**2022-2023-1学期**

**C++项目报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 文件状态  [ ]草稿  [√]正式发布  [ ]正在修改 | 文件标识 | realSmallSQL | |
| 当前版本 | 第三版 | |
| 小组成员 | 2021091202004 | 王森 |
| （学号） | 杨镇豪 |
| （学号） | 胡伊然 |
| 完成日期 | 2023/2/13 | |

电子科技大学信息与软件工程学院

版本历史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本/状态 | 作者 | 参与者 | 起止日期 | 备注 |
| 第一版 | 杨镇豪 | 王森，胡伊然 | 2022/12/12~2023/1/12 |  |
| 第二版 | 胡伊然 | 杨镇豪，王森 | 2023/1/24~2022/2/4 |  |
| 第三版 | 王森 | 胡伊然，杨镇豪 | 2023/2/5~2023/2/20 |  |
|  |  |  |  |  |

目录

[第一章 引言 1](#_Toc127624081)

[1.1 项目背景 1](#_Toc127624082)

[1.2 术语与缩写解释 1](#_Toc127624083)

[第二章 系统需求 2](#_Toc127624084)

[2.1 系统需求分析 2](#_Toc127624085)

[第三章 系统设计 4](#_Toc127624086)

[3.1 系统结构 4](#_Toc127624087)

[3.2 系统设计 4](#_Toc127624088)

[3.3 输出状态 5](#_Toc127624089)

[3.4 系统功能 5](#_Toc127624090)

[第四章 系统实现 5](#_Toc127624091)

[4.1 Table的建立与更多操作 5](#_Toc127624092)

[4.2 远程连接的重要部分 6](#_Toc127624093)

[4.3 B+树索引的重要部分 6](#_Toc127624094)

[第五章 系统测试 6](#_Toc127624095)

[第六章 结语 7](#_Toc127624096)

[参考文献 8](#_Toc127624097)

# 引言

## 1.1 项目背景

数据库管理系统是一个能够提供数据录入、修改、查询的数据操作软件，具有数据定义、数据操作、数据存储与管理、数据维护、通信等功能，且能够允许多用户使用。另外，数据库管理系统的发展与计算机技术发展密切相关。而且近年来，计算机技术逐渐成为人们生活的重要组成部分。为此，若要进一步完善计算机数据库管理系统，技术人员就应当不断创新、改革计算机技术，并不断拓宽计算机数据库管理系统的应用范围，从而真正促进计算机数据库管理系统技术的革新。

为了练习使用C++语言并且深入学习数据库原理等相关知识，我们小组决定自己编写一个微型数据库管理系统。

## 1.2 术语与缩写解释

|  |  |
| --- | --- |
| 术语/缩写 | 解释 |
| Key | B+树主键值，在程序中要求是primary键为int才可以进行比较 |
| Entry | 含有主键值和偏移量的结构，作为单个节点导入B+树 |
| 数据块 | B+树的每个节点由多个Entry组成，这个节点就叫做数据块 |
| 远程模式 | 在开始程序的时候，可以通过输入1进入远程方式，这样就可以通过远程方式对数据库进行操作 |
| 客户端 | 如果设置为远程方式，则可以通过多个listener.cc向service.cc发出命令，我们称开启的listener.cc为客户端 |
| 服务端 | 我们称接收命令的service.cc为服务端 |
| M\_order | B+树的阶数 |
|  |  |

# 第二章 系统需求

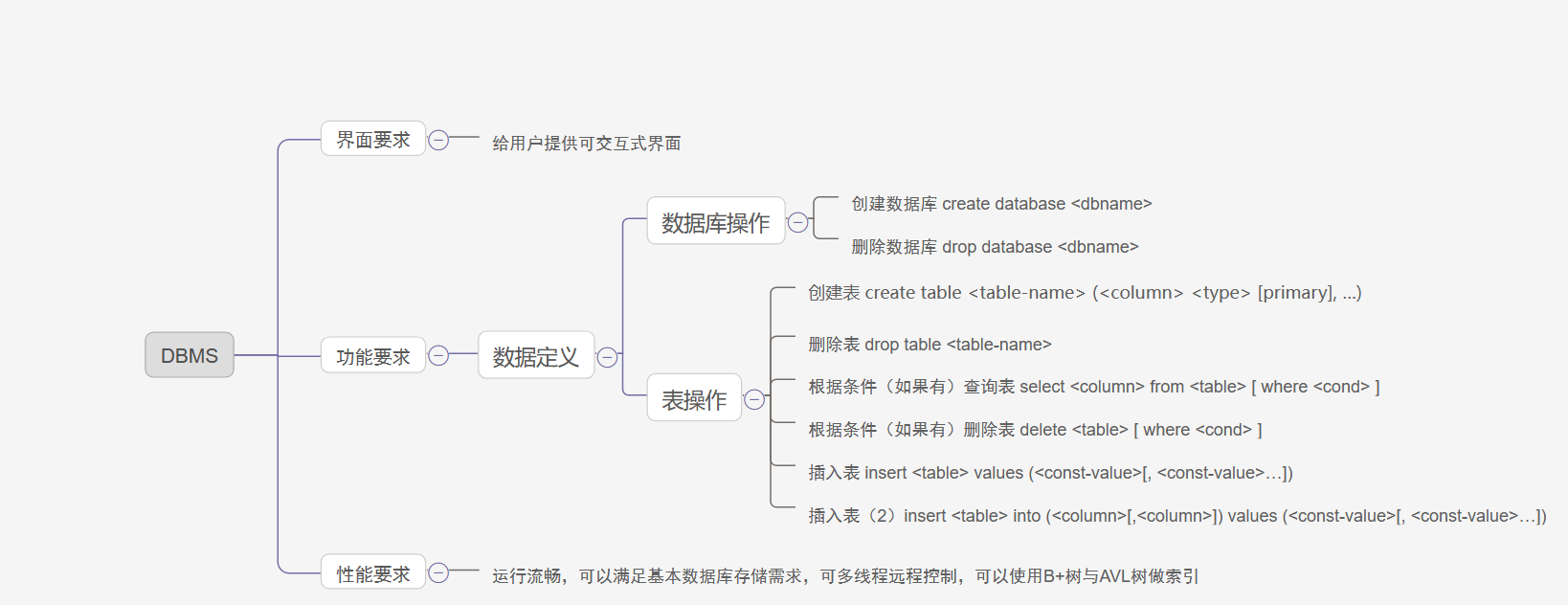


图2-1 系统需求图

## 2.1 系统需求分析

1.数据定义：DBMS提供数据定义语言DDL（Data Definition Language），供用户定义数据库的三级模式结构。DDL主要用于建立、修改数据库的库结构以及删除数据库。DDL所描述的库结构仅仅给出了数据库的框架，数据库的框架信息被存放在数据字典（Data Dictionary）中。

2.数据操作：DBMS提供数据操作语言DML（Data Manipulation Language），供用户实现对数据的插入一个或多个数据等操作。

3.数据库的运行管理：数据库的运行管理功能是DBMS的运行控制、管理功能DCL（Data Control Language），但是由于我们只是简单模拟了mySQL的功能，所以相应的我们只做了通过ip查询的函数

4.性能要求：我们需要索引以此更快的进行数据的查找，除此之外，为了满足远程的需要，我们还需要做远程多线程处理，以此满足远程数据库操作的需要。

5.我们还需要满足客户端方面的需求，首先，我们需要通过编写函数来显示我们表的状态，除此之外，我们还需要一定的错误反馈功能，我们做了简单的错误反馈并对此进行了异常的定义。

6.我们需要接收用户方面的输入信息作为我们数据库管理程序的依据，所以我们就免不了对输入数据的处理，我们打算在某个sql语句中尝试进行解析，以期望达到括号与逗号的位置不必相互贴合的效果。除此之外，我们需要满足远程客户的需求，我们计划将结果放入日志中暂为储存，如果结果小于1024个字符，我们将这个东西返回给用户。

7.最后，我们还需要做到基本的无卡顿，全程流畅的要求，以及项目在 GitHub上进行代码托管，另外需要比较详细的readme与类图。

以上，便是我们的项目的全部需求

# 第三章 系统设计

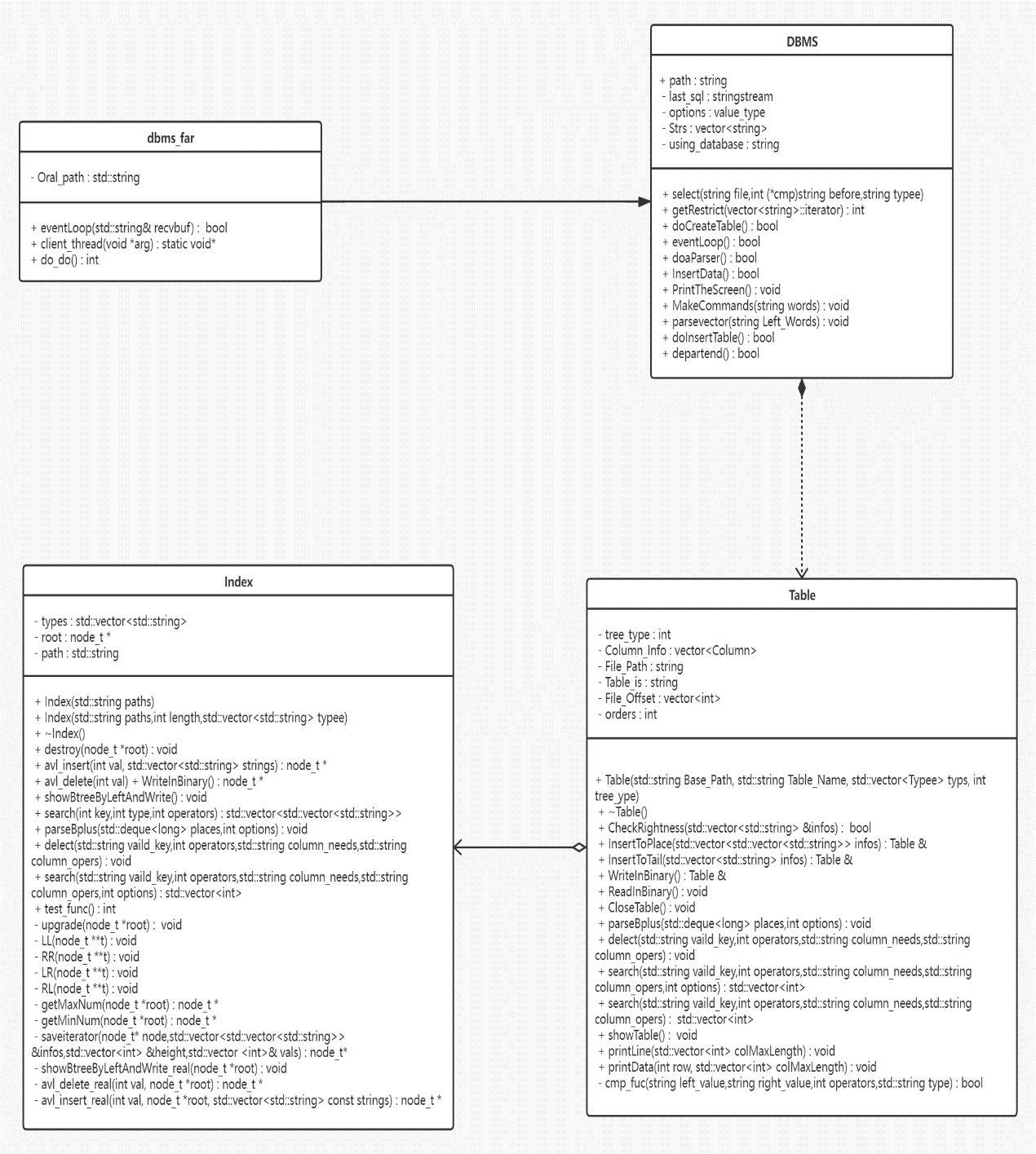


图 3-1 系统架构图

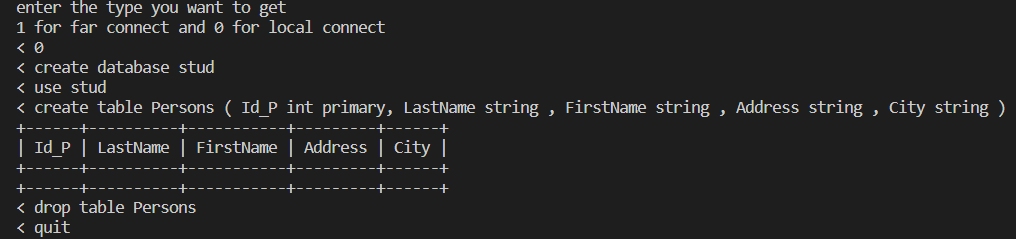
## 3.1 系统结构

上图是我们的系统架构图，我们认为总体的系统应该由上述四个部分组成一共有上述4个架构，由于dbms\_far与DBMS结构相似，实际上我们只有三个结构，即Table(表)，Index(索引)，以及DBMS(数据控制系统)，但是在实际的操作中，我们发现了更多问题亟待解决，所以我们对表的结构进行了微调，实际的表图可以在项目文件的file/img/realsmallSQL.pdf中进行查看。

## 3.2 系统设计

我们需要数据库管理的系统，除此之外，我们需要数据查询效率更高以及数据库管理效率更高，除此还要支持远程操作，为此，我们做出了如下的设计，我们将DBMS与DBMS\_far进行了结合，使远程可以与本地共用一个DBMS地址，我们通过这种方法将DBMS终端工作量进行了削减，除此，我们还需要索引的处理，在这里，我们发现，作为AVL和B+树的索引，其本质上是一个键值对的形式，这就可以像SDL库里的结构一样去处理这些数据，这就需要我们增加一个Index索引类，处理这些，除此之外，由于B+树结构过于复杂，我们不得不将其分离出系统，作为单独的一个类，除此之外，我们将剩余的功能放入Table类，这样我们就可以通过这种方式解决这一问题，从而进行数据库的增删改查功能，以及数据库的表的输出。

## 3.3 输出状态

 图 3-2 单程输出图

我们模仿了mysql数据库的输出状态，如图3-2所示而在远程连接的时候我们增加了输出的状态，如图3-3所示，其中client\_ip是连接的ip，port是连接的端口，nums是现在的远程终端连接数。

图片包含 图形用户界面

描述已自动生成 图 3-3 远程输出图

## 3.4 系统功能

我们完成了所有项目作业需求的功能，并增加了B+树与AVL索引以及远程的访问。

# 第四章 系统实现

## 4.1 Table的建立与更多操作

由于我们组的table函数过长，所以我们截取重要的部分做了流程图。如图4-1所示，我们另外截取了一部分table的代码，这一段代码是table的二进制读写，所需的。

图示

描述已自动生成

图 4-1 Table的简略图

  Table &WriteInBinary()

{

BAddTree<int,Entry> tree(orders);

int nots=1;

std::ofstream file(File\_Path + "/" + Table\_is, std::ios::out | std::ios::binary);

int nums = Column\_Info.size();

int numss = Column\_Info[0].Columns.size();

int primaryplace = 0;

int place = 0;

file.write((char \*)&nums, sizeof(int));

file.write((char \*)&numss, sizeof(int));

for (auto Column\_every : Column\_Info)

{

type\_Document types;

const char \*num = Column\_every.id.data();

sprintf(types.id\_convert, "%s", Column\_every.id.data());

sprintf(types.type\_convert, "%s", Column\_every.type.data());

types.Restrict = Column\_every.Restrict;

if (types.Restrict == Primary\_KY)

{

primaryplace = place;

if(types.type\_convert=="string")

{

nots=0;

}

}

place++;

file.write((char \*)&types, sizeof(types));

}

int need=0;

for(auto Col:Column\_Info)

{

if(Col.Restrict==Primary\_KY)

{

break;

}

need++;

}

int m=0;

File\_Offset.clear();

for (auto Column\_every : Column\_Info)

{

int i = file.tellp();

File\_Offset.emplace\_back(i);

int j=0;

for (auto info : Column\_every.Columns)

{

info\_Document infos;

sprintf(infos.Info\_One, "%s", info.data());

file.write((char \*)&infos, sizeof(infos));

if(m==0&&nots==1)

{

long k = file.tellp();

//std::cout << i << std::endl;

Entry ant(atoll(Column\_Info[need].Columns[j].c\_str()),k);

tree.insert(ant);

}

j++;

}

m++;

}

if(nots==1)

{

tree.searchall("Bplustree/idx/"+ Table\_is);

std::vector<std::string> dec;

for(auto cols:Column\_Info)

{

dec.emplace\_back(cols.id);

}

Index \*index = new Index(Table\_is,Column\_Info.size(),dec);

std::vector<std::string> decc;

for(int i=0;i<Column\_Info[0].Columns.size();i++)

{

for(auto cols:Column\_Info)

{

decc.emplace\_back(cols.Columns[i]);

}

index->avl\_insert(atoi(Column\_Info[need].Columns[i].c\_str()),decc);

}

index->showBtreeByLeftAndWrite();

int i=1+1;

}

file.close();

return \*this;

}

void ReadInBinary()

{

/\*\*

\* @brief 将二进制文件TABLE\_NAME.table读出

\*

\*

\*/

type\_Document temptype;

info\_Document info;

std::ifstream file(File\_Path + "/" + Table\_is,

std::ios::in | std::ios::binary);

BAddTree<int,Entry> b(orders);

if(!file.is\_open())

{

file.close();

return;

}

int nums, numss;

file.read((char \*)&nums, sizeof(int));

file.read((char \*)&numss, sizeof(int));

int c = 0;

while (c != nums)

{

Column types;

types.id = "", types.Restrict = 0, types.type = "", types.Columns = {""};

Column\_Info.emplace\_back(types);

file.read((char \*)&temptype, sizeof(temptype));

c++;

std::string str1 = temptype.id\_convert;

Column\_Info[c].id.swap(str1);

std::string str2 = temptype.type\_convert;

Column\_Info[c].type.swap(str2);

Column\_Info[c].Restrict = temptype.Restrict;

}

Column\_Info.erase(Column\_Info.begin());

Column types;

types.id = "", types.Restrict = 0, types.type = "", types.Columns = {""};

Column\_Info.emplace\_back(types);

std::ifstream in("Bplustree/idx/"+Table\_is,std::ios::in|std::ios::binary);

// if(!in.eof())

// b.ReadBtree(0,in);

// std::cout<<"read things"<<std::endl;

// b.tree\_traversal(itf);

File\_Offset.clear();

for (int j = 0; j < nums; j++)

{

auto offset = file.tellg();

File\_Offset.emplace\_back(offset);

if (tree\_type == AVL)

{

/\*insert AVL tree things\*/

}

for (int i = 0; i < numss; i++)

{

if (j == 0 && i >= 0 )

{

auto offset = file.tellg();

//std::cout <<offset<<std::endl;

}

file.read((char \*)&info, sizeof(info));

Column col;

Column\_Info[j].Columns.emplace\_back(std::string{""});

Column\_Info[j].Columns[i] = info.Info\_One;

}

Column\_Info[j].Columns.pop\_back();

auto iter = Column\_Info[j].Columns.begin();

}

Column\_Info.pop\_back();

}

## 4.2 远程连接的重要部分

远程连接作为我们的项目中较为核心的部分，我们选择将代码展现出来。

static void\* client\_thread(void \*arg)

{

int recvlen = 0;

char recvbuf[1024] = "";

long connfd = (long)arg;

dbms\_far dms;

while((recvlen = recv(connfd,recvbuf,sizeof(recvbuf),0))>0)

{

std::cout<<"recv\_buf:"<<recvbuf<<recvlen<<std::endl;

std::mutex mut;

send(connfd, recvbuf, recvlen, 0);

mut.lock();

std::string al(recvbuf);

if(dms.eventLoop(al))

{

memset(recvbuf,'\0',1024);

std::cout<<">>";

}

mut.unlock();

}

std::cout<<"client closed"<<recvbuf<<std::endl;

close(connfd);

nums--;

return NULL;

}

int do\_do()

{

int sockfd;

int log;

int connfd;

pthread\_t thread\_id;

struct sockaddr\_in server,client;

bzero(&server,sizeof(server));

server.sin\_family = AF\_INET;

server.sin\_port = htons(PORT);

server.sin\_addr.s\_addr = htonl(INADDR\_ANY);//在本地的

//创建套接字

sockfd = socket(AF\_INET,SOCK\_STREAM,0);

if(sockfd < 0)

std::cout<<"socket create error\n"<<std::endl;

log = bind(sockfd,(struct sockaddr\*)&server,sizeof(server));

if(log < 0)

{

std::cout<<"bindfd error\n"<<std::endl;

close(sockfd);

exit(-1);

}

log = listen(sockfd,20);

if(log < 0)

{

std::cout<<"listen error\n"<<std::endl;

printf("listen error\n");

close(sockfd);

exit(-1);

}

std::cout<<"waiting client >>>>>"<<std::endl;

while(1)

{

socklen\_t client\_len = sizeof(client);

//accept

connfd = accept(sockfd,(struct sockaddr\*)&client,&client\_len);

if(connfd < 0)

{

std::cout<<"connfd error"<<std::endl;

continue;

}

char cli\_ip[INET\_ADDRSTRLEN] = "";

inet\_ntop(AF\_INET, &client.sin\_addr, cli\_ip, INET\_ADDRSTRLEN);

std::cout<<"----------------------------------------------"<<std::endl;

printf("client ip=%s,port=%d,nums=%d\n", cli\_ip,ntohs(client.sin\_port),++nums);

pthread\_create(&thread\_id,NULL,&client\_thread,(void\*)connfd);

pthread\_detach(thread\_id);//线程分离，结束时自动回收线程

}

close(sockfd);

return 0;

};

这一代码由两部分构成一部分是主代码do\_do，而另一部分是附代码client\_thread，我们在do\_do中分配端口网址，然后将分配的结果传给pthread\_create函数，通过这种方式，我们实现了远程的多线程结果，除此之外，我们增加了线程锁，从而解决了线程冲突的问题。

我们演示完毕服务端的核心代码。这一部分是用户端的核心代码，这一个类负责连接并向服务端循环发送信息。当输入为quit时，直接退出程序，除此，我们还为这个程序增加了一个快速查看ip的功能，但在此不做赘述。

class dolinstener

{

public:

dolinstener(std::string readbuf)

{

bzero(&server, sizeof(server));

server.sin\_family = AF\_INET;

server.sin\_port = htons(PORT);

server.sin\_addr.s\_addr = inet\_addr(readbuf.c\_str());//the local\_ip you get and you should copy on it.

sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

if (sockfd < 0)

{

std::cout << "sockfd error\n"

<< std::endl;

exit(-1);

}

log = connect(sockfd, (struct sockaddr \*)&server, sizeof(server));

if (log < 0)

{

std::cout << "connect error\n"

<< std::endl;

close(sockfd);

exit(-1);

}

}

void closes()

{

close(sockfd);

}

void rotateplay()

{

fgets(sendbuf, sizeof(sendbuf), stdin);

send(sockfd, sendbuf, strlen(sendbuf), 0);

recv(sockfd, recvbuf, sizeof(recvbuf), 0);

std::string rec(recvbuf);

rec[4]='\0';

if(strcmp(rec.c\_str(),"quit")==0)

{

send(sockfd,"exit",strlen("exit"), 0);

close(sockfd);

exit(1);

}

std::cout <<recvbuf<< std::endl;

std::cout<<"<<";

\*sendbuf = {0};

\*recvbuf = {0};

}

char sendbuf[1024];

char recvbuf[1024];

private:

int sockfd, log;

struct sockaddr\_in server;

};

## 4.3 B+树索引的重要部分

B+树作为一种很复杂的数据结构，这里我们主要介绍他的插入删除与写入文件做简单的介绍，详细的代码可以看BplusTree文件夹。

由于代码管理问题，我们的索引只支持主键为int类型的索引，我们先介绍较为简单的Insert函数，插入后，我们开始判重，如果重复则返回错误，否则进行插入，插入之后我们开始判断是否出现了上溢，如果(m\_order+1) / 2>子节点的数量，则出现了上溢，随后，我们便可以对该节点进行分裂。

随后我们介绍Delete函数，首先我们判重，如果重复则返回错误，否则进行插入，插入之后我们开始判断是否出现了下溢，如果(m\_order+1) / 2<子节点的数量，则出现了下溢，随后，我们就可以对旁边的节点进行借与还，分为叶节点和非叶节点进行处理，一共分为3种情况，

随后我们介绍Save和Read函数，我们先遍历该树，然后按照树的位置顺序进行排列，便于之后的读取。具体的实现可以看BAddTree.h中的实现。

template<typename K, typename E>

inline bool BAddTree<K, E>::insert(E const &e)

{

BAddTreeLeafNode<K, E>\*v = this->search(e);

if (v) {//不允许插入重复值 因为插入要上溢下溢可能导致一个节点的2边都存在重复值，导致查找太过复杂

return false;

}

else {

K k = e.key();

v = static\_cast<BAddTreeLeafNode<K, E>\*>(this->m\_hitParentNode);

int \_s = v->key.size();

int i = 0;

for (; i < \_s; ++i) {

K&tk = v->key[i];

if (tk > k)break;

}

v->key.insert(v->key.begin()+i,k);//插入关键字

++this->m\_size;//更新规模

v->child.insert(v->child.begin() + i + 1, nullptr);//插入空分支

E\*\_e = new E();

\*\_e =std::move(e);//使用右值移动

v->e.insert(v->e.begin() + i, \_e);//插入记录

solveOverFlow(v);//上溢则分裂

return true;

}

}

template<typename K, typename E>

inline bool BAddTree<K, E>::remove(E const &e)

{

BAddTreeLeafNode<K, E>\*v = this->search(e);

if (!v) {

return false;

}

int rank = 0; while (v->key[rank] != e.key())++rank;

//删除

v->key.erase(v->key.begin() + rank);//删除关键字

E\*\_e = v->e[rank];

v->e.erase(v->e.begin() + rank);//删除记录

--this->m\_size;

delete \_e;//释放记录

v->child.erase(v->child.begin() + rank + 1);//删除分支

solveUnderFlow(v);//检测下溢

return true;

}

# 第五章 系统测试

## 5.1 本地测试

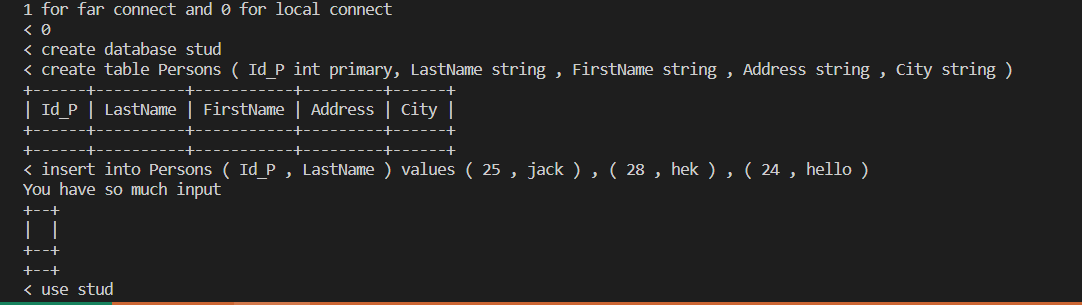


图 5-1 测试图1

文本

描述已自动生成我们先尝试了在没有表的情况下insert表，他回答了You have so much input的异常，并返回了一个空表，这证明它可以对错误进行返回，如图5-1所示

图 5-2 测试图2

**文本

描述已自动生成** 我们之后尝试了改变之后插入表，如图5-2所示，发现可以插入，并且可以正常查找。我们随后进行删除，如图5-4所示。

图 5-3 测试图3

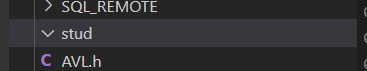
****

图 5-4 测试图4

## 5.2 远程测试

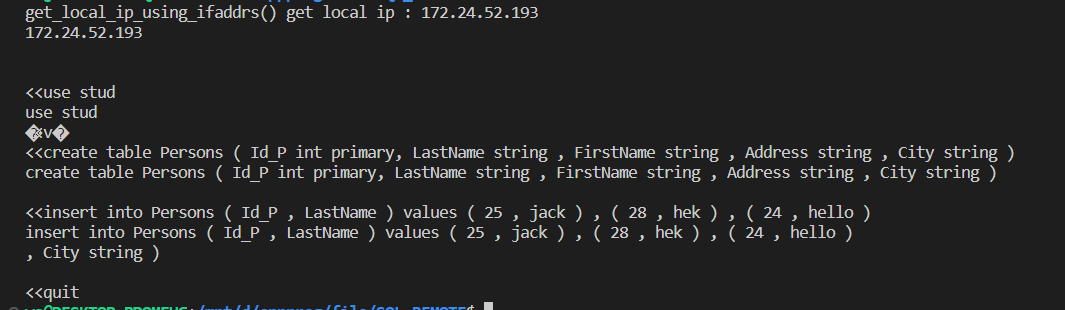


图 5-5 客户端测试

我们在客户端进行了测试，发现可以正常运行，达到要求。

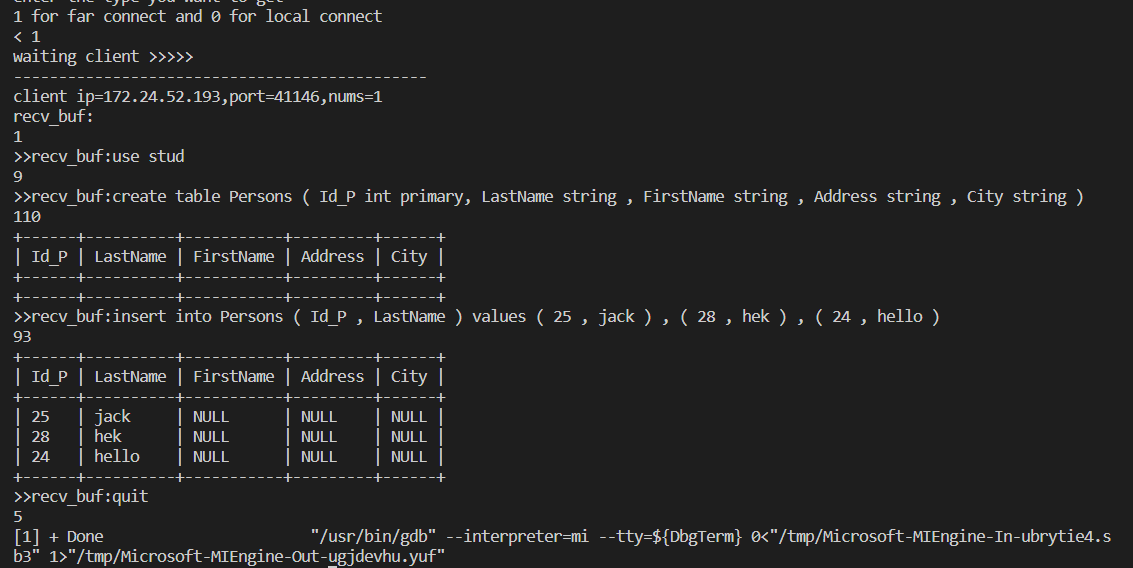


图5-6服务端信息

# 第六章 结语

6.1 项目总结

我们项目是一个基于C++的模仿MySQL的数据处理系统，我们通过代码协作托管的方式进行协作商讨，一起攻坚困难，在编写B+树这一部分时，我们小组的成员充分理解了B+树，并对C++的模板有了更充分的理解，我们通过改变思考的方式解决了大量的难题，除此之外，我们小组的成员尽力保持代码的规范，我们用到了很多数据结构和语法，如AVL树，B+树，链表，线程锁等等，可以说，编写一个大型的C++项目是一项足够大的挑战，没有小组协作和规范的代码，项目很难持续下去。

6.2 不足与期望

我们的项目仍有很多不足，比如B+树和AVL树不能支持非主键int类型的索引，而且也不支持联合主键索引，而且支持的指令数量非常有限，除此之外，我们还需要对远程多线程功能进行优化，我们需要将服务端返回的结果返回到客户端中，除此之外，我们还需要将B+树和AVL树的索引完善，使其达到最好的状态。另外，我们的代码有一定的冗余，这一点需要以后进行修改。除此之外，我们需要增加项目的兼容性，以使之能在Windows和Linux下使用，我们还需要一个Linux下的操作面板，使之具有图形化的功能，任重而道远，所以，我们小组希望以后继续进行这一个项目的开展，请随时关注。

6.3 项目心得

王森-2021091202004:

通过这一个项目，我对C++的认识更加深入，除此之外，作为组长，我深刻意识到了组织项目活动的重要性，一个大型的C++项目一定是需要小组的相互协作才可以成功的，我负责本项目的索引的部分，在这一部分，我更加深入的理解了C++的模板，C++的多线程与C++远程的使用方法，并使之融会贯通形成了对此的理解，由此，这一项目让我在锻炼与他人合作以及代码能力方面都有所进步。

# 参考文献

[1]崔斌, 吕雁飞, 陈学轩. 闪存环境下B+树索引重访[J]. 计算机应用, 2010(1):4.

[2]简易数据库管理系统(DBMS)设计与实现, CSDN, 2020年6月25日, <https://blog.csdn.net/qq_33187136/article/details/106959733>

[3] 吴会松;浅谈几种数据库系统的结构设计 [J].电脑知识与技术,1996,9:8-10.DOI:10.14004/j.cnki.ckt.2018.1180.

[4] MySQL索引-B+树, cnblogs, 2021年5月19日, [MySQL索引-B+树（看完你就明白了） - 苍青浪 - 博客园 (cnblogs.com)](https://www.cnblogs.com/cangqinglang/p/15042752.html)

[5] 吴恒山, 徐晓军, 桂浩. 基于改进B+树索引的结构连接算法[J]. 计算机工程, 2005, 31(16):3.

[6] 余华云, 蔡建红, 张维. 基于多线程的远程数据库电话访问系统的设计与实现[J]. 计算机工程, 1999, 25(4):3