数据结构课程设计

名称:图书管理系统

作者:白忠魏

班级:计科3

学院:信息工程学院

学号:200805030326

1. **需求分析**

完成简单的图书管理业务

1)新书入库:登记新书的编号.书名.作者和数量

2)书目信息维护 : 删除,更新

3)读者信息维护:新增,删除读者

4)查询

5)借阅,归还

语音提示功能

1. 用户进行操作时语音提示.提高交互性

用户界面

1. 美观,实用
2. **功能实现**
3. 数据库(c++实现) 关系类数据库
4. 索引----B树
5. 查询方式----sql
6. 存储方式---顺序+索引

2)语音功能—(c# mstts)

3)界面(c#)

1. **概要设计**

**文档&技术**

1. **B树--**

即二叉搜索树：

1.所有非叶子结点至多拥有两个儿子（Left和Right）；

2.所有结点存储一个关键字；

3.非叶子结点的左指针指向小于其关键字的子树，右指针指向大于其关键字的子树；

在项目中用做数据库的索引

1. **Sql--**

SQL是高级的非过程化编程语言，允许用户在高层数据结构上工作。他不要求用户指定对数据的存放方法，也不需要用户了解具体的数据存放方式，所以具有完全 不同底层结构的不同数据库系统可以使用相同的SQL语言作为数据输入与管理的接口。它以记录集合作为操纵对象，所有SQL语句接受集合作为输入，返回集合 作为输出，这种集合特性允许一条SQL语句的输出作为另一条SQL语句的输入，所以SQL语言可以嵌套，这使他具有极大的灵活性和强大的功能，在多数情况 下，在其他语言中需要一大段程序实现的一个单独事件只需要一个SQL语句就可以达到目的，这也意味着用SQL语言可以写出非常复杂的语句

在项目中为用户提供最简单的sql语句

1. **TTS –**

TTS是语音合成应用的一种，它将储存于电脑中的文件，如帮助文件或者网页，转换成自然语音输出。TTS可以帮助有视觉障碍的人阅读计算机上的信息，或者 只是简单的用来增加文本文档的可读性。现在的TTL应用包括语音驱动的邮件以及声音敏感系统。TTS经常与声音识别程序一起使用。现在有很多TTS的产 品，包括Read Please 2000， Proverbe Speech Unit，以及Next Up Technology的TextAloud。朗讯、 Elan、以及 AT&T都有自己的语音合成产品。

本项目中作为发声引擎

四**. 详细设计**

**程序流图**

****

**图形界面**

1. 登陆

需提供用户名和密码 默认为admin admin

密码用md5+乱序的方式存放于DB/PWD.mdb(文本文件)

//混合密码

public static string GetMd6Str(string ConvertString)

{

MD5CryptoServiceProvider md5 = new MD5CryptoServiceProvider();

string t2 = BitConverter.ToString(md5.ComputeHash(UTF8Encoding.Default.GetBytes(ConvertString)), 4, 8).Replace("-", "B");

t2 += BitConverter.ToString(md5.ComputeHash(UTF8Encoding.Default.GetBytes(ConvertString)), 4, 8).Replace("-","Z");

t2 += BitConverter.ToString(md5.ComputeHash(UTF8Encoding.Default.GetBytes(ConvertString)), 4, 8).Replace("-", "W");

return t2;

}

**语音提示**

使用微软的TTS引擎

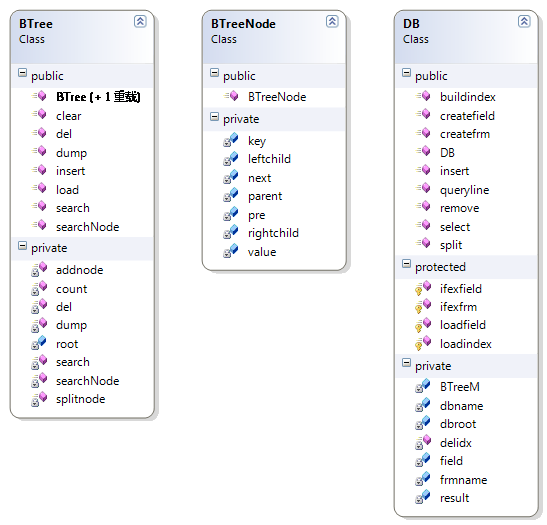
语言程序安装包带有TTS5.1..

优先使用windows7 vista 自带的TTS5.3

(XP不支持LILI语音库..故会使用SAM)

封装成DLL了 namespace TTS; 静态类speaker.speak(string thins);

**核心--数据库系统的类图**

**** 说明: 索引采用B树..动态阶数..除根节点外每层的最大key为2/m+1

数据库系统有个public method query(std::string query)用于执行简单的sql语句

|  |  |  |
| --- | --- | --- |
| 语句 | 功能 | 例子 |
| Create |  |  |
| ----database | 创建数据库 | create database std |
| ----table | 创建表  和字段数据 | create table `book` (`id` key,`booktitle` text,`publisher` text,`type` text,`writer` text,`maxnums` int,`lendnums` int) |
| Use | 选择数据库 | use sda |
| Insert | 插入数据 | insert into `book` values (`3`,`电子`,`电子工业`,`null`,`null`,`5`,`3`) |
| Delete | 删除数据 | delete from `book` where `booktitle`='汇编' |
| Select | 查找数据 | select from `book` where `id`<='7' |
| Update | 更新数据 |  |

数据库系统代码行约3000,update功能暂未实现..可以先delete然后再insert..

查询时给入相应的sql语句

查询到的结果会存入result这个迭代器

外部封装成DLL,可调用参数

fetchline() 格式化结果,取一行

query(string line) 执行sql

String^ get(String^ tablename) 取当前行的字段名下的字段值

注:命名空间 DBcore

数据库文件物理系统

**DB**

**│**

**├─book**

**│ data.dbs**

**│ id.idx**

**│**

**├─lendinfo**

**│ data.dbs**

**│ id.idx**

**│**

**└─reader**

**data.dbs**

**sid.idx**

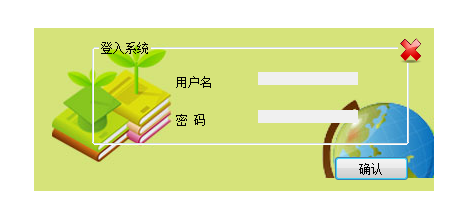
idx文件就是存放的B树索引文件

dbs是数据库主文件

当通过索引操作时读取索引文件进B树..其他情况下均顺序操作

五.**调试与运行**

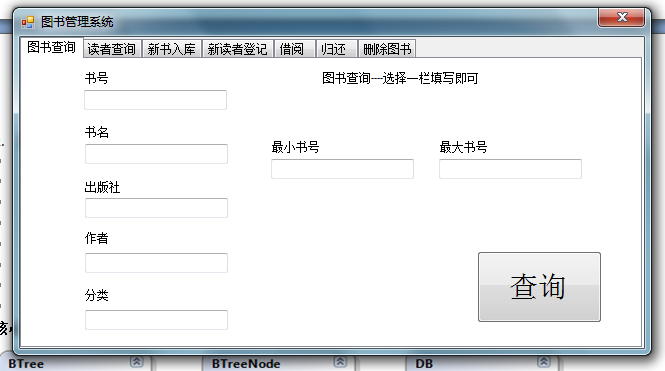
语音提示输入



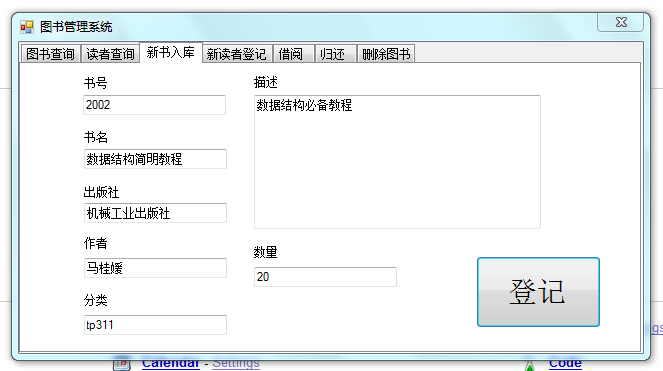
登陆界面



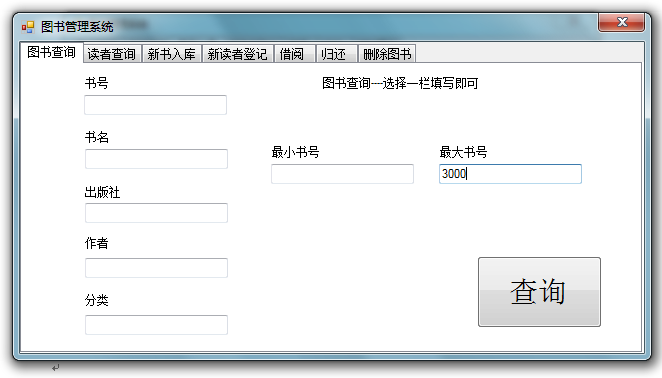
登陆成功



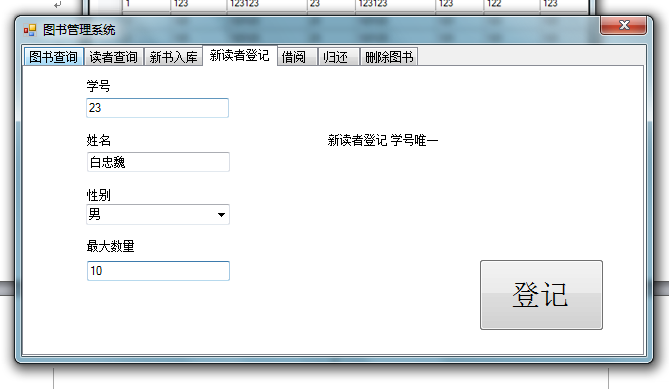
主界面可进行图书查询,读者查询等功能的应用



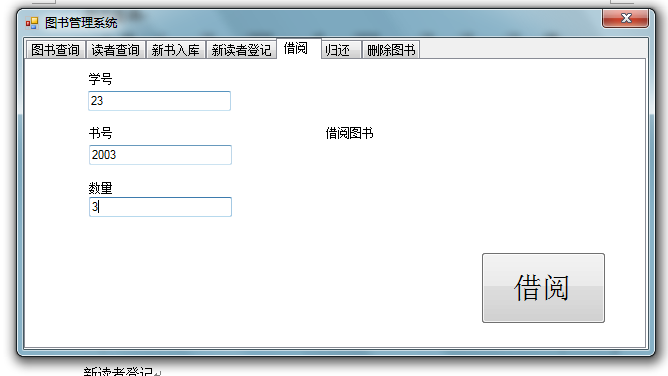
新书入库



图书查询..可按任意一项查询..书号唯一



新读者登记

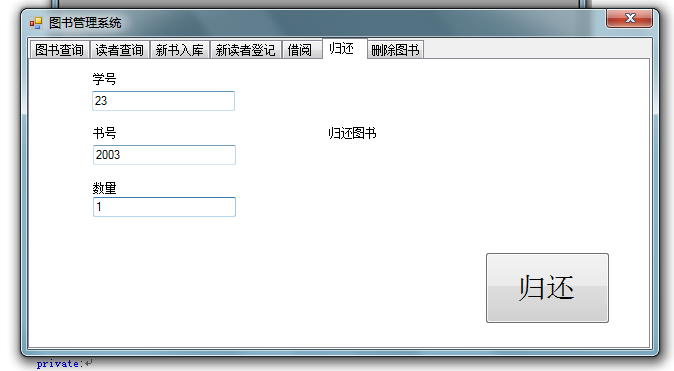


借阅3本





结果..库存减少..拥有数增加



归还图书



**-----索引BTREE.h**

class BTree{

private:

/\*\*结果迭代器

\*用于保存临时结果

\*@access private

\*/

vector<string> k;

/\*\*根节点

\*@access private

\*/

BTreeNode\* root;

/\*\*添加节点

\*@access private

\*@param BTreeNode\* p 节点

\*@param string value 节点值

\*@param int key 节点key

\*@param int m 层级默认为

\*/

BTreeNode\* addnode(BTreeNode\* p,string value,int key,int m=1);

/\*\*遍历p节点及其子节点..输出至文件

\*@param ofstream\* fout

\*@param BTreeNode\* p 节点

\*/

int dump(ofstream\* fout,BTreeNode\* p);

/\*\*删除这个结点

\*不删除节点空间

\*@param BTreeNode\* p 节点

\*return BTreeNode\*

\*/

BTreeNode\* del(BTreeNode\* p);

/\*\*搜索B树

\*@param BTreeNode\* p 节点

\*@param int key 节点索引key值

\*@return BTreeNode\*

\*/

BTreeNode\* searchNode(BTreeNode\* p,int key);

/\*\*计算p节点所在层级的节点数量

\*@param BTreeNode\* p 节点

\*@return int 节点数量

\*/

int count(BTreeNode\* p);

/\*\*分离p节点所在层级的所有节点..层级为m

\*@param BTreeNode\* p 节点

\*@param int m 节点层级

\*@return void

\*/

void splitnode(BTreeNode\* p,int m);

/\*\*遍历树查找key=的节点

\*@return value

\*/

string BTree::search(BTreeNode\* p,int key,string pit="\0");

/\*\*遍历节点所在层级及其子节点找寻比它小的节点

\*@return string

\*/

string BTree::tovector\_le(BTreeNode\* p,int max\_key,int m=0);

/\*\*搜索B树中所有key比maxkey小的节点

\*@param BTreeNode\* p 节点

\*@param int key 节点索引key值

\*@return int

\*/

int BTree::searchNode\_le(int maxkey,BTreeNode\* w,int dep=0);

/\*\*遍历节点所在层级及其子节点找寻比它大的节点

\*@return string

\*/

string BTree::tovector\_gt(BTreeNode\* p,int min\_key,int m=0);

/\*\*搜索B树中所有key比minkey大的节点

\*@param BTreeNode\* p 节点

\*@param int key 节点索引key值

\*@return int

\*/

int BTree::searchNode\_gt(int minkey,BTreeNode\* w,int dep=0);

public:

/\*\*空构造函数

\*/

BTree();

/\*\*遍历树查找key=的节点

\*@return BTreeNode\*

\*/

BTreeNode\* BTree::searchNode(int key);

/\*\*默认构造函数

\*@param string rootvalue 根节点值

\*@param int rootkey 根节点索引key值

\*/

BTree(string rootvalue,int rootkey);

/\*\*插入新节点到B树

\*@param string value 节点值

\*@param int key 节点索引key值

\*@return BTreeNode\*

\*/

BTreeNode\* insert(string value,int key);

/\*\*遍历节点下子节点..清空value相等的子节点..和节点的空间

\*@param BTreeNode\* p 节点

\*/

bool del(int key);

/\*\*遍历整个B树..输出至文件

\*@param ofstream\* fout

\*/

int dump(ofstream\* fout);

/\*\*读取文件..读取前不会清空当前节点

\*@param ifstream\* fin

\*/

int load(ifstream\* fin);

/\*\*清空整个B树

\*/

void clear();

/\*\*遍历树查找key=的节点

\*@return value

\*/

string BTree::search(int key){

return search(this->root,key);

}

/\*\*遍历树查找key<maxkey的节点

\*@return vector<string>

\*/

vector<string> BTree::searchNode\_le(int maxkey,int dep=0);

/\*\*遍历树查找key>minkey的节点

\*@return vector<string>

\*/

vector<string> BTree::searchNode\_gt(int minkey,int dep=0);

};

**数据库操作—DBcore.h**

class DB

{

friend ref class DBcore;

private:

//数据库根目录

string dbroot;

//表名

string frmname;

//B树迭代器

vector<BTree\*> BTreeM;

//字段名

vector<string> field;

//字段属性

vector<string> fieldtype;

//数据库名

string dbname;

protected:

/\*\*删除索引

\*@param int Bpos

\*@param string value

\*/

void delidx(int Bpos,string value);

/\*\*将value以s分割

\*@param string value

\*@param char s

\*@return vector<string>

\*/

vector<string> split(string value,char s);

/\*\*建立表

\*@param string db\_name 数据库名

\*/

void createfrm(string frm\_name);

/\*\*创建字段

\*@param string frmname 表名

\*@vector<string> 字段名

\*/

void createfield(string frmname,vector<string> tb);

/\*\*找寻结果

\*@param string frmname 表名

\*@param string tdname 字段名

\*@param string value

\*@return vector<string>\* 找到的值的key

\*/

vector<string>\* select(string frmname,string tdname,string value,int y=-1);

/\*\*找寻比maxvalue小的结果

\*@param string frmname 表名

\*@param string tdname 字段名

\*@param string maxvalue 最大值

\*@return vector<string>\* 找到的值的key

\*/

vector<string>\* selectls(string frmname,string tdname,string maxvalue,int y=-1);

/\*\*找寻比maxvalue小的结果并删除

\*@param string frmname 表名

\*@param string tdname 字段名

\*@param string maxvalue 最大值

\*@return vector<string>\* 找到的值的key

\*/

vector<string>\* removels(string frmname,string tdname,string maxvalue,int y=-1);

/\*\*找寻比minvalue大的结果并删除

\*@param string frmname 表名

\*@param string tdname 字段名

\*@param string maxvalue 最大值

\*@return vector<string>\* 找到的值的key

\*/

vector<string>\* removegt(string frmname,string tdname,string minvalue,int y=-1);

/\*\*找寻比minvalue大的结果

\*@param string frmname 表名

\*@param string tdname 字段名

\*@param string maxvalue 最大值

\*@return vector<string>\* 找到的值的key

\*/

vector<string>\* selectgt(string frmname,string tdname,string minvalue,int y=-1);

/\*\*将result中对应的删除

\*@param string frmname 表名

\*@param string tdname 字段名

\*/

void removeall(string frmname,string tdname,vector<string> valueAl);

/\*\*找寻结果存入result中

\*@param string frmname 表名

\*@param string tdname 字段名

\*@param string maxvalue 最大值

\*/

vector<string>\* selectall(string frmname,string tdname,vector<string> valueAl);

/\*\*读取索引

\*@param string index\_filename 索引文件

\*@param int i 序号

\*/

void loadindex(string index\_filename,int i);

/\*\*表是否存在

\*@return bool

\*/

bool ifexfrm();

/\*\*读取字段

\*@param string frmname 表名

\*@return bool

\*/

bool loadfield(string frmname);

/\*\*是否存在字段

\*@return bool

\*/

bool ifexfield();

/\*\*删除与值相等的数据

\*@param string frmname 表名

\*@param string tdname 字段名

\*@param string value 值

\*/

vector<string>\* remove(string frmname,string tdname,string value,int y=-1);

/\*\*插入数据

\*@param string frmname 表名

\*@vector<string> row 字段名

\*/

void insert(string frmname,vector<string> row);

public:

/\*\*初始化根节点

\*/

DB(string dbroot);

/\*\*结果迭代器

@access public

\*/

vector<string>\* result;

/\*\*分析并执行简单的单行sql语句

\*@param string queryline

\*/

void queryline(string queryline);

};

**}**

**附录…其他代码..**

**数据库BTree.cpp**

//B树主操作cpp文件

//

//by baizhongwei

//blog http://haku.webcpp.net

#include "stdafx.h"

#include "BTreeM.h"

int BTree::count(BTreeNode\* p)

{

BTreeNode\* p2=p;

//先往前计算

int p\_i=0;

while(p2->pre)

{

p\_i++;

p2=p2->pre;

}

//往后计算

int n\_i=0;

while(p->next)

{

n\_i++;

p=p->next;

}

return p\_i+n\_i+1;

}

BTree::BTree(){

this->root=NULL;

}

BTree::BTree(string rootvalue,int rootkey){

this->root=new BTreeNode(rootvalue,rootkey);

}

BTreeNode\* BTree::del(BTreeNode\* p){

if(p==NULL) return 0;

int up=0;

//如果右孩子为空且左孩子为空.直接删除

if(p->leftchild==NULL&&p->rightchild==NULL)

{

int now=p->key;

if(p->parent!=NULL)

{

BTreeNode\* h=p->parent;

while(h)

{

//如果key大于now

//40 +42+

// 41

if(h->key>now&&h->leftchild==p)

{

h->leftchild=p->next;

if(h->pre)h->pre->rightchild=p->next;

up=1;

break;

}

//游标下移

if(h->next)

{

h=h->next;

}

else

{

break;

}

}

if(up!=1&&h->rightchild==p)

{

h->rightchild=p->next;

if(h->next)h->next->leftchild=p->next;

}

}

if(p->parent==NULL)

{

this->root=NULL;

if(p->pre)

{

this->root=p->pre;

}

if(p->next)

{

this->root=p->next;

}

}

//如果存在前续..跳过这个结点.

if(p->pre)

{

p->pre->next=p->next;

}

//如果存在后续..跳过这个结点.

if(p->next)

{

p->next->pre=p->pre;

}

return p;

}

//如果左孩子不为空且右孩子为空..左子支上移..然后分裂之

if(p->leftchild!=NULL&&p->rightchild==NULL)

{

BTreeNode\* r=p->leftchild;

int now=p->key;

r->parent=p->parent;

while(r->next)

{

r->parent=p->parent;

r=r->next;

}

r->parent=p->parent;

if(p->parent!=NULL)

{

BTreeNode\* h=p->parent;

while(h)

{

//如果key大于now

//40 +42+

// 41

if(h->key>now&&h->leftchild==p)

{

h->leftchild=p->next;

if(h->pre)h->pre->rightchild=p->next;

up=1;

break;

}

//游标下移

if(h->next)

{

h=h->next;

}

else

{

break;

}

}

if(up!=1&&h->rightchild==p)

{

h->rightchild=p->next;

if(h->next)h->next->leftchild=p->next;

}

}

if(p->parent==NULL)

{

this->root=NULL;

if(p->pre)

{

this->root=p->pre;

}

if(p->next)

{

this->root=p->next;

}

}

//如果存在前续..跳过这个结点.

if(p->pre)

{

p->pre->next=p->leftchild;

p->pre->rightchild=p->leftchild->leftchild;

}

//如果存在后续..跳过这个结点.

if(p->next)

{

p->next->pre=r;

p->next->leftchild=r->rightchild;

}

if(p->parent==NULL)

{

this->root=p->leftchild;

splitnode(this->root,1);

}

return p;

}

//如果右孩子不为空且左孩子为空..右子支上移..然后分裂之

if(p->leftchild==NULL&&p->rightchild!=NULL)

{

BTreeNode\* r=p->rightchild;

int now=p->key;

//设置父节点

r->parent=p->parent;

while(r->next)

{

r->parent=p->parent;

r=r->next;

}

r->parent=p->parent;

if(p->parent!=NULL)

{

BTreeNode\* h=p->parent;

while(h)

{

//如果key大于now

//40 +42+

// 41

if(h->key>now&&h->leftchild==p)

{

h->leftchild=p->next;

if(h->pre)h->pre->rightchild=p->next;

up=1;

break;

}

//游标下移

if(h->next)

{

h=h->next;

}

else

{

break;

}

}

if(up!=1&&h->rightchild==p)

{

h->rightchild=p->next;

if(h->next)h->next->leftchild=p->next;

}

}

if(p->parent==NULL)

{

this->root=NULL;

}

//如果存在前续..跳过这个结点.

if(p->pre)

{

p->pre->next=p->rightchild;

p->pre->rightchild=p->rightchild->leftchild;

}

//如果存在后续..跳过这个结点.

if(p->next)

{

p->next->pre=r;

p->next->leftchild=r->rightchild;

}

if(p->parent==NULL)

{

this->root=p->leftchild;

splitnode(this->root,1);

}

if(p->parent==NULL)

{

this->root=p->rightchild;

splitnode(this->root,1);

}

return p;

}

//如果右孩子不为空且左孩子不为空..进入左子支的最后一个最大的递归

if(p->leftchild!=NULL&&p->rightchild!=NULL)

{

BTreeNode\* r=p->leftchild;

BTreeNode\* w=NULL;

while(r->next)

{

r=r->next;

}

while(r->rightchild!=NULL)

{

while(r->next)

{

r=r->next;

}

if(r->rightchild!=NULL)break;

r=r->rightchild;

}

w=r;

if(r->rightchild)r=r->rightchild;

//转移节点

if(r->pre)r->pre->next=NULL;

//父级删除之

if(w->rightchild==r){w->rightchild=NULL;}

//复制顶节点的属性

if(p->leftchild!=r)r->leftchild=p->leftchild;

r->rightchild=p->rightchild;

r->pre=p->pre;

r->next=p->next;

r->parent=p->parent;

if(p->next)p->next->pre=r;

if(p->pre)p->pre->next=r;

if(p->leftchild&&p->leftchild!=r)

{

BTreeNode\* s=p->leftchild;

while(s)

{

s->parent=r;

s=s->next;

}

}

if(p->rightchild){

BTreeNode\* s=p->rightchild;

while(s)

{

s->parent=r;

s=s->next;

}

}

if(p->parent==NULL||p==root){

this->root=r;

return p;

}

if(p->parent->leftchild==p)

{

p->parent->leftchild=r;

}

if(p->parent->rightchild==p)

{

p->parent->rightchild=r;

}

return p;

}

return NULL;

}

bool BTree::del(int key){

BTreeNode\* p=searchNode(root,key);

if(p==NULL) return 0;

/\*p->parent=new BTreeNode("321",5);

this->root=p->parent;\*/

p=del(p);

if(p!=NULL)delete p;

return 1;

}

BTreeNode\* BTree::insert(string value,int key)

{

if(this->root==NULL)

{

this->root=new BTreeNode(value,key);

return this->root;

}

return addnode(root,value,key);

}

void BTree::splitnode(BTreeNode\* p,int m)

{

if(m==1&&count(p)>2)

{

//cout<<"所在是根节点..大于..分离节点"<<endl;

//找到处于中间的节点

while(p->pre)p=p->pre;

BTreeNode\* p\_mid=p->next;

BTreeNode\* p\_r=p->next->next;

p\_mid->leftchild=p;

p\_mid->rightchild=p\_r;

p\_mid->pre=NULL;

p\_mid->next=NULL;

//左节点清理

p->parent=p\_mid;

p->next=NULL;

BTreeNode\* vec=NULL;

vec=p->leftchild;

while(vec)

{

vec->parent=p;

vec=vec->next;

}

vec=p->rightchild;

while(vec)

{

vec->parent=p;

vec=vec->next;

}

//右节点清理

p\_r->parent=p\_mid;

p\_r->pre=NULL;

vec=p\_r->leftchild;

while(vec)

{

vec->parent=p\_r;

vec=vec->next;

}

vec=p\_r->rightchild;

while(vec)

{

vec->parent=p\_r;

vec=vec->next;

}

//新根节点

this->root=p\_mid;

}

else{

if(m!=1&&count(p)>m){

//cout<<m<<"级节点过多..分离节点"<<endl;

//cout<<"取中间节点..位置:"<<m/2<<endl;

//标记p

int i=0;

//左节点

while(p->pre)p=p->pre;

BTreeNode\* p\_mid=p;

//中间节点

while(i++<m/2&&p\_mid->next)

{

//cout<<i<<endl;

p\_mid=p\_mid->next;

}

BTreeNode\* p\_r=p\_mid->next;

//左节点清理

//更新父节点

BTreeNode\* pt=p;

while(pt->next!=NULL&&pt!=p\_mid)

{

pt->parent=p\_mid;

pt=pt->next;

}

p\_mid->pre->next=NULL;

//右节点清理

//更新父节点

pt=p\_mid->next;

while(pt!=NULL)

{

pt->parent=p\_mid;

pt=pt->next;

}

if(p\_r!=NULL)p\_r->pre=NULL;

//中间节点清理

p\_mid->leftchild=p;

p\_mid->rightchild=p\_r;

p\_mid->pre=NULL;

p\_mid->next=NULL;

//修改中间节点的父节点

BTreeNode\* p\_p=p\_mid->parent;

/\*这里可能有问题..待检测\*/

p\_mid->parent=p\_p->parent;

//在上层寻找p应该插入的位置

int key\_p=0;

i=0;

while(p\_p)

{

key\_p=p\_p->key;

//如果第一个就比key大..则直接插入

if(i==0&&key\_p>p\_mid->key)

{

//cout<<key\_p<<endl;

p\_mid->next=p\_p;

p\_p->pre=p\_mid;

p\_p->leftchild=p\_mid->rightchild;

if(p\_p==this->root)this->root=p\_mid;

splitnode(p\_p,m-1);

break;

}

//如果所在节点的key比中间节点key小

//则在所在key后插入中间节点

if(key\_p<p\_mid->key)

{

//cout<<key\_p<<endl;

BTreeNode\* p\_pn=p\_p->next;

p\_p->next=p\_mid;

p\_mid->pre=p\_p;

p\_p->rightchild=p\_mid->leftchild;

p\_mid->leftchild->parent=p\_p;

if(p\_pn){

p\_pn->leftchild=p\_mid->rightchild;

p\_mid->next=p\_pn;

p\_pn->pre=p\_mid;

p\_mid->rightchild->parent=p\_pn;

p\_mid=p\_pn;

}

//cout<<m<<endl;

splitnode(p\_p,m-1);

break;

}

p\_p=p\_p->next;

i++;

}

}

}

}

BTreeNode\* BTree::addnode(BTreeNode\* p,string value,int key,int m){

//如果p是空节点..返回空

if(p==NULL)

{

return p;

}

int now\_key=p->key;

//如果相等..则返回空

if(now\_key==key)

{

//throw exception("已经存在此节点");

return NULL;

}

else{

//如果当前节点值比key大.且左节点不为空.去找当前节点左子支

if(now\_key>key&&p->leftchild!=NULL)

{

return addnode(p->leftchild,value,key,m+1);

}

//如果当前节点值比key大.且左节点为空.在这个结点前加入节点且

if(now\_key>key&&p->leftchild==NULL)

{

/\*\*cout<<"节点key:"<<key<<" 加入层级:"<<m<<endl;\*/

BTreeNode\* i=new BTreeNode(value,key);

BTreeNode\* p\_h=p->pre;

p->pre=i;

i->next=p;

i->pre=p\_h;

if(p\_h!=NULL)p\_h->next=i;

i->rightchild=p->leftchild;

i->parent=p->parent;

//把父节点的下一级设为新的节点

if(p->parent!=NULL&&p->parent->leftchild==p){

p->parent->leftchild=i;

}

else if(p->parent!=NULL&&p->parent->rightchild==p)

{

p->parent->rightchild=i;

}

//如果是根节点..转换之

if(p==this->root)

{

this->root=i;

}

//分离节点所在层

splitnode(i,m);

}

//如果当前节点值比key小.且右子支不为空.去找当前节点右子支

if(now\_key<key&&p->rightchild!=NULL)

{

//如果这个结点的右子支的key比右边节点key小..去右边节点寻找

if(p->next!=NULL&&p->rightchild->key<p->next->key)

{

return addnode(p->next,value,key,m);

}

return addnode(p->rightchild,value,key,m+1);

}

//如果当前节点值比key小.且右子支为空.去找当前节点后继

if(now\_key<key&&p->rightchild==NULL)

{

if(p->next!=NULL)

{

return addnode(p->next,value,key,m);

}

else

{

/\*\*cout<<"节点key:"<<key<<" 加入层级:"<<m<<endl;\*/

BTreeNode\* i=new BTreeNode(value,key);

p->next=i;

i->pre=p;

i->leftchild=p->rightchild;

i->parent=p->parent;

//分离节点所在层

splitnode(i,m);

}

}

}

return p;

}

BTreeNode\* BTree::searchNode(int key)

{

return searchNode(this->root,key);

}

BTreeNode\* BTree::searchNode(BTreeNode\* p,int key){

//如果p是空节点..返回空

if(p==NULL)

{

return NULL;

}

int now\_key=p->key;

//如果相等..则返回空

if(now\_key==key)

{

//throw exception("已经存在此节点");

return p;

}

else{

//如果当前节点值比key大.且左节点不为空.去找当前节点左子支

if(now\_key>key&&p->leftchild!=NULL)

{

return searchNode(p->leftchild,key);

}

//如果当前节点值比key大.且左节点为空.返回空

if(now\_key>key&&p->leftchild==NULL)

{

return NULL;

}

//如果当前节点值比key小.且右子支不为空.去找当前节点右子支

if(now\_key<key&&p->rightchild!=NULL)

{

//如果这个结点的右子支的key比右边节点key小..去右边节点寻找

if(p->next!=NULL&&p->rightchild->key<p->next->key)

{

return searchNode(p->next,key);

}

return searchNode(p->rightchild,key);

}

//如果当前节点值比key小.且右子支为空.去找当前节点后继

if(now\_key<key&&p->rightchild==NULL)

{

if(p->next!=NULL)

{

return searchNode(p->next,key);

}

else

{

return NULL;

}

}

}

return p;

}

int BTree::dump(ofstream\* fout)

{

return dump(fout,this->root);

}

int BTree::dump(ofstream\* fout,BTreeNode\* p){

if(p==NULL) return 0;

fout->write(p->value.c\_str(),p->value.length());

fout->write("\x0B",1);

stringstream s;

s<<p->key;

fout->write(s.str().c\_str(),s.str().length());

fout->write("\x7E",1);

if(p->pre==NULL)dump(fout,p->leftchild);

dump(fout,p->rightchild);

dump(fout,p->next);

return 1;

}

int BTree::load(ifstream\* fin){

char ch;

stringstream s;

string value="";

string key="";

int y=0;

if(fin->fail())return 0;

while(!fin->eof()) {

ch = fin->get();

s<<ch;

if (ch == '\x0B')

{

//cout<<s.str()<<endl;

//清空s,,并保存到value

value=s.str().substr(0,s.str().length()-1);

s.str("");

}

if (ch == '\x7E')

{

y++;

//cout<<s<<endl;

key=s.str().substr(0,s.str().length()-1);

this->insert(value,(int)atof(key.c\_str()));

//清空s

s.str("");

}

}

return y;

}

void BTree::clear()

{

//del(this->root);

this->root=NULL;

}

string BTree::search(BTreeNode\* p,int key,string pit)

{

if(p==NULL) return "\0";

if(p->pre==NULL){

pit=search(p->leftchild,key);

if(pit!="\0")return pit;

}

pit=search(p->rightchild,key);

if(pit!="\0")return pit;

pit=search(p->next,key);

if(pit!="\0")return pit;

if(p->key==key)return p->value;

return "\0";

}

string BTree::tovector\_le(BTreeNode\* p,int max\_key,int m){

if(p==NULL)return " ";

if(p->pre==NULL)tovector\_le(p->leftchild,max\_key,m+1);

tovector\_le(p->rightchild,max\_key,m+1);

tovector\_le(p->next,max\_key,m+1);

if(p->key<max\_key)

{

//cout<<p->key<<endl;

stringstream ss;

ss<<p->key;

this->k.push\_back(ss.str());

ss.str("");

}

return " ";

}

int BTree::searchNode\_le(int maxkey,BTreeNode\* w,int dep){

if(dep==0)k.clear();

BTreeNode\* p=NULL;

//如果存在w节点(从上一级递归下来的)切换到p节点

if(w!=NULL)

{

p=w;

}

else

{

p=searchNode(this->root,maxkey);

}

//上级节点不为空且上级节点的key比它小

if(p->parent!=NULL&&p->parent->key<maxkey)

{

return searchNode\_le(maxkey,p->parent,1);

}

//前序节点不为空且前序节点的key比它小

if(p->pre!=NULL&&p->pre->key<maxkey)

{

return searchNode\_le(maxkey,p->pre,1);

}

//从节点开始遍历那层的节点

this->tovector\_le(p,maxkey);

return 1;

}

string BTree::tovector\_gt(BTreeNode\* p,int min\_key,int m){

if(p==NULL)return " ";

if(p->pre==NULL)tovector\_gt(p->leftchild,min\_key,m+1);

tovector\_gt(p->rightchild,min\_key,m+1);

tovector\_gt(p->next,min\_key,m+1);

if(p->key>min\_key)

{

//cout<<p->key<<endl;

stringstream ss;

ss<<p->key;

this->k.push\_back(ss.str());

ss.str("");

}

return " ";

}

int BTree::searchNode\_gt(int minkey,BTreeNode\* w,int dep){

if(dep==0)k.clear();

BTreeNode\* p=NULL;

//如果存在w节点(从上一级递归下来的)切换到p节点

if(w!=NULL)

{

p=w;

}

else

{

p=searchNode(this->root,minkey);

}

//上级节点不为空且上级节点的key比它大

if(p->parent!=NULL&&p->parent->key>minkey)

{

return searchNode\_gt(minkey,p->parent,1);

}

//从节点开始遍历那层的节点

this->tovector\_gt(p,minkey);

return 1;

}

vector<string> BTree::searchNode\_le(int maxkey,int dep){

//加入临时节点

this->insert("NULL",maxkey);

this->searchNode\_le(maxkey,NULL,0);

//删除临时节点

this->del(maxkey);

return k;

}

vector<string> BTree::searchNode\_gt(int minkey,int dep){

//加入临时节点

this->insert("NULL",minkey);

this->searchNode\_gt(minkey,NULL,0);

//删除临时节点

this->del(minkey);

return k;

}

**数据库BD.cpp**

#include "stdafx.h"

#include "DB.h"

namespace DBcore{

void DB::delidx(int Bpos,string value){

if(fieldtype[Bpos]!="key")return;

int y=(int)atof(value.c\_str());

if(y==0)y=value.length();

//cout<<"删除key:"<<y<<endl;

BTreeM.at(Bpos)->del(y);

string filename=this->dbroot+"\\"+this->dbname+"\\"+this->frmname+"\\"+field[Bpos]+".idx";

ofstream\* fout=new ofstream(filename.c\_str(),ios\_base::binary);

BTreeM[Bpos]->dump(fout);

try{

fout->close();

}

catch(exception ex)

{

cout<<ex.what();

}

delete fout;

}

vector<string> DB::split(string value,char s){

vector<string> res;

int i=0;

char ch;

stringstream val;

for(i=0;(unsigned int)i<value.length();i++)

{

ch=value.at(i);

if(ch==s||i==value.length()-1)

{

res.push\_back(val.str());

val.str("");

continue;

}

val<<ch;

}

return res;

}

void DB::createfrm(string frm\_name){

this->frmname=frm\_name;

if(this->dbname=="")

{

cout<<"error:尚未选择数据库"<<endl;

return;

}

if(ifexfrm())

{

cout<<"error:表文件已经存在"<<endl;

return;

}

try{

string dirname=this->dbroot+"\\"+this->dbname+"\\\\"+this->frmname+"\\\\";

\_mkdir(dirname.c\_str());

//写入新的文件

string filename=this->dbroot+"\\"+this->dbname+"\\"+this->frmname+"\\data.dbs";

ofstream fout(filename.c\_str(),ios\_base::binary);

if(fout.fail())throw exception("create data file failed");

fout.close();

cout<<"success:表创建成功"<<endl;

}

catch(exception ex)

{

cout<<"error:表创建失败\r\n原因:"<<ex.what()<<endl;

throw exception("error error create table :");

}

}

void DB::createfield(string frmname,vector<string> tb){

this->frmname=frmname;

if(this->dbname=="")

{

cout<<"error:尚未选择数据库"<<endl;

throw exception("error no database selected");

return;

}

this->field.clear();

this->fieldtype.clear();

if(!ifexfrm())return;

//如果存在数据

if(ifexfield())return;

try{

string filename=this->dbroot+"\\"+this->dbname+"\\"+frmname+"\\"+"data.dbs";

ofstream fout(filename.c\_str(),ios\_base::binary);

int i=0;

for(i=0;i<(int)tb.size();i++)

{

//cout<<tb[i]<<endl;

fout.write(tb[i].c\_str(),tb[i].length());

field.push\_back(tb[i].substr(0,tb[i].find('|')).c\_str());

if(i!=(int)tb.size()-1)fout.write("\x0b",1);

fieldtype.push\_back(tb[i].substr(tb[i].find('|')+1,tb[i].size()-1).c\_str());

if(fieldtype[i]=="key")

{

string filename=this->dbroot+"\\"+this->dbname+"\\"+frmname+"\\"+this->field[i]+".idx";

ofstream\* fout=new ofstream(filename.c\_str(),ios\_base::binary);

fout->close();

delete fout;

}

BTreeM.push\_back(new BTree());

BTreeM[i]->clear();

}

fout.write("\x0c",1);

cout<<"success:字段创建成功"<<endl;

fout.close();

}

catch(exception ex)

{

cout<<"error:字段创建失败\r\n原因:"<<ex.what()<<endl;

throw exception("error error create fields :");

}

}

vector<string>\* DB::select(string frmname,string tdname,string value,int y){

int i=0;

if(y==-1)

{

if(this->dbname=="")

{

cout<<"error:尚未选择数据库"<<endl;

throw exception("error no database selected");

return NULL;

}

this->frmname=frmname;

loadfield(frmname);

//在索引中查询位置

y=(int)atof(value.c\_str());

}

int tablepos=-1;

//得到表对应tablepos

for(i=0;i<(int)field.size();i++){

if(field[i]==tdname)

{

tablepos=i;

break;

}

}

if(tablepos==-1)

{

cout<<"error:错误的表名"<<endl;

throw exception("error wrong table name");

return NULL;

}

if(fieldtype[tablepos]=="key")

{

string valueLe=BTreeM[tablepos]->search(y);

string fposs="";

if(valueLe=="")

{

cout<<"warning:没有找到"<<endl;

return NULL;

}

//这个迭代器valueL用来保存从索引里面得到的每个value的大小

vector<int> valueL;

stringstream valueS;

for(int i=0;(unsigned int)i<valueLe.length();i++)

{

valueS<<valueLe[i];

if(valueLe[i]=='#')

{

fposs=valueLe.substr(0,i);

valueS.str("");

}

if(valueLe[i]=='|'||i==(int)valueLe.length()-1)

{

valueL.push\_back((int)atof(valueS.str().c\_str()));

valueS.str("");

}

}

valueLe="";

//cout<<fposs;

int fpos=(int)atof(fposs.c\_str());

//cout<<fpos;

string filename=this->dbroot+"\\"+this->dbname+"\\"+this->frmname+"\\data.dbs";

ifstream fin(filename.c\_str());

//定位

fin.seekg(fpos);

//开始读取

//清空结果迭代器

this->result->clear();

char ch=' ';

stringstream s;

int ii=0;

while(!fin.eof()) {

if((int)valueL.size()==0)break;

if(valueL.at(0)==ii){

//cout<<s.str()<<endl;

this->result->push\_back(s.str());

s.str("");

//vector<int>::iterator iter = colors.begin();

valueL.erase(valueL.begin());

ii=0;

//跳过一个字符

fin.get();

}

ch = fin.get();

s<<ch;

ii++;

}

fin.close();

if(this->result->at(tablepos)==value){

cout<<"找到"<<endl;

return this->result;

}else

{

cout<<"waring:不匹配..寻找下节点"<<endl;

return select(frmname,tdname,value,y+1);

}

}

else

{

//cout<<fposs;

int fpos=0;

//cout<<fpos;

string filename=this->dbroot+"\\"+this->dbname+"\\"+this->frmname+"\\data.dbs";

ifstream fin(filename.c\_str());

//定位

fin.seekg(fpos);

//开始读取

//清空结果迭代器

this->result->clear();

char ch=' ';

stringstream s;

this->field.size();

//临时迭代器

vector<string> rs;

rs.clear();

int found=0;

int i=0;

while(!fin.eof()) {

ch = fin.get();

if(ch=='\x0B')

{

cout<<s.str()<<"--"<<value<<endl;

rs.push\_back(s.str());

if(s.str()==value)

//cout<<"wa"<<endl;

if(tablepos==i&&s.str()==value)

{

found=1;

}

i++;

s.str("");

}

else

{

if(ch=='\x0C')

{

//cout<<s.str()<<endl;

rs.push\_back(s.str());

if(tablepos==i&&s.str()==value)

{

found=1;

}

i++;

//如果成功//将临时vector加入result

if(found){

for(i=0;i<(int)rs.size();i++)

{

this->result->push\_back(rs.at(i));

}

}

found=0;

i=0;

cout<<s.str()<<endl;

s.str("");

rs.clear();

}

else

{

s<<ch;

}

}

}

fin.close();

return this->result;

}

}

void DB::queryline(string queryline)

{

stringstream action;

stringstream leftstring;

int i=0;

char ch='a';

int p=0;

for(i=0;(unsigned int)i<queryline.length();i++)

{

ch=queryline.at(i);

if(ch==' ')

{

p=1;

}

if(p!=1)

{

action<<ch;

}

else

{

leftstring<<ch;

}

}

//选择数据库

if(action.str()=="use")

{

cout<<"选择数据库:"<<leftstring.str().substr(1,leftstring.str().length()-1)<<endl;

this->dbname=leftstring.str().substr(1,leftstring.str().length()-1);

return;

}

//建立数据库或者表

if(action.str()=="create")

{

queryline=leftstring.str();

leftstring.str("");

action.str("");

p=0;

for(i=0;(unsigned int)i<queryline.length();i++)

{

ch=queryline.at(i);

//跳过第一个空白

if(i==0&&ch==' ')continue;

if(ch==' ')

{

p=1;

}

if(p!=1)

{

action<<ch;

}

else

{

leftstring<<ch;

}

}

//cout<<action.str();

if(action.str()=="database"){

cout<<"创建数据库"<<this->dbroot+"\\"+leftstring.str().substr(1,leftstring.str().length()-1)<<endl;

string dirname=this->dbroot+"\\"+leftstring.str().substr(1,leftstring.str().length()-1);

\_mkdir(dirname.c\_str());

return;

}

//创建表

if(action.str()=="table"){

queryline=leftstring.str();

leftstring.str("");

action.str("");

p=0;

for(i=0;(unsigned int)i<queryline.length();i++)

{

ch=queryline.at(i);

//跳过第一个空白

if(i==0&&ch==' ')continue;

if(ch==' ')

{

p=1;

}

if(p!=1)

{

action<<ch;

}

else

{

leftstring<<ch;

}

}

this->createfrm(action.str().substr(1,action.str().length()-2));

vector<string> tb;

queryline=leftstring.str();

leftstring.str("");

action.str("");

p=0;

int yyy=-1;

string tdstring;

for(i=0;(unsigned int)i<queryline.length();i++)

{

ch=queryline.at(i);

//跳过第一个空白和(号

if(i==0&&ch==' ')continue;

if(i==1&&ch=='(')continue;

if(ch==','||ch==')')

{

if(p==0){

leftstring<<action.str().substr(1,action.str().length()-1);

yyy=action.str().find('`',1);

if(yyy==-1)

{

cout<<"语法错误"<<endl;

return;

}

string tbname=action.str().substr(1,yyy-1);

string tbtype=action.str().substr(yyy+2,action.str().length()-1);

tdstring=tbname+"|"+tbtype;

p=1;

}

else

{

leftstring<<action.str().substr(2,action.str().length()-1);

yyy=leftstring.str().find('`',1);

if(yyy==-1)

{

cout<<"语法错误"<<endl;

return;

}

string tbname=leftstring.str().substr(0,yyy);

string tbtype=leftstring.str().substr(yyy+2,leftstring.str().length()-1);

tdstring=tbname+"|"+tbtype;

}

//string asd=leftstring.str();

tb.push\_back(tdstring);

action.str("");

leftstring.str("");

}

action<<ch;

}

this->createfield(this->frmname,tb);

//cout<<leftstring.str()<<endl;

return;

}

cout<<"语法错误"<<endl;

return;

}

//插入数据库

if(action.str()=="insert")

{

queryline=leftstring.str();

leftstring.str("");

action.str("");

p=0;

for(i=0;(unsigned int)i<queryline.length();i++)

{

ch=queryline.at(i);

//跳过第一个空白

if(i==0&&ch==' ')continue;

if(ch==' ')

{

p=1;

}

if(p!=1)

{

action<<ch;

}

else

{

leftstring<<ch;

}

}

if(action.str()=="into"){

queryline=leftstring.str();

leftstring.str("");

action.str("");

p=0;

for(i=0;(unsigned int)i<queryline.length();i++)

{

ch=queryline.at(i);

//跳过第一个空白

if(i==0&&ch==' ')continue;

if(ch==' ')

{

p=1;

}

if(p!=1)

{

action<<ch;

}

else

{

leftstring<<ch;

}

}

this->frmname=action.str().substr(1,action.str().length()-2);

queryline=leftstring.str();

//cout<<queryline<<endl;

leftstring.str("");

action.str("");

p=0;

for(i=0;(unsigned int)i<queryline.length();i++)

{

ch=queryline.at(i);

//跳过第一个空白

if(i==0&&ch==' ')continue;

if(ch==' ')

{

p=1;

}

if(p!=1)

{

action<<ch;

}

else

{

leftstring<<ch;

}

}

vector<string> tb;

queryline=leftstring.str();

if(action.str()!="values")

{

cout<<"语法错误"<<endl;

return;

}

leftstring.str("");

action.str("");

p=0;

for(i=0;(unsigned int)i<queryline.length();i++)

{

ch=queryline.at(i);

//跳过第一个空白和(号

if(i==0&&ch==' ')continue;

if(i==1&&ch=='(')continue;

if(ch==','||ch==')')

{

if(p==0){

leftstring<<action.str().substr(1,action.str().length()-2);

p=1;

}

else

{

leftstring<<action.str().substr(2,action.str().length()-3);

}

tb.push\_back(leftstring.str());

action.str("");

leftstring.str("");

}

action<<ch;

}

this->insert(this->frmname,tb);

//cout<<leftstring.str()<<endl;

return;

}

cout<<"语法错误"<<endl;

throw exception("error spell");

return;

}

//数据库查询语句..

if(action.str()=="select")

{

queryline=leftstring.str();

leftstring.str("");

action.str("");

p=0;

for(i=0;(unsigned int)i<queryline.length();i++)

{

ch=queryline.at(i);

//跳过第一个空白

if(i==0&&ch==' ')continue;

if(ch==' ')

{

p=1;

}

if(p!=1)

{

action<<ch;

}

else

{

leftstring<<ch;

}

}

string a=action.str();

if(action.str()=="from"){

queryline=leftstring.str();

leftstring.str("");

action.str("");

p=0;

for(i=0;(unsigned int)i<queryline.length();i++)

{

ch=queryline.at(i);

//跳过第一个空白

if(i==0&&ch==' ')continue;

if(ch==' ')

{

p=1;

}

if(p!=1)

{

action<<ch;

}

else

{

leftstring<<ch;

}

}

//表名

string frmname=action.str();

//

queryline=leftstring.str();

leftstring.str("");

action.str("");

p=0;

for(i=0;(unsigned int)i<queryline.length();i++)

{

ch=queryline.at(i);

//跳过第一个空白

if(i==0&&ch==' ')continue;

if(ch==' ')

{

p=1;

}

if(p!=1)

{

action<<ch;

}

else

{

leftstring<<ch;

}

}

//cout<<leftstring.str()<<endl;

int eq=leftstring.str().find("=");

int bg=leftstring.str().find(">");

int ls=leftstring.str().find("<");

vector<string> rs;

//if(eq!=-1&&bg!=-1&&bg<eq){

//

//}

rs.clear();

//如果有等号

if(eq!=-1)

{

vector<string> a=split(leftstring.str().substr(1,leftstring.str().length()),'=');

if(ls!=-1||bg!=-1)

{

a.at(0)=a.at(0).substr(1,a.at(0).length()-3);

}

else

{

a.at(0)=a.at(0).substr(1,a.at(0).length()-2);

}

a.at(1)=a.at(1).substr(1,a.at(1).length()-1);

this->select(frmname.substr(1,frmname.length()-2),a.at(0),a.at(1));

//临时vector

//将result存入rs这个临时vector

for(i=0;i<(int)this->result->size();i++)

{

rs.push\_back(result->at(i));

}

}

if(bg!=-1)

{

vector<string> a;

if(eq!=-1)

{

a=split(leftstring.str().substr(1,leftstring.str().length()),'=');

}

else

{

a=split(leftstring.str().substr(1,leftstring.str().length()),'>');

}

if(ls!=-1||(bg!=-1&&eq!=-1))

{

a.at(0)=a.at(0).substr(1,a.at(0).length()-3);

}

else

{

a.at(0)=a.at(0).substr(1,a.at(0).length()-2);

}

a.at(1)=a.at(1).substr(1,a.at(1).length()-1);

this->selectgt(frmname.substr(1,frmname.length()-2),a.at(0),a.at(1));

//如果存在=号..加入临时rs进result

for(i=0;i<(int)rs.size();i++)

{

this->result->push\_back(rs.at(i));

}

return;

}

if(ls!=-1)

{

vector<string> a;

if(eq!=-1)

{

a=split(leftstring.str().substr(1,leftstring.str().length()),'=');

}

else

{

a=split(leftstring.str().substr(1,leftstring.str().length()),'<');

}

if((ls!=-1&&eq!=-1)||bg!=-1)

{

a.at(0)=a.at(0).substr(1,a.at(0).length()-3);

}

else

{

a.at(0)=a.at(0).substr(1,a.at(0).length()-2);

}

a.at(1)=a.at(1).substr(1,a.at(1).length()-1);

this->selectls(frmname.substr(1,frmname.length()-2),a.at(0),a.at(1));

//如果存在=号..加入临时rs进result

for(i=0;i<(int)rs.size();i++)

{

this->result->push\_back(rs.at(i));

}

return;

}

return;

}

cout<<"语法错误"<<endl;

throw exception("error spell");

return;

}

//删除数据库中字段

if(action.str()=="delete")

{

queryline=leftstring.str();

leftstring.str("");

action.str("");

p=0;

for(i=0;(unsigned int)i<queryline.length();i++)

{

ch=queryline.at(i);

//跳过第一个空白

if(i==0&&ch==' ')continue;

if(ch==' ')

{

p=1;

}

if(p!=1)

{

action<<ch;

}

else

{

leftstring<<ch;

}

}

string a=action.str();

if(action.str()=="from"){

queryline=leftstring.str();

leftstring.str("");

action.str("");

p=0;

for(i=0;(unsigned int)i<queryline.length();i++)

{

ch=queryline.at(i);

//跳过第一个空白

if(i==0&&ch==' ')continue;

if(ch==' ')

{

p=1;

}

if(p!=1)

{

action<<ch;

}

else

{

leftstring<<ch;

}

}

//表名

string frmname=action.str();

//

queryline=leftstring.str();

leftstring.str("");

action.str("");

p=0;

for(i=0;(unsigned int)i<queryline.length();i++)

{

ch=queryline.at(i);

//跳过第一个空白

if(i==0&&ch==' ')continue;

if(ch==' ')

{

p=1;

}

if(p!=1)

{

action<<ch;

}

else

{

leftstring<<ch;

}

}

//cout<<leftstring.str()<<endl;

int eq=leftstring.str().find("=");

int bg=leftstring.str().find(">");

int ls=leftstring.str().find("<");

vector<string> rs;

//if(eq!=-1&&bg!=-1&&bg<eq){

//

//}

//如果有等号

if(eq!=-1)

{

vector<string> a=split(leftstring.str().substr(1,leftstring.str().length()),'=');

if(ls!=-1||bg!=-1)

{

a.at(0)=a.at(0).substr(1,a.at(0).length()-3);

}

else

{

a.at(0)=a.at(0).substr(1,a.at(0).length()-2);

}

a.at(1)=a.at(1).substr(1,a.at(1).length()-1);

this->remove(frmname.substr(1,frmname.length()-2),a.at(0),a.at(1));

//临时vector

}

if(bg!=-1)

{

vector<string> a;

if(eq!=-1)

{

a=split(leftstring.str().substr(1,leftstring.str().length()),'=');

}

else

{

a=split(leftstring.str().substr(1,leftstring.str().length()),'>');

}

if((ls!=-1&&eq!=-1)||bg!=-1)

{

a.at(0)=a.at(0).substr(1,a.at(0).length()-3);

}

else

{

a.at(0)=a.at(0).substr(1,a.at(0).length()-2);

}

a.at(1)=a.at(1).substr(1,a.at(1).length()-1);

this->removegt(frmname.substr(1,frmname.length()-2),a.at(0),a.at(1));

//如果存在=号..加入临时rs进result

for(i=0;i<(int)rs.size();i++)

{

this->result->push\_back(rs.at(i));

}

return;

}

if(ls!=-1)

{

vector<string> a;

if(eq!=-1)

{

a=split(leftstring.str().substr(1,leftstring.str().length()),'=');

}

else

{

a=split(leftstring.str().substr(1,leftstring.str().length()),'<');

}

if((ls!=-1&&eq!=-1)||bg!=-1)

{

a.at(0)=a.at(0).substr(1,a.at(0).length()-3);

}

else

{

a.at(0)=a.at(0).substr(1,a.at(0).length()-2);

}

a.at(1)=a.at(1).substr(1,a.at(1).length()-1);

this->removels(frmname.substr(1,frmname.length()-2),a.at(0),a.at(1));

//如果存在=号..加入临时rs进result

for(i=0;i<(int)rs.size();i++)

{

this->result->push\_back(rs.at(i));

}

return;

}

return;

}

cout<<"语法错误"<<endl;

return;

}

//删除数据库中字段

if(action.str()=="update")

{

queryline=leftstring.str();

leftstring.str("");

action.str("");

p=0;

for(i=0;(unsigned int)i<queryline.length();i++)

{

ch=queryline.at(i);

//跳过第一个空白

if(i==0&&ch==' ')continue;

if(ch==' ')

{

p=1;

}

if(p!=1)

{

action<<ch;

}

else

{

leftstring<<ch;

}

}

string a=action.str();

string frmname=action.str().substr(1,action.str().length()-2);

//

queryline=leftstring.str();

leftstring.str("");

action.str("");

p=0;

for(i=0;(unsigned int)i<queryline.length();i++)

{

ch=queryline.at(i);

//跳过第一个空白

if(i==0&&ch==' ')continue;

if(ch==' ')

{

p=1;

}

if(p!=1)

{

action<<ch;

}

else

{

leftstring<<ch;

}

cout<<"语法错误"<<endl;

return;

}

cout<<"语法错误"<<endl;

throw exception("error spell");

}

}

DB::DB(string dbroot)

{

this->dbroot=dbroot;

result=new vector<string>();

}

void DB::insert( string frmname,vector<string> row )

{

if(this->dbname=="")

{

cout<<"error:尚未选择数据库"<<endl;

throw new exception("error no database selected");

return;

}

this->frmname=frmname;

loadfield(frmname);

if(!ifexfrm())return;

if(!ifexfield())return;

if(row.size()!=this->field.size()){

cout<<"错误的数量"<<endl;

return;

}

int y=(unsigned int)atof(row[0].c\_str());

int i=0;

int tablepos=-1;

for(i=0;i<(int)fieldtype.size();i++)

{

if(fieldtype[i]=="key")

{

tablepos=i;

break;

}

}

if(tablepos==-1){

cout<<"无索引数据"<<endl;

throw exception("error no index file existed");

return;

}

//cout<<BTreeM[0]->search(y)<<endl;

if(BTreeM[tablepos]->search(y)!="\0"){

cout<<"已经存在..添加失败"<<endl;

throw exception("key already existed");

return;

}

string filename=this->dbroot+"\\"+this->dbname+"\\"+frmname+"\\"+"data.dbs";

//得到文件大小

ifstream f1(filename.c\_str());

f1.seekg(0,ios::end);

int filesize=f1.tellg();

f1.close();

ofstream fout(filename.c\_str(),ios\_base::app);

cout<<"新行:";

stringstream s;

//键的索引

stringstream columidx;

for(i=0;i<(int)row.size();i++)

{

columidx<<row[i].length();

if(i!=(int)row.size()-1)columidx<<'|';

}

//cout<<columidx.str();

for(i=0;i<(int)row.size();i++)

{

//编入索引

int index=(int)atof(row[i].c\_str());

if(index==0)index=row[i].length();

s<<filesize;

s<<'#'<<columidx.str();

if(fieldtype[i]=="key")

{

while(BTreeM[i]->insert(s.str(),index)==NULL)

{

index++;

//cout<<"warning:索引项已经存在,自增..新的索引项"<<index<<endl;

}

}

s.str("");

//保存索引

if(fieldtype[i]=="key")

{

string filename=this->dbroot+"\\"+this->dbname+"\\"+this->frmname+"\\"+this->field[i]+".idx";

ofstream\* idxout=new ofstream(filename.c\_str(),ios\_base::binary);

BTreeM[i]->dump(idxout);

idxout->close();

}

//cout<<row[i];

//注意string后会附加一个\0

fout.write(row[i].c\_str(),row[i].length());

if(i!=(int)row.size()-1)fout.write("\v",1);

}

cout<<"加入成功"<<endl;

fout.write("\f",1);

fout.close();

}

vector<string>\* DB::remove( string frmname,string tdname,string value,int y/\*=-1\*/ )

{

if(this->dbname=="")

{

cout<<"error:尚未选择数据库"<<endl;

throw exception("error no database selected");

return NULL;

}

if(y==-1)

{

/\*if(select(frmname,tdname,value)==NULL){

cout<<"error:没有此节点,,删除失败";

return NULL;

}\*/

this->frmname=frmname;

loadfield(frmname);

y=(int)atof(value.c\_str());

if(y==0)y=value.length();

}

int i=0;

int tablepos=-1;

//得到表对应tablepos

for(i=0;i<(int)field.size();i++){

if(field[i]==tdname)

{

tablepos=i;

break;

}

}

if(tablepos==-1)

{

cout<<"error:错误的表名"<<endl;

return NULL;

}

if(fieldtype[tablepos]=="key"){

//在索引中查询位置

string valueLe=BTreeM[tablepos]->search(y);

string fposs="";

if(valueLe=="")

{

cout<<"error:没有找到..无法删除"<<endl;

return NULL;

}

//这个int为整个要删除的长度

int vL=0;

//每个关键字长度

vector<int> valueL;

stringstream valueS;

for(int i=0;(unsigned int)i<valueLe.length();i++)

{

valueS<<valueLe[i];

if(valueLe[i]=='#')

{

fposs=valueLe.substr(0,i);

valueS.str("");

}

if(valueLe[i]=='|'||i==(int)valueLe.length()-1)

{

valueL.push\_back((int)atof(valueS.str().c\_str()));

vL=vL+(int)atof(valueS.str().c\_str());

valueS.str("");

}

}

vL=vL+(int)field.size()-1;

int fpos=(int)atof(fposs.c\_str());

//cout<<fposs.c\_str();

//cout<<fpos;

//查找其他关键字

string filename=this->dbroot+"\\"+this->dbname+"\\"+this->frmname+"\\data.dbs";

ifstream fin(filename.c\_str());

//定位

fin.seekg(fpos);

//开始读取

//清空结果迭代器

this->result->clear();

char ch=' ';

stringstream s;

int ii=0;

while(!fin.eof()) {

if((int)valueL.size()==0)break;

if(valueL.at(0)==ii){

//cout<<s.str()<<endl;

this->result->push\_back(s.str());

s.str("");

//vector<int>::iterator iter = colors.begin();

valueL.erase(valueL.begin());

ii=0;

//跳过一个字符

fin.get();

}

ch = fin.get();

s<<ch;

ii++;

}

fin.close();

//删除文件

filename=this->dbroot+"\\"+this->dbname+"\\"+this->frmname+"\\data.dbs";

fstream fst(filename.c\_str());

//先寻找第一个节点..如果相同则删除.不同则说明原节点已经被删除或被其他节点占用..寻找下一个key

if(value!=this->result->at(tablepos))

{

return remove(frmname,tdname,value,y+1);

}

//重定位

fst.seekg(fpos);

//cout<<fpos<<"--"<<vL<<endl;

ii=0;

while(ii<vL) {

fst.put('\xEE');

ii++;

}

cout<<"key:"<<this->result->at(tablepos)<<"删除成功"<<endl;

fst.close();

//ii=0;

//在索引中删除

//for(ii=0;(unsigned int)ii<this->result->size();ii++)

{

//cout<<;

this->delidx(tablepos,this->result->at(0));

}

return NULL;

}

else

{

//cout<<fposs;

int fpos=0;

//cout<<fpos;

string filename=this->dbroot+"\\"+this->dbname+"\\"+this->frmname+"\\data.dbs";

fstream fin(filename.c\_str());

//定位

fin.seekg(fpos);

//开始读取

//清空结果迭代器

this->result->clear();

char ch=' ';

stringstream s;

this->field.size();

//前行位置

int prelenth=0;

//行长度

int rowlen=0;

int found=0;

int i=0,j=0;

while(!fin.eof()) {

ch = fin.get();

rowlen++;

if(ch=='\x0B')

{

cout<<s.str()<<"--"<<value<<endl;

if(tablepos==i&&s.str()==value)

{

found=1;

}

i++;

s.str("");

}

else

{

if(ch=='\x0C')

{

cout<<s.str()<<endl;

if(tablepos==i&&s.str()==value)

{

found=1;

}

i++;

//如果成功删除

if(found){

cout<<"之前"<<prelenth<<endl;

fin.seekg(prelenth);

for(j=0;j<rowlen-1;j++)

{

fin.write("#",1);

}

}

prelenth+=rowlen;

fin.seekg(prelenth);

cout<<rowlen<<endl;

found=0;

i=0;

rowlen=0;

s.str("");

}

else

{

s<<ch;

}

}

}

fin.close();

return this->result;

}

}

bool DB::ifexfield()

{

if((int)field.size()!=0)

{

return true;

}

cout<<"error:无字段数据"<<endl;

//throw exception("error no fields");

return false;

}

bool DB::loadfield( string frmname )

{

this->field.clear();

this->fieldtype.clear();

this->BTreeM.clear();

string filename=this->dbroot+"\\"+this->dbname+"\\"+frmname+"\\data.dbs";

ifstream fin(filename.c\_str(),ios\_base::binary);

if(fin.fail())

{

fin.close();

cout<<"error:文件打开失败"<<endl;

throw exception("error files");

//throw exception("字段名不存在");

//清空已经导入的错误的表

this->field.clear();

this->fieldtype.clear();

return false;

}

char ch;

stringstream s;

string value="";

int i=0;

while(!fin.eof()) {

ch = fin.get();

s<<ch;

if (ch == '\x0b')

{

value=s.str().substr(0,s.str().length()-1);

//cout<<"tips:导入字段:"<<value<<endl;

this->field.push\_back(value.substr(0,value.find('|')));

this->fieldtype.push\_back(value.substr(value.find('|')+1,value.length()));

//读取字段的索引

loadindex(value.substr(0,value.find('|')),i);

s.str("");

i++;

}

if (ch == '\x0c')

{

//cout<<s.str()<<endl;

//清空s,,并保存到value

value=s.str().substr(0,s.str().length()-1);

//cout<<"tips:导入字段:"<<value<<endl;

this->field.push\_back(value.substr(0,value.find('|')));

this->fieldtype.push\_back(value.substr(value.find('|')+1,value.length()));

//读取字段的索引

loadindex(value.substr(0,value.find('|')),i);

s.str("");

i++;

return true;

}

}

this->field.clear();

this->fieldtype.clear();

cout<<"error:表中不存在字段..可能是错误的文件"<<endl;

throw exception("error files");

return false;

}

bool DB::ifexfrm()

{

if(this->frmname!="")

{

string filename=this->dbroot+"\\"+this->dbname+"\\"+this->frmname+"\\data.dbs";

ifstream fin(filename.c\_str(),ios\_base::binary);

if(fin.fail())

{

fin.close();

cout<<"error:表不存在"<<endl;

//throw exception("error no table existed");

//throw exception("表不存在");

return false;

}

return true;

}

else

{

cout<<"error:尚未选择表"<<endl;

throw exception("error no table selected");

//throw exception("尚未选择表");

return false;

}

}

void DB::loadindex( string index\_filename,int i )

{

if(fieldtype[i]!="key")

{

BTreeM.push\_back(new BTree());

BTreeM[i]->clear();

return;

}

if(!ifexfrm())return;

string filename=this->dbroot+"\\"+this->dbname+"\\"+this->frmname+"\\"+index\_filename+".idx";

ifstream\* fin=new ifstream(filename.c\_str(),ios\_base::binary);

BTreeM.push\_back(new BTree());

BTreeM[i]->clear();

int y=BTreeM[i]->load(fin);/\*

cout<<BTreeM[i]->searchNode(1)<<endl;

cout<<BTreeM[i]->searchNode(2)<<endl;

cout<<BTreeM[i]->searchNode(3)<<endl;

cout<<BTreeM[i]->searchNode(4)<<endl;

cout<<BTreeM[i]->searchNode(5)<<endl;\*/

cout<<"tips:导入索引"<<this->field[i]<<"共"<<y<<"条"<<endl;

}

vector<string>\* DB::selectall( string frmname,string tdname,vector<string> valueAl )

{

vector<string>\* rs=new vector<string>();

rs->clear();

int i=0,j=0;

for(i=0;i<(int)valueAl.size();i++)

{

select(frmname,tdname,valueAl[i]);

//保存result的进入rs

for(j=0;j<(int)this->result->size();j++)

{

rs->push\_back(result->at(j));

}

}

//清空结果集

this->result->clear();

//将临时数据存入result

for(i=0;i<(int)rs->size();i++)

{

this->result->push\_back(rs->at(i));

}

delete rs;

return this->result;

}

void DB::removeall( string frmname,string tdname,vector<string> valueAl )

{

int i=0,j=0;

for(i=0;i<(int)valueAl.size();i++)

{

//cout<<valueAl.at(i)<<endl;

this->remove(frmname,tdname,valueAl[i]);

}

//清空结果集

this->result->clear();

return;

}

vector<string>\* DB::selectgt( string frmname,string tdname,string minvalue,int y/\*=-1\*/ )

{

int i=0;

if(y==-1)

{

if(this->dbname=="")

{

cout<<"error:尚未选择数据库"<<endl;

throw exception("error no database selected");

return NULL;

}

this->frmname=frmname;

loadfield(frmname);

//在索引中查询位置

y=(int)atof(minvalue.c\_str());

}

int tablepos=-1;

//得到表对应tablepos

for(i=0;i<(int)field.size();i++){

if(field[i]==tdname)

{

tablepos=i;

break;

}

}

if(tablepos==-1)

{

cout<<"error:错误的表名"<<endl;

throw exception("error table name");

return NULL;

}

vector<string> valueGt=BTreeM[tablepos]->searchNode\_gt(y);

return this->selectall(frmname,tdname,valueGt);

}

vector<string>\* DB::removegt( string frmname,string tdname,string minvalue,int y/\*=-1\*/ )

{

int i=0;

if(y==-1)

{

if(this->dbname=="")

{

cout<<"error:尚未选择数据库"<<endl;

throw exception("error no database selected");

return NULL;

}

this->frmname=frmname;

loadfield(frmname);

//在索引中查询位置

y=(int)atof(minvalue.c\_str());

}

int tablepos=-1;

//得到表对应tablepos

for(i=0;i<(int)field.size();i++){

if(field[i]==tdname)

{

tablepos=i;

break;

}

}

if(tablepos==-1)

{

cout<<"error:错误的表名"<<endl;

return NULL;

}

vector<string> valueGt=BTreeM[tablepos]->searchNode\_gt(y);

this->removeall(frmname,tdname,valueGt);

return this->result;

}

vector<string>\* DB::removels( string frmname,string tdname,string maxvalue,int y/\*=-1\*/ )

{

int i=0;

if(y==-1)

{

if(this->dbname=="")

{

cout<<"error:尚未选择数据库"<<endl;

throw exception("error no database selected");

return NULL;

}

this->frmname=frmname;

loadfield(frmname);

//在索引中查询位置

y=(int)atof(maxvalue.c\_str());

if(y==0)y=maxvalue.length();

}

int tablepos=-1;

//得到表对应tablepos

for(i=0;i<(int)field.size();i++){

if(field[i]==tdname)

{

tablepos=i;

break;

}

}

if(tablepos==-1)

{

cout<<"error:错误的表名"<<endl;

return NULL;

}

vector<string> valueLe=BTreeM[tablepos]->searchNode\_le(y);

this->removeall(frmname,tdname,valueLe);

return this->result;

}

vector<string>\* DB::selectls( string frmname,string tdname,string maxvalue,int y/\*=-1\*/ )

{

int i=0;

if(y==-1)

{

if(this->dbname=="")

{

cout<<"error:尚未选择数据库"<<endl;

throw exception("error no database selected");

return NULL;

}

this->frmname=frmname;

loadfield(frmname);

//在索引中查询位置

y=(int)atof(maxvalue.c\_str());

if(y==0)y=maxvalue.length();

}

int tablepos=-1;

//得到表对应tablepos

for(i=0;i<(int)field.size();i++){

if(field[i]==tdname)

{

tablepos=i;

break;

}

}

if(tablepos==-1)

{

cout<<"error:错误的表名"<<endl;

return NULL;

}

vector<string> valueLe=BTreeM[tablepos]->searchNode\_le(y);

return this->selectall(frmname,tdname,valueLe);

}

}

**封装的数据库DLL**

#pragma once

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

#include "DB.h"

namespace DBcore {

/// <summary>

/// CCDControl 摘要

/// </summary>

///

/// 警告: 如果更改此类的名称，则需要更改

/// 与此类所依赖的所有.resx 文件关联的托管资源编译器工具的

/// “资源文件名”属性。否则，

/// 设计器将不能与此窗体的关联

/// 本地化资源正确交互。

public ref class DBcore : public System::Windows::Forms::UserControl

{

public:

DBcore(String^ dbroot)

{

d=new DB(ConvertToString(dbroot));

line=new vector<string>();

}

void query(String^ query)

{

try

{

d->result->clear();

d->queryline(ConvertToString(query));

nowline=0;

totalnums=d->result->size();

}

catch (Exception^ e)

{

throw e;

}

}

/\*\*将结果处理成行

\*每运行一次这个语句下移一行

\*/

bool fetchline()

{

if(d->field.size()<=0) return false;

int i=0,y=0;

line->clear();

string tmp="";

while(y<(int)(totalnums/d->field.size()))

{

for (i=0;i<(int)d->field.size();i++)

{

tmp=d->result->at(y\*d->field.size()+i);

if(y==nowline)

{

line->push\_back(tmp);

}

}

if(y==nowline)

{

nowline++;

break;

}

y++;

}

if(line->size()==0)

return false;

return true;

}

/\*\*得到当前行时的值

\*@param String^ tablename

\*/

String^ get(String^ tablename)

{

if(line->size()==0)

return "";

int i=0;

int tablepos=-1;

//得到表对应tablepos

for(i=0;i<(int)d->field.size();i++){

if(d->field[i]==ConvertToString(tablename))

{

tablepos=i;

break;

}

}

if(tablepos==-1)

{

cout<<"error:错误的表名"<<endl;

return "";

}

return gcnew String(line->at(tablepos).c\_str());

}

private:

DB\* d;

vector<string>\* line;

int nowline;

int totalnums;

protected:

std::string ConvertToString(System::String^ str)

{

int q=(int)System::Runtime::InteropServices::Marshal::StringToHGlobalAnsi(str);

char\* p=(char\*)q;

return std::string(p);

}

};

}

**主界面Form1.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using DBcore;

using TTS;

using Readfile;

namespace BookLib

{

public partial class Form1 : Form

{

public DBcore.DBcore db;

public Form1()

{

InitializeComponent();

db = new DBcore.DBcore(".\\");

}

private void Form1\_Load(object sender, EventArgs e)

{

db.query("use DB");

//db.query("insert into `book` values ('1','电子电工技术','电子电工类的专业书..优秀的课本','老师','电子工业出版社','电子类','3','5')");

}

//强制退出

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

Application.Exit();

}

//查询图书

private void button1\_Click(object sender, EventArgs e)

{

bookresult.Rows.Clear();

if (textBox1.Text != "")

{

db.query("select from `book` where `id`='" + textBox1.Text + "'");

}

if (textBox2.Text != "")

{

db.query("select from `book` where `booktitle`='" + textBox2.Text + "'");

}

if (textBox3.Text != "")

{

db.query("select from `book` where `publisher`='" + textBox3.Text + "'");

}

if (textBox12.Text != "")

{

db.query("select from `book` where `writer`='" + textBox12.Text + "'");

}

if (textBox11.Text != "")

{

db.query("select from `book` where `type`='" + textBox11.Text + "'");

}

if (textBox18.Text != "")

{

int i = int.Parse(textBox18.Text) - 1;

db.query("select from `book` where `id`>'" + i.ToString() + "'");

}

if (textBox19.Text != "")

{

int i = int.Parse(textBox19.Text) + 1;

db.query("select from `book` where `id`<'" + i.ToString() + "'");

}

while (db.fetchline())

{

string id = db.get("id");

string booktitle = db.get("booktitle");

string description = db.get("description");

string writer = db.get("writer");

string publisher = db.get("publisher");

string type = db.get("type");

string store = db.get("store");

string nums = db.get("nums");

DataGridViewRow row = new DataGridViewRow();

row.CreateCells(bookresult);

row.Cells[0].Value = id;

row.Cells[1].Value = booktitle;

row.Cells[2].Value = description;

row.Cells[3].Value = writer;

row.Cells[4].Value = publisher;

row.Cells[5].Value = type;

row.Cells[6].Value = store;

row.Cells[7].Value = nums;

bookresult.Rows.Add(row);

}

bookresult.Visible = true;

}

//隐藏结果框

private void tabControl1\_MouseClick(object sender, MouseEventArgs e)

{

bookresult.Visible = false;

readerresult.Visible = false;

}

//添加新图书

private void button2\_Click(object sender, EventArgs e)

{

if (new\_bookid.Text == "")

{

MessageBox.Show("请填写书号");

return;

}

try

{

db.query("insert into `book` values ('" + new\_bookid.Text + "','" + new\_bookname.Text + "','" + new\_bookdes.Text + "','" + new\_writer.Text + "','" + new\_publisher.Text + "','" + new\_cate.Text + "','" + new\_nums.Text + "','" + new\_nums.Text + "')");

speaker.speak("新图书添加成功");

}

catch (Exception ex)

{

speaker.speak("新图书添加失败");

}

}

//删除图书

private void button3\_Click(object sender, EventArgs e)

{

if (textBox8.Text != "")

{

try

{

db.query("select from `book` where `id`='" + textBox8.Text + "'");

if (db.fetchline())

{

string store = db.get("store");

string nums = db.get("nums");

if (store != nums)

{

speaker.speak("图书尚有未归还;无法删除");

return;

}

db.query("delete from `book` where `id`='" + textBox8.Text + "'");

speaker.speak("图书注销成功");

return;

}

else

{

speaker.speak("此图书不存在");

return;

}

}

catch (System.Exception ex)

{

speaker.speak("此图书不存在");

return;

}

}

else

{

speaker.speak("请输入书号");

}

}

//新读者添加

private void button4\_Click(object sender, EventArgs e)

{

try

{

db.query("insert into `reader` values ('" + textBox13.Text + "','" + textBox10.Text + "','" + comboBox1.Text + "','" + textBox4.Text + "','0','0')");

speaker.speak("新读者添加成功");

}

catch (Exception ex)

{

speaker.speak("新读者添加失败;学号已经存在");

}

}

private void comboBox1\_KeyPress(object sender, KeyPressEventArgs e)

{

e.Handled = true;

}

//借阅图书

private void button5\_Click(object sender, EventArgs e)

{

int nums;

try

{

nums = int.Parse(textBox5.Text);

}

catch (System.Exception ex)

{

speaker.speak("请输入正确的数量");

return;

}

//得到学生资料

string sid = textBox7.Text;

db.query("select from `reader` where `sid`='" + sid + "'");

string sname = "";

string ssex = "";

string smaxnum = "";

string slending = "";

string sowe = "";

int slend = 0;

if (db.fetchline())

{

sname = db.get("name");

ssex = db.get("sex");

smaxnum = db.get("maxnum");

slending = db.get("lending");

sowe = db.get("owe");

slend = int.Parse(slending);

int snum = int.Parse(smaxnum);

//如果过多

if (slend + nums > snum)

{

speaker.speak("该读者不能再借");

return;

}

slend = slend + nums;

}

else

{

speaker.speak("此读者不存在");

return;

}

//得到图书资料

string bid = textBox6.Text;

string booktitle = "无";

string description = "无";

string writer = "无";

string publisher = "无";

string type = "无";

string store = "0";

string bnums = "0";

int bstore = 0;

db.query("select from `book` where `id`='" + bid + "'");

if (db.fetchline())

{

booktitle = db.get("booktitle");

description = db.get("description");

writer = db.get("writer");

publisher = db.get("publisher");

type = db.get("type");

store = db.get("store");

bnums = db.get("nums");

bstore = int.Parse(store);

if (bstore - nums < 0)

{

speaker.speak("库存不足");

return;

}

bstore = bstore - nums;

}

else

{

speaker.speak("此图书不存在");

return;

}

DateTime t1 = new DateTime(1970, 1, 1);

DateTime t2 = DateTime.Now;

long a = t2.Ticks - t1.Ticks / 1000;

int yy = (int)a;

int i = 0;

for (i = 0; i < nums; i++)

{

//重复..直到插入成功

d:

try

{

db.query("insert into `lendinfo` values ('" + yy + "','" + bid + "','" + sid + "')");

}

catch (Exception ex)

{

yy++;

goto d;

}

}

db.query("delete from `reader` where `sid`='" + sid + "'");

db.query("insert into `reader` values ('" + sid + "','" + sname + "','" + ssex + "','" + smaxnum + "','" + slend + "','" + sowe + "')");

db.query("delete from `book` where `id`='" + bid + "'");

db.query("insert into `book` values ('" + bid + "','" + booktitle + "','" + description + "','" + writer + "','" + publisher + "','" + type + "','" + bstore.ToString() + "','" + bnums + "')");

speaker.speak("操作成功");

}

private void textBox6\_KeyPress(object sender, KeyPressEventArgs e)

{

if (e.KeyChar < 59 && e.KeyChar > 47 || e.KeyChar == 8)

{

e.Handled = false;

}

else

{

e.Handled = true;

}

}

private void button7\_Click(object sender, EventArgs e)

{

int nums;

try

{

nums = int.Parse(textBox15.Text);

}

catch (System.Exception ex)

{

speaker.speak("请输入正确的数量");

return;

}

//得到学生资料

string sid = textBox17.Text;

db.query("select from `reader` where `sid`='" + sid + "'");

string sname = "";

string ssex = "";

string smaxnum = "";

string slending = "";

string sowe = "";

int slend = 0;

if (db.fetchline())

{

sname = db.get("name");

ssex = db.get("sex");

smaxnum = db.get("maxnum");

slending = db.get("lending");

sowe = db.get("owe");

slend = int.Parse(slending);

int snum = int.Parse(smaxnum);

slend = slend - nums;

if (slend < 0)

{

speaker.speak("数量出错");

return;

}

}

else

{

speaker.speak("此读者不存在");

return;

}

//得到图书资料

string bid = textBox16.Text;

string booktitle = "";

string description = "";

string writer = "";

string publisher = "";

string type = "";

string store = "";

string bnums = "";

int bstore = 0;

db.query("select from `book` where `id`='" + bid + "'");

if (db.fetchline())

{

booktitle = db.get("booktitle");

description = db.get("description");

writer = db.get("writer");

publisher = db.get("publisher");

type = db.get("type");

store = db.get("store");

bnums = db.get("nums");

bstore = int.Parse(store);

bstore = bstore + nums;

if (bstore > int.Parse(bnums))

{

speaker.speak("数量出错");

return;

}

}

else

{

speaker.speak("此图书不存在");

return;

}

//检测图书是否是该读者所借

DBcore.DBcore db2 = new DBcore.DBcore(".\\");

db2.query("use DB");

db2.query("select from `lendinfo` where `readerid`='" + sid + "'");

int isyours=0;

System.Collections.ArrayList lendid=new System.Collections.ArrayList();

while(db2.fetchline())

{

if(db2.get("bookid")==bid)

{

isyours=1;

lendid.Add(db2.get("id"));

}

}

if(isyours==0)

{

speaker.speak("该书非该读者拥有");

return;

}

DateTime t1 = new DateTime(1970, 1, 1);

DateTime t2 = DateTime.Now;

long a = t2.Ticks - t1.Ticks / 1000;

int yy = (int)a;

int i = 0;

//清理图书借阅单

for (i = 0; i < nums; i++)

{

db.query("delete from `lendinfo` where `id`='" + lendid[i] + "'");

}

db.query("delete from `reader` where `sid`='" + sid + "'");

db.query("insert into `reader` values ('" + sid + "','" + sname + "','" + ssex + "','" + smaxnum + "','" + slend + "','" + sowe + "')");

db.query("delete from `book` where `id`='" + bid + "'");

db.query("insert into `book` values ('" + bid + "','" + booktitle + "','" + description + "','" + writer + "','" + publisher + "','" + type + "','" + bstore.ToString() + "','" + bnums + "')");

speaker.speak("操作成功");

}

private void button6\_Click(object sender, EventArgs e)

{

readerresult.Rows.Clear();

if (textBox9.Text != "")

{

db.query("select from `reader` where `sid`='" + textBox9.Text + "'");

}

if (textBox14.Text != "")

{

db.query("select from `reader` where `name`='" + textBox14.Text + "'");

}

//循环读取

while (db.fetchline())

{

string id = db.get("sid");

//去找借的书的名字

string booknames = "";

DBcore.DBcore db2 = new DBcore.DBcore(".\\");

db2.query("use DB");

db2.query("select from `lendinfo` where `readerid`='" + id + "'");

while (db2.fetchline())

{

string bcd = db2.get("bookid");

DBcore.DBcore db3 = new DBcore.DBcore(".\\");

db3.query("use DB");

db3.query("select from `book` where `id`='" + bcd + "'");

while (db3.fetchline())

{

booknames = booknames +db3.get("booktitle") +"||";

}

}

string booktitle = db.get("name");

string description = db.get("sex");

string writer = db.get("lending");

string publisher = db.get("maxnum");

string type = db.get("owe");

DataGridViewRow row = new DataGridViewRow();

row.CreateCells(readerresult);

row.Cells[0].Value = id;

row.Cells[1].Value = booktitle;

row.Cells[2].Value = description;

row.Cells[3].Value = writer;

row.Cells[4].Value = publisher;

row.Cells[5].Value = type;

row.Cells[6].Value = booknames;

//添加新行

readerresult.Rows.Add(row);

}

readerresult.Visible = true;

}

//限制输入数字

void onlynums(object sender, KeyPressEventArgs e)

{

if (e.KeyChar < 59 && e.KeyChar > 47 || e.KeyChar == 8)

{

e.Handled = false;

}

else

{

e.Handled = true;

}

}

private void textBox1\_KeyPress(object sender, KeyPressEventArgs e)

{

onlynums(sender, e);

}

private void textBox9\_KeyPress(object sender, KeyPressEventArgs e)

{

onlynums(sender, e);

}

private void textBox5\_KeyPress(object sender, KeyPressEventArgs e)

{

onlynums(sender, e);

}

private void textBox7\_KeyPress(object sender, KeyPressEventArgs e)

{

onlynums(sender, e);

}

private void textBox13\_KeyPress(object sender, KeyPressEventArgs e)

{

onlynums(sender, e);

}

private void textBox8\_KeyPress(object sender, KeyPressEventArgs e)

{

onlynums(sender, e);

}

private void new\_bookid\_KeyPress(object sender, KeyPressEventArgs e)

{

onlynums(sender, e);

}

private void new\_nums\_KeyPress(object sender, KeyPressEventArgs e)

{

onlynums(sender, e);

}

private void textBox4\_KeyPress(object sender, KeyPressEventArgs e)

{

onlynums(sender, e);

}

private void textBox17\_KeyPress(object sender, KeyPressEventArgs e)

{

onlynums(sender, e);

}

private void textBox16\_KeyPress(object sender, KeyPressEventArgs e)

{

onlynums(sender, e);

}

}

}