    table=pd.read\_csv(os.path.join(target,"Table.csv"),index\_col=0)#读取关系模式

    nvk=(not val.\_\_contains\_\_(key))#检索是否是自运算

    if(nvk):

        if(basic.iloc[idx]=="int"):#不是则直接得到更新值

            tv=int(val)

        else:

            tv=val.replace("'","")

            tv=tv.replace('"','')

    elif(basic.iloc[idx]=="int"):#否则检索自运算的操作符

        op=val.find('+')

        if(op==-1):

            op=val.find('-')

        if(op==-1):

            op=val.find('\*')

        if(op==-1):

            op=val.find('/')

        if(op!=-1):

            dv=int(val[op+1:])

        else:

            print('Update Error:"+","-","\*" and "/" not found.')

            return

    else:

        print('Update Error:"+","-","\*" and "/" not supported on attribute "'+key+'" with type "str".')

        return

    for res in result:#将索引更新拆分为旧索引的删除和新索引的插入

        if(basic.iloc[pos]=="int"):

            extra=int(table[primary].iloc[res])#计算唯一化属性

        else:extra=table[primary].iloc[res]

        if(basic.iloc[idx]=="int"):

            v=int(table[key].iloc[res])

        else:v=table[key].iloc[res]

        index[key].delete(value(v,extra))#删除更新前的索引

        if(not nvk):#计算自运算的结果

            if(val[op]=='+'):

                tv=int(table[key].iloc[res])+dv

            elif(val[op]=='-'):

                tv=int(table[key].iloc[res])-dv

            elif(val[op]=='\*'):

                tv=int(table[key].iloc[res])\*dv

            else:tv=int(table[key].iloc[res])/dv

        row=[vals for vals in table.iloc[res]]

        row[idx]=tv

        table.iloc[res]=row#更新关系模式

        index[key].insert(value(tv,extra),res)#插入更新后的索引

    table.to\_csv(os.path.join(target,"Table.csv"))#写回关系模式

    for attr in attributes:

        np.save(os.path.join(target\_index,attr+".npy"),index[attr])#写回索引

    print("Update excuted.")

1. Select类

#该文件定义和实现了在指定关系模式上的查询操作

import os

import numpy as np

import pandas as pd

def CrossJoin(A,B):#返回两个关系模式的笛卡尔积

    resultA=pd.concat([A,pd.DataFrame({"Key":[0 for i in range(len(A))]})],axis=1)

    resultB=pd.concat([B,pd.DataFrame({"Key":[0 for i in range(len(A))]})],axis=1)

    return resultA.merge(resultB,on="Key").drop("Key",axis=1)

def excute(path,capacity,sql):

    idxf=sql.find(" from ")

    if(idxf==-1):

        print('Syntax Error:Keyword "From" not found.')

        return

    select=sql[:idxf].replace(" ","").split(",")#检索投影属性

    idxw=sql.find(" where ")#检查"Where"关键词的存在性

    if(idxw==-1):

        tb\_name=sql[idxf+6:]#获取数据库名称

    else:tb\_name=sql[idxf+6:idxw]

    tb\_name=tb\_name.replace(" ","")

    if(tb\_name.\_\_contains\_\_(',')):#局限性

        print("'Limitation:Query on multiple relation schemas not supported.")

        return

    target=os.path.join(path,tb\_name)#获取指定关系模式的文件夹路径

    if(not os.path.exists(target)):#如果指定关系模式不存在

        print('Insert Error:Table "'+tb\_name+'" does not exist.')

        return

    meta=pd.read\_csv(os.path.join(target,"Meta.csv"))#读取元数据

    attributes=[]

    for a in meta["Attribute"]:#属性名

        if(type(a)!=float):

            attributes.append(a)

        else:break

    target\_index=os.path.join(target,"Index")

    index={attributes[i]:np.load(os.path.join(target\_index,attributes[i]+".npy"),allow\_pickle=True).item() for i in range(len(attributes))}#读取索引

    result=set(i for i in range(capacity))#初始化结果集合

    search=sql[idxw+7:].split(" and ")#根据"Where"条件筛选

    for sear in search:

        sear=sear.replace(" ","")

        if(sear.\_\_contains\_\_('>=')):

            cmp='>='

            key\_value=sear.split('>=')

        elif(sear.\_\_contains\_\_('<=')):

            cmp='<='

            key\_value=sear.split('<=')

        elif(sear.\_\_contains\_\_('!=')):

            cmp='!='

            key\_value=sear.split('!=')

        elif(sear.\_\_contains\_\_('>')):

            cmp='>'

            key\_value=sear.split('>')

        elif(sear.\_\_contains\_\_('<')):

            cmp='<'

            key\_value=sear.split('<')

        elif(sear.\_\_contains\_\_('<>')):

            cmp='<>'

            key\_value=sear.split('<>')

        elif(sear.\_\_contains\_\_('=')):

            cmp='='

            key\_value=sear.split('=')

        else:

            print('Select Error:No comparison operator like ">",">=","<","<=","=","!=" or "<>" found in "'+sear+'"!')

            return

        key=key\_value[0]

        if(key not in attributes):

            print('Select Error:Attribute "'+key+'" not contained in "'+tb\_name+'".')

            return

        val=key\_value[1]

        if((val[0]=="'" and val[-1]=="'") or (val[0]=='"' and val[-1]=='"')):

            val=val[1:-1]

        else:val=int(val)

        if(cmp=='>'):

            res=index[key].findgt(val)

        elif(cmp=='>='):

            res=index[key].findge(val)

        elif(cmp=='<'):

            res=index[key].findlt(val)

        elif(cmp=='<='):

            res=index[key].findle(val)

        elif(cmp=='='):

            res=index[key].find(val)

        else:

            res=index[key].findne(val)

        if(res==None):

            result=None

            break

        result=result & set(res)

        print("\nResult:")

        if(result==None):#集合为空则结果为空

            print()

            return

        table=pd.read\_csv(os.path.join(target,"Table.csv"),index\_col=0)#读取关系模式

        for sel in select:#打印属性列表

            print(sel,end="\t")

        print()

        for res in result:#打印符合条件的属性值

            for sel in select:

                print(table[sel].iloc[res],end="\t")

            print()

        print()

1. 使用手册

3.1基本设置

1)大小写不敏感

2)数据类型暂时只支持"int"和"varchar",后者的实际长度未作约束

3)字符串以" "或' '标识

4)暂时仅支持单一属性作为主码并支持主码约束的检测,暂不支持外码,属性集上的约束暂时只能记录为元数据

5)暂时仅支持单个关系模式上的数据筛选(Select)

6)下述”拒绝支持”一般意味着程序本身而非Python会打印对应的错误信息

7)运行方式为使用Vscode软件打开MySQL文件夹,并点击运行Console文件

3.2支持的指令

支持的指令如下(双引号内的内容代表用户在此处的输入值为一个自定义变量,且双引号本身不必输入):

**1)System类(以下不必以';'结尾)**

(1)"list":

打印当前目录。

(2)"cd 'name'":

进入名称为"name"的指定子目录。只允许从Database目录进入某个子目录(即某个数据库),非法访问将被拒绝。

(3)"cd.":

返回当前目录的父目录。只允许从Database的某个子目录(即某个数据库)返回Database目录,非法返回将被无效化。

(4)"exit":

退出程序。

**2)Create类**

(1)"Create Database 'db\_name';":

创建名为"db\_name"的数据库。该指令强制在Database目录下执行,在非法的目录下创建数据库会被拒绝(如在数据库中创建数据库)。

(2)"Create Database 'tb\_name'(attribute1 datatype1 constraint1,attribute2 datatype2,...,primary key(attribute));":

创建关系模式"tb\_name"。"Attribute","Datatype"和"Constraint"分别表示属性名,属性的数据类型和属性上的约束,其中属性上的约束是可选的(可写可不写)。"primary key"(主码)是必选的。

该指令强制在Database的某个子目录(即某个数据库)下执行,在非法的目录下创建关系模式会被拒绝(如在Database目录下创建关系模式)。

以下内容将被拒绝执行:指定数据库或关系模式已经存在/"Create"关键词后的对象不属于"Database"或"Table"中的任意一种/括号不匹配/声明多个主码/未声明主码。

**3)Drop类**

(1)"Drop Database 'db\_name';":

删除名为"db\_name"的数据库。该指令强制在Database目录下执行,在非法的目录下删除数据库会被拒绝(如在数据库中删除数据库)。

(2)"Drop Database 'tb\_name';":

删除名为"tb\_name"的关系模式。该指令强制在Database的某个子目录(即某个数据库)下执行,在非法的目录下删除关系模式会被拒绝(如在Database目录下删除关系模式)。此后的指令均遵守这一原则,故不再赘述。

以下内容将被拒绝执行:指定数据库或关系模式不存在/"Drop"关键词后的对象不属于"Database"或"Table"中的任意一种。

**4)Insert类**

(1)"Insert into 'tb\_name' values(value1,value2);":

在名为"tb\_name"的关系模式中插入一个元组,元组属性值的排列顺序和关系模式中对应属性的排列顺序一致。

以下内容将被兼容执行:元组属性值的数量小于等于关系模式(自动填充null)。

以下内容将被拒绝执行:违反主码约束/元组属性的数量大于关系模式/指定关系模式不存在/关键词"into"或"values"缺失/元组属性值的数据类型不符合对应的关系模式属性值的数据类型。

**5)Delete类**

(1)"Delete from 'tb\_name';":

删除名为"tb\_name"的关系模式中的所有元组,但保留关系模式本身。

(2)"Delete from 'tb\_name' where attribute1=value1 and attribute2>=value2 and...;":

删除名为"tb\_name"的关系模式中的符合条件的元组。

作为首个涉及"where"关键词的指令,目前暂时只支持用"and"关键词所连接的下列检索条件之一或任意组合:"**=**","**!=**"(**<>**),"**>**","**>=**","**<**","**<=**"。此后的指令均符合这一前提。

以下内容将被拒绝执行:指定的关系模式不存在/"Drop"关键词后缺少"from"关键词/"where"关键词后的检索条件无法支持/检索的属性不存在。

**6)Update类**

(1)"Update 'tb\_name' set attribute=value;":

更新名为"tb\_name"的关系模式中的所有元组的指定属性。

"set"关键词后的"attribute=expression"中的表达式expression可以为常量value(int或string型)或包含attribute本身和常量值value的组合式,支持的组合式为下列组合式之一:attribute=attribute+value,attribute=attribute-value,

attribute=attribute\*value,attribute=attribute/value。

(2)"Update 'tb\_name' set attribute=value; where attribute1=value1 and attribute2>=value2 and...;":

更新名为"tb\_name"的关系模式中的符合条件的元组。"set"关键词的相关内容同上。

以下内容将被拒绝执行:指定的关系模式不存在/"Update"关键词后缺少"set"关键词/表达式expression不合法/"where"关键词后的检索条件无法支持/检索的属性不存在。

**7)Select类**

(1)"Select attribut1,attribut2,...from 'tb\_name';"

选择名为"tb\_name"的关系模式中的所有元组的对应属性并打印在控制台。

尽管程序提供了笛卡尔积(Cross Join)的函数,但由于笛卡尔积通常需要和自然连接(Natural Join)和嵌套查询等相对更加高级的语法组合使用,单纯打印庞大的笛卡尔积结果是效率低下且一般没有实际意义的,因此在本次实验中暂不使用该函数,也因此暂不提供多个关系模式上的查询。

(2)"Select attribut1,attribut2,...from 'tb\_name' where attribute1=value1 and attribute2>=value2 and...;"

选择名为"tb\_name"的关系模式中的符合条件的元组的对应属性并打印在控制台。

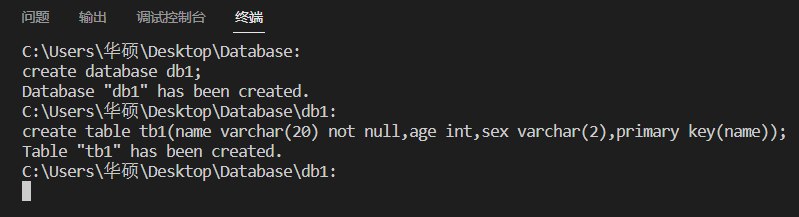
以下内容将被拒绝执行:指定的关系模式不存在/"Select"关键词后缺少"from"关键词/"where"关键词后的检索条件无法支持/检索的属性不存在。

3.3索引

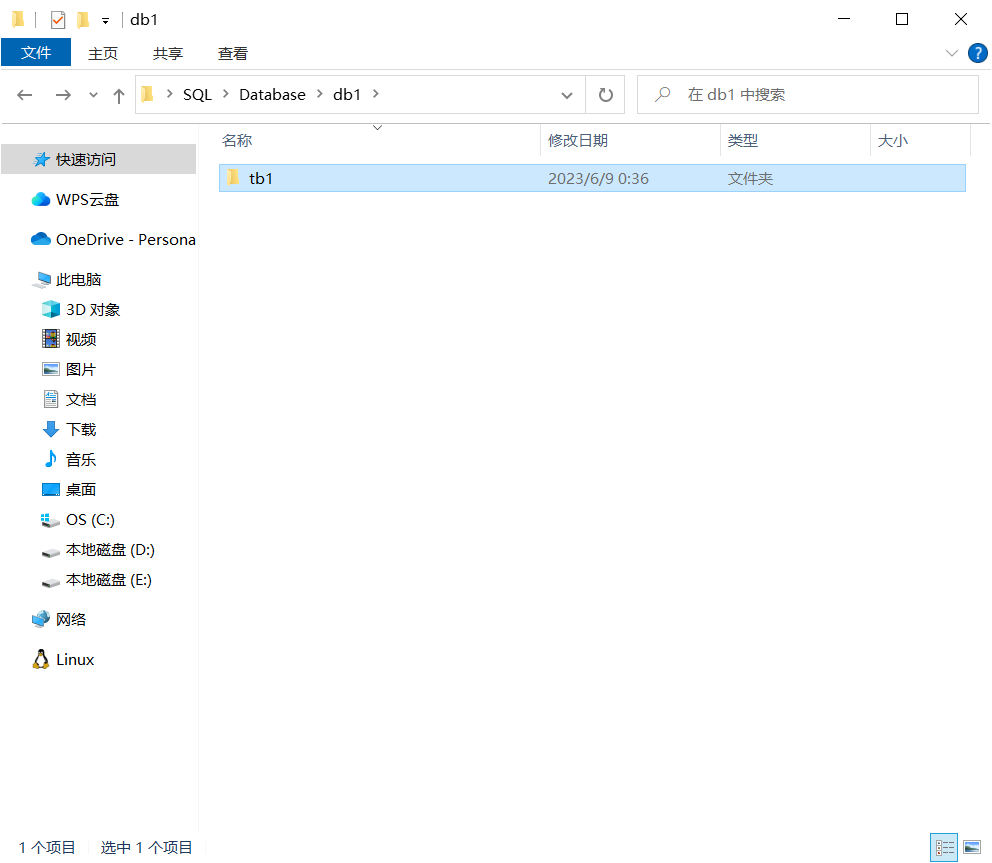
程序将在关系模式创建时隐式的为每个属性使用主码作为唯一化属性创建B+树索引,其插入,删除和更新也是随着相关SQL指令的执行而隐式执行的。

1. 功能测试和效率分析
2. 功能测试

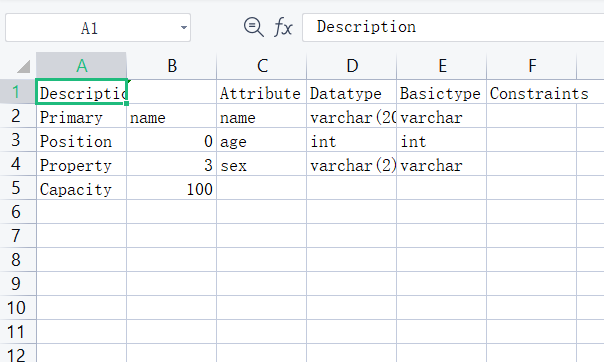
1)创建数据库和关系模式(Create)



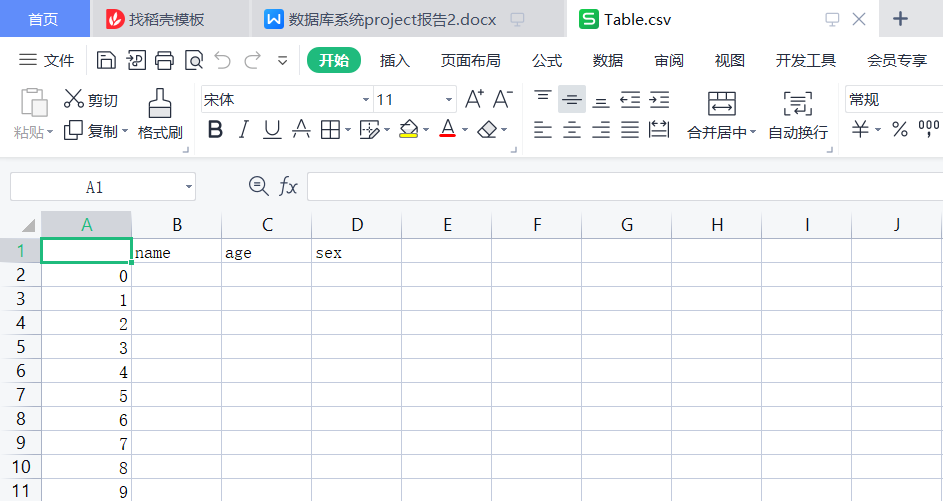
**图2.1.1 测试代码**



**图2.1.2 创建的数据库和关系模式**

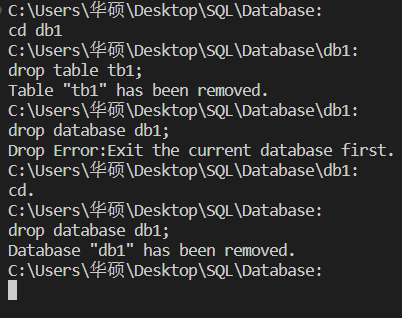


**图2.1.3 描述关系模式的元数据**



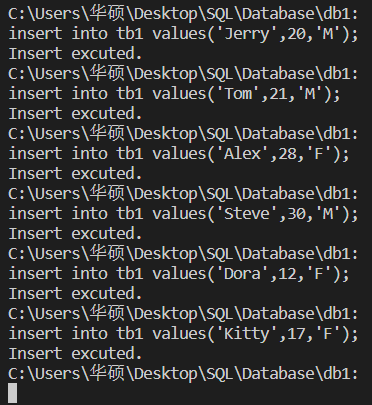
**图2.1.4 创建的关系模式**

2)删除数据库和关系模式(Drop)

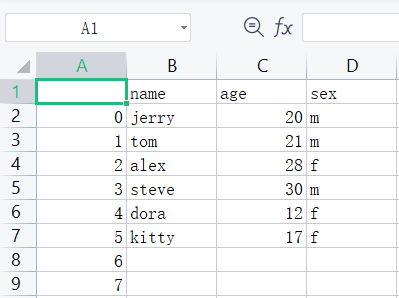


**图2.2 测试代码**

3)插入(Insert)

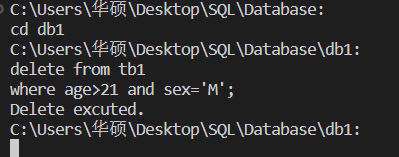


**图2.3.1 测试代码**

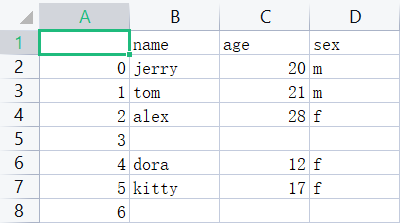


**图2.3.2 插入元组后的关系模式**

4)删除(Delete)

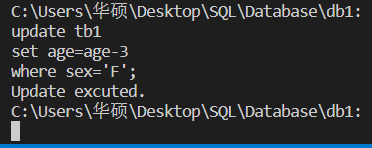


**图2.4.1 测试代码**

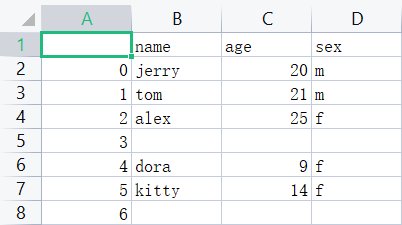


**图2.4.2 删除指定元组后的关系模式**

1. 更新(Update)

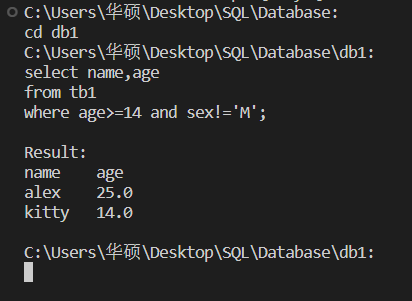


**图2.5.1 测试代码**



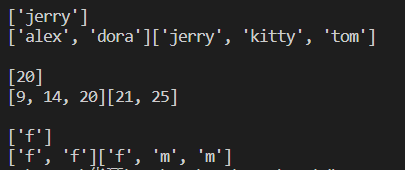
**图2.5.2 更新指定元组后的关系模式**

1. 查询(Select)



**图2.6 测试代码和查询结果**

1. 索引(Index)



**图2.7 前述各操作后的B+树**

1. 效率分析

效率分析主要分为B+树索引上的查询和更新效率分析和返回结果在关系模式上的检索和数据处理效率分析两个部分。

1. 索引查询和更新效率

典型的B+树节点规模被选为和磁盘块的规模一样大。实验中预设的B+树参数n=4,总容量为100,等价为25个磁盘块。在n=100的情况下,假设有10^6个搜索码值,执行一次查找只需要访问个节点。因此,最多只需要读取4个磁盘块。相同情况下的平衡二叉树大约需要访问20个节点,可以明显看出B+树的优势所在。并且,由于树中的根节点通常被频繁访问,很可能已经处于缓冲区中,因此通常只需要读取3个或者更少的块。

在遍历到叶子层并检索到目标搜索码值之前,在节点内部的搜索码值检索以二分方式进行,二分函数lower\_bound将以单次的代价高效的查询目标搜索码值或者相应的分割搜索码值。

在遍历到叶子层并检索到目标搜索码值之后,对于辅助索引,这些记录可能位于不同的块上,在最坏情况下可能导致M次(M为答案集规模)随机I/O操作;对于聚集索引,这些记录将在连续的块中,且每个块包含多条记录,因此范围查询的成本与答案的规模基本呈正比关系,因而可以显著降低成本。

实验中实现的B+树消除了重复搜索码值,因此插入和删除操作的代价与B+树高度呈正比,在最坏情况下的复杂度也与呈正比(N为被索引文件中的记录数量)。虽然B+树上的插入和删除操作(更新可以被拆分为旧索引的删除和新索引的插入)较为复杂,但所需的I/O操作相对较少。这是很重要的优点,因为I/O操作非常昂贵。

1. 关系模式检索和数据处理效率

本次实验主要实现的是and关键词连接而成where语句表达式,对于每个单独的谓词在索引文件上的查询结果,最终的结果集为它们的交集,因此总的复杂度约为),其中M为谓词数,N为单次查询结果的规模(对于仅含or关键词的表达式,其实现方式和复杂度是类似的;or,and以及其他部分运算符组合成的where语句表达式代价则需要额外考虑运算符优先级的判断。通过栈结构实现的这类判断的时间复杂度基本上是线性的)。

对于插入和删除操作,Empty文件的内容是用数据结构Set(集合,底层为二叉树)进行维护的。在实际的硬件中应该可以通过将有效位(Valid Bit)置0来简单的表示删除,而不必真的将对应位置的数据清除(置0)。本次实验以模拟为主,故不作进一步考虑。

四、团队协作

1. 小组分工

2.