**数据库系统project报告**

2021-2022学年第2学期（CST31106）

|  |  |
| --- | --- |
| 数据库系统project任务书 | |
| 名称 | 数据库存储管理设计与模拟实现 |
| 类型 | □验证性 □设计性 综合性 |
| 内容 | 关系型数据库采用自描述的方式进行数据库系统管理，关于关系的相关数据也存储在数据库管理系统里。  针对关系型数据库存储模块，进行需求分析，并设计一个简单的数据库系统存储模块，根据设计模拟实现数据存储管理器，主要实现：数据库创建、表格创建、数据添加、删除、更新等操作过程中，数据库系统所进行的存储方面的管理；设计索引，并比较有索引和无索引的区别  模拟实现采用：python或者Java实现具体功能，数据存储采用excel或者文本文件，可用文本文件模拟硬盘数据块，不需要实现SQL语句的执行编译，SQL语句采用函数实现。 |
|  | （1）设计方案要合理；  （2）能基于该存储管理模块实现数据的存储和读取；  （3）设计方案有一定的效率分析。 |
| 任务时间 | 2022年4月28日至2022年5月20日 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 小组成员 | | | | | | |
| 20194195 | | 20194159 | | 20194207 | | 20191583 |
| 王梓宇 | | 张翀 | | 钟祥新 | | 周逸群 |
| 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 |

小组分工:

|  |  |
| --- | --- |
| 王梓宇 | 数据库设计，调试和编写大部分第一版数据库的基础功能模块，并形成第一版可用的数据库；对实验报告做最后的完善和补充工作。 |
| 张翀 | 数据库设计，修复了部分第一版数据库bug，并在此基础上增添了额外的功能模块，对比了有无索引的情况下的查询时间，形成第二版数据库。 |
| 周逸群 | 数据库设计，测试和验收第二版数据库的功能是否满足需求，并编写报告 |
| 钟祥新 | 数据库设计，测试和验收第二版数据库的功能是否满足需求，并编写报告 |

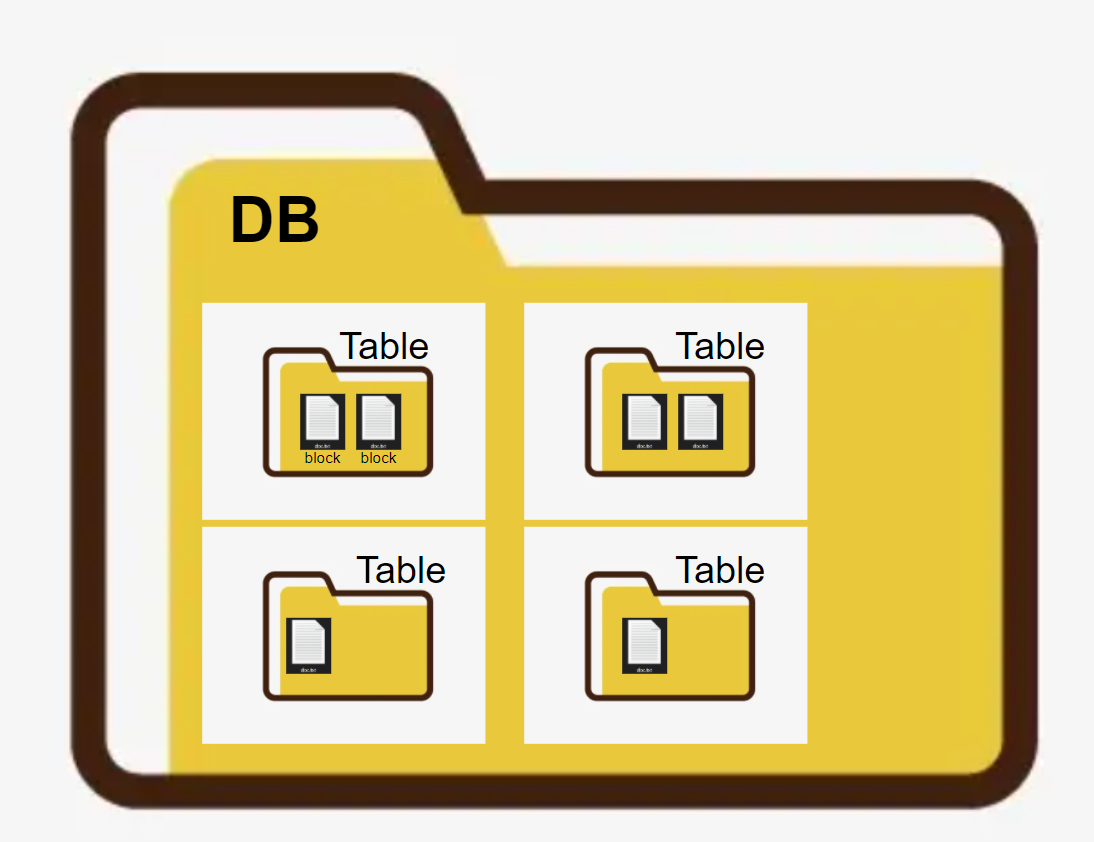
## 1 需求分析

本项目要求从底层实现一个能够满足基本需求的数据库系统存储模块，其基本功能包括建库、建表、增删改等。本项目的重点在于实现数据库底层的存储管理机制，因此需特别设计底层数据块的管理机制。

实验要求比较数据库系统在有无索引情况下的速度，所以我们需要结合这一要求进一步讨论底层数据存储的具体形式。实验允许用excel或者文本的形式保存数据，编程语言可以选用python或者java。考虑到python和jave的excel包本身即提供了类似索引的查询加速功能，不利于比较我们实现的索引机制性能，因此我们决定采用文本文件作为底层存储。在具体存储方式上，我们选择了java序列化方式。表中各个块及其数据，以及数据库管理表的数据，以及索引文件都视为java类，需要存储时会用序列化的方式将它们以2进制的形式存到.txt文件里。需要用时则根据这些.txt将其加载为对应的java类。

除了上述底层功能需求之外，本项目提供的数据库系统还应实现一个用户交互界面。考虑到本项目的重点在于底层逻辑，因可以用命令行的形式提供交互。

## 2 设计思路



**图一：文件存储层次结构**

首先给出文件存储的层次结构。我们主要以父子文件夹来表达数据库与表的包含关系。层次结构如下所示：在顶层数据路径中，我们将其中的每一个文件夹都视为一个数据库；每一个数据库文件夹中又包含多个子文件夹，每个子文件夹被视为一张表; 每一个子

文件夹中包含了多个文本文件，每一个文本文件被视为一个物理块，保存了序列化后的数据；每个物理块的大小固定为4KB，大小刚好为内存中的一页.序列化后的数据文件无法直接访问，进一步起到了加密的作用。

对于每一个数据库来说，需要单独设置一张表记录各种其各表的属性名称及参数；对于每一个数据表来说，需要有一个表内的索引文件。

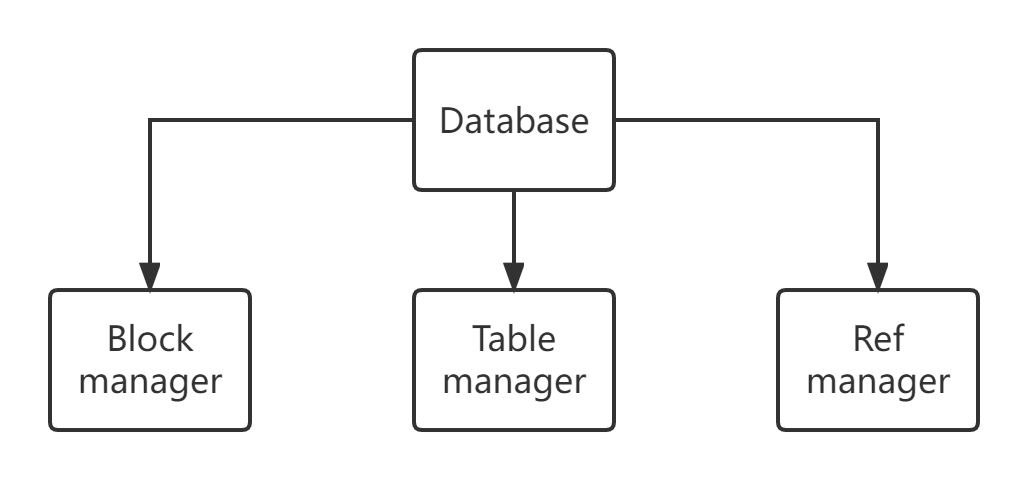
接下来我们综合考虑索引和块机制。处于逻辑简洁性以及运行时效率的考虑，我们将物理块中保存的数据定义为无序的。为了实现高效的空间管理，我们需要再引入一个位示图的结构。每一张表拥有一个自己的位示图，位示图的行代表着这个表的占用的某个物理块，位示图的列进一步代表块中的某个具体数据，值为1表示该数据有效，0表示无效。位示图的行列结合起来，标记数据库某一具体的数据位的占用情况。常用的替换策略包括FIFO\LRU\随机等，出于简洁的角度，在这里我们的数据替换采用随机的策略。索引机制用哈希表的形式实现。索引文件以序列化的形式保存在硬盘上，在数据库启动时载入内存。



**图二：用于标识表中数据是否有效的位示图**

上图为逻辑上的位示图，在具体实现它时，我们并没有把每个块的位示图信息都汇总在一起形成一个单独的文件，相反，每个块对应的在内存中的java类都有bitmap字段，它表示了对应此块的这一行位示图。一个表中所有块的bitmap字段汇总起来才是完整的位示图。

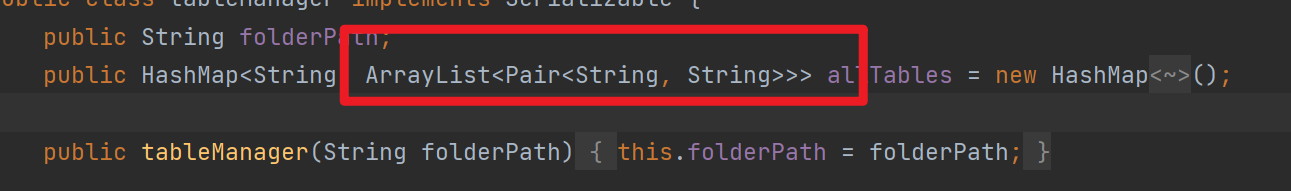
## 3 系统实现



**图三：数据库逻辑管理结构**

本次数据库按照下面的结构设计，我们用Java实现三个类，分别为：

**Table manager:** 表管理器，为了避免在每个block中存储重复的信息，我们定义table manager类。，用于存储同个数据库中不同表的参数，具体而言，存储每个表的“属性名:数据类型”。

* 
* **图四：blockManager类主要字段allTables的属性**

**Block manager**: 块管理器，由于数据库存在分块存储，因此定义block manager用于管理一个数据库中的所有表，可以选择一次性加载数据库中所有表的所有块，或者数据库中某一个表的所有块。

**Reference manager**: 索引管理器，用于管理数据库下所有表的索引，可以加载数据库中所有索引 (loadAllRef) 和保存数据库中的所有索引 (saveAllRef)。单个ref用于管理某个表的索引。

下面根据CRUD操作详细介绍每个功能的底层实现

### 3.1 数据库创建

在main函数中，根据菜单提示选择“创建数据库”

public static void getMenu1() {  
 System.out.println("创建数据库请输入 1");  
 System.out.println("查找数据库请输入 2");  
 System.out.println("进入数据库请输入 3");  
 System.out.println("删除数据库请输入 4");  
 System.out.println("获取菜单请输入 5");  
 System.out.println("退出程序请输入 6");  
}

点击“1”后进入table类，调用createDB函数。该函数功能很简单，就是在根文件夹下建立新的数据库文件夹。

public static void CreateDB(String name) throws IOException {  
 File file = new File(name);  
 file.mkdirs();  
}

完成数据库创建后，显示进一步操作的菜单：

public static void getMenu2() {  
 System.out.println("创建表请输入 1");  
 System.out.println("查找此数据库下所有的表请输入 2");  
 System.out.println("向某表中插入数据请输入 3");  
 System.out.println("删除某表的数据请输入 4");  
 System.out.println("修改某表的数据请输入 5");  
 System.out.println("查找某表的数据请输入 6");  
 System.out.println("获取菜单请输入 7");  
 System.out.println("删除某张表请输入 8");  
 System.out.println("退出此数据库请输入 9");  
}

这里我们依次介绍每个功能的实现细节。

#### 3.1.1 创建表

创建表最重要的函数是table类的CreateTable，首先从命令行输入读入用户输入。用户首先输入“表格名称”，然后按照“属性:类型”的格式输入该表的属性和数据类型。输入“exit”结束创建表。



**图五：功能菜单以及插入数据时的命令行界面图**

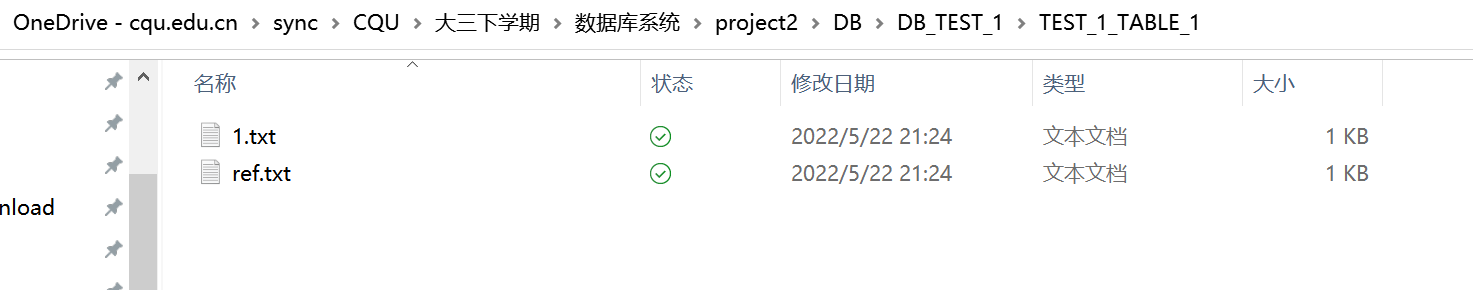
底层实现方面：依次读入用户输入，按照给定格式解析，将其存入ArrayList中。

Scanner scan = new Scanner(System.in);  
System.out.print("输入表格名称：");  
String name = scan.nextLine();  
System.out.println("输入表格属性名称及类型，以:分隔，每行仅包含一个属性.");  
System.out.println("请在第一行输入主键,主键请在类型后以\_pk结尾, 最后一行以单独的exit结尾：\"");  
  
File table = new File(DB + "\\" + name);  
table.mkdir();  
ArrayList<Pair<String, String>> params = new ArrayList<Pair<String, String>>();  
int size = 0;  
do {  
 String line = scan.nextLine();  
 if (line.equals("exit")) {  
 break;  
 }  
 String[] column = line.split(":");  
 params.add(new Pair<String, String>(column[0], column[1]));  
 if (column[1].startsWith("I")) {  
 size += 4;  
 } else if (column[1].startsWith("F")) {  
 size += 4;  
 } else {  
 String[] tmp = column[1].split("\_");  
 size += Integer.parseInt(tmp[1]);  
 }  
} while (true);

结束用户输入后，再创建该表的块，并创建索引。

main.tManager.insert(name, params); // tableManager存储table的参数  
// 创建该数据库的第一个块  
Block firstBlock = new Block(DB + "\\" + name, "1", 4096 / size);  
  
HashMap<String, Block> temp = new HashMap<String, Block>();  
temp.put("1.txt", firstBlock);  
main.bManager.blocks.put(name, temp);  
  
// 为该表建立索引  
ref tableRef = new ref(DB + "\\" + name);  
main.rManager.allRef.put(name, tableRef);

可以看到，在对应的数据库文件夹下，成功新建了表和索引文件。



#### 3.1.2 插入数据

首先在命令行中提示用户输入“3”以选择插入数据：

System.out.println("请输入您要进入的数据库名称");  
String name =sc.nextLine();  
dbName=rootPath+ "\\" + name;  
tManager= DataBase.tableManager.loadTables(dbName+ "\\tableManager.txt");  
bManager= blockManager.loadDBAllBlock(dbName);  
rManager= refManager.loadAllRef(dbName);  
func1();

用户输入需要插入数据的表名，然后载入该表：

System.out.println("请输入你要操作的表名");  
String tableName =sc.nextLine();  
String tablePath =dbName+ "\\" + tableName;  
ArrayList<String> values = new ArrayList<String>();  
ArrayList<Pair<String, String>> prams =tManager.allTables.get(tableName);

剩下的逻辑和上面创建数据库中的步骤，在此不再赘述：

boolean test = Boolean.parseBoolean(sc.nextLine());  
// 找到主键,并保证传入后续函数中的第一个值即为主键  
for (int i = 0; i < prams.size() && !flag; i++) {  
 System.out.print(prams.get(i).getKey());  
 System.out.println(" " + prams.get(i).getValue());  
 if (i == 0) {  
 System.out.println("请输入主键:");  
 String pkValue =sc.nextLine();  
 if (rManager.getRef(tableName).r.containsKey(pkValue)) {  
 System.out.println("表中已存在该主键,此次插入操作失败!");  
 flag = true;  
 } else {  
 values.add(pkValue);  
 }  
 } else {  
 System.out.println("请输入该属性的值");  
 values.add(sc.nextLine());  
 }

下图展示了插入数据和查询数据的过程，从查询的表返回的数据可知数据插入成功。



**图六：插入和查询数据时的命令行界面图**

注：其中可看到name字段长度过长，与左边的ID字段相连的现象，这是由于数据库基于字符串的.format函数进行格式化显示，默认一个属性的长度为10，长度为10的字符串已经适用大部分情况，长度过长会显得其他属性过于空旷。所以折衷取为10。

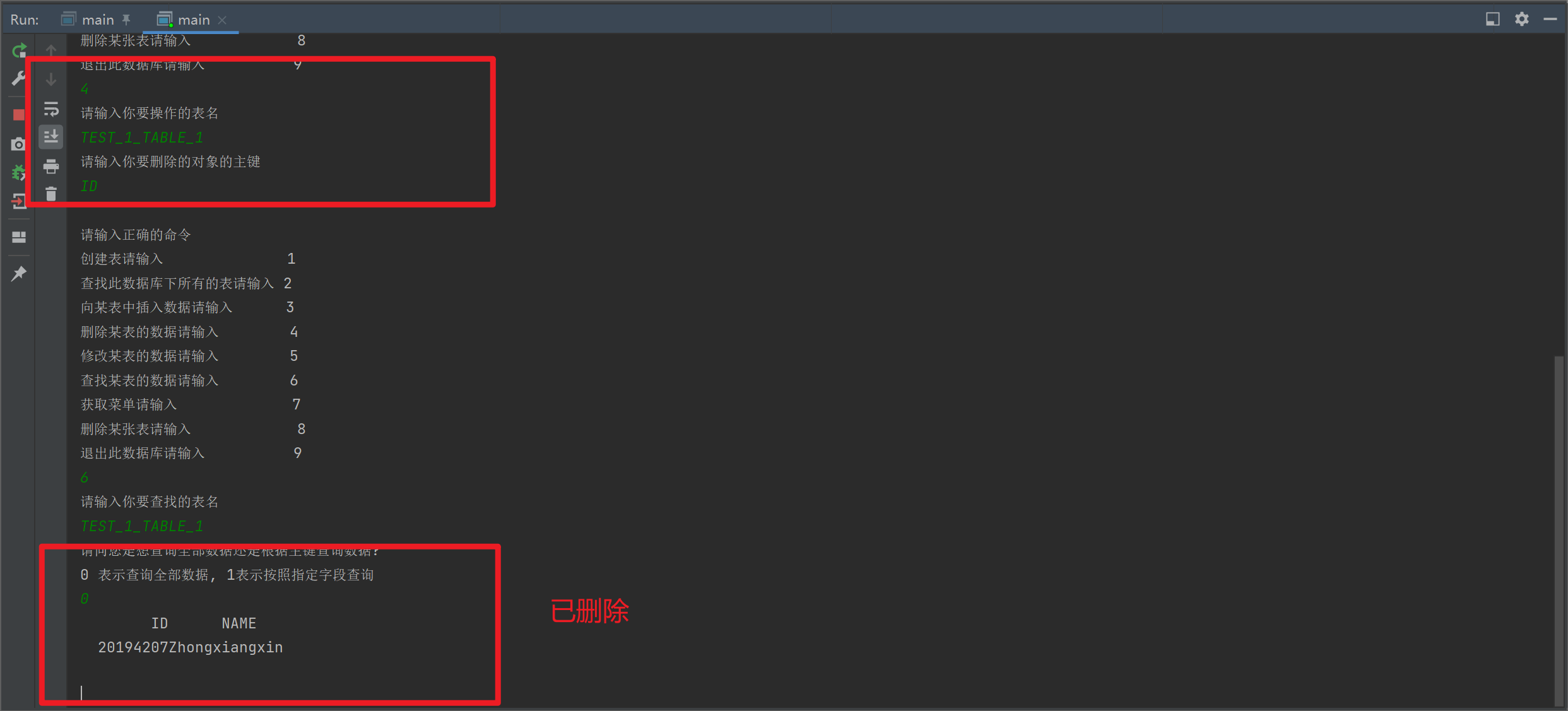
#### 3.1.3 删除某表的数据

因为数据库的表以块的形式存储，如果要删除某表的数据，第一步是根据索引文件找到该数据位于哪个块，第二步找到该数据位于该块的哪一个行，最后将该对象对应的位示图中的值标记为0，（标记为0表示该数据无效）并从索引文件中删除。



**图七：删除数据功能模块的代码示例**

可以看到，在命令行输入删除表TEST\_1\_TABLE\_1中主键为ID的命令后，再次查询时该表中已经没有这条数据，说明删除成功。



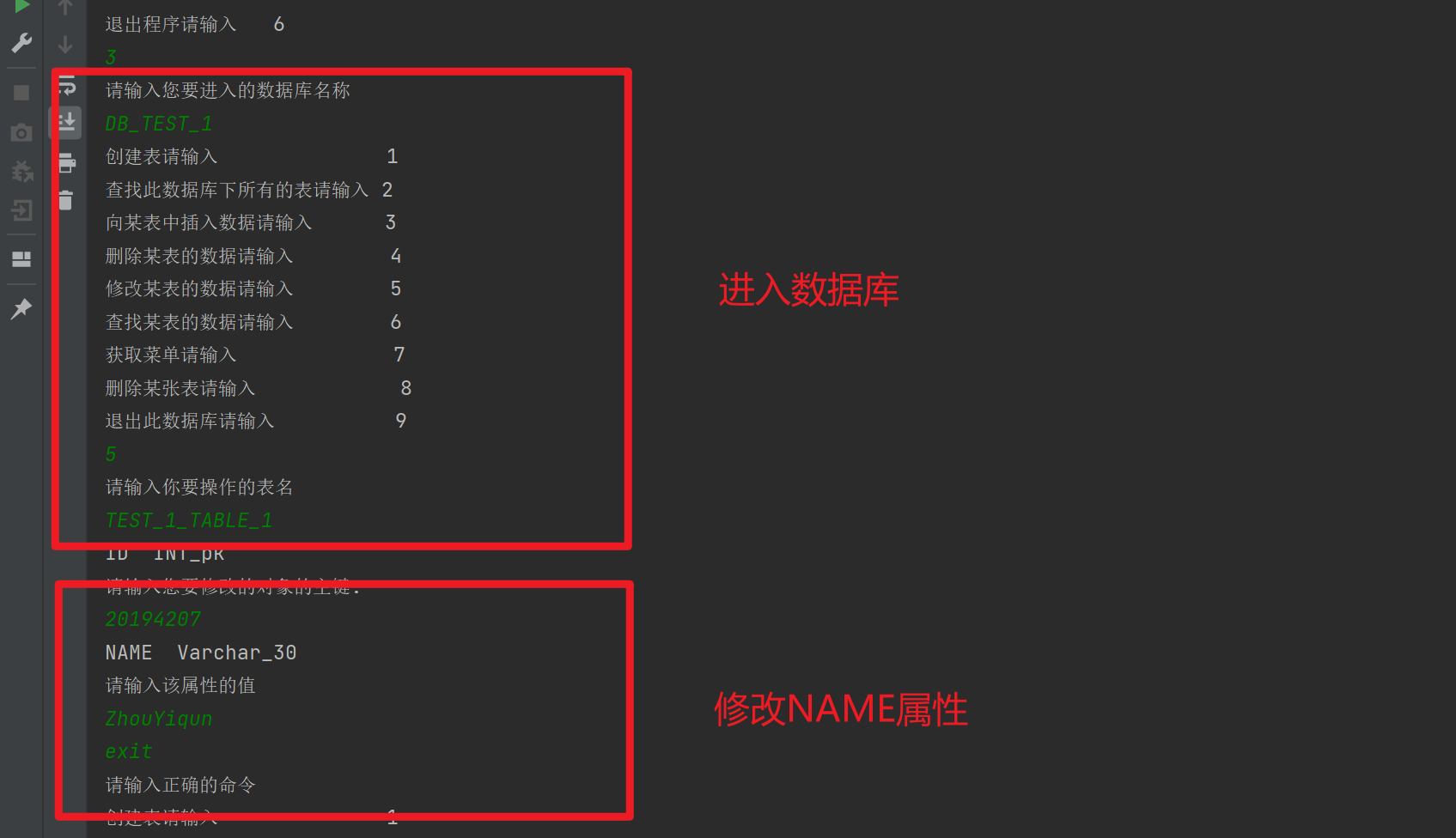
**图八：成功删除演示数据**

#### 3.1.4 修改某表的数据

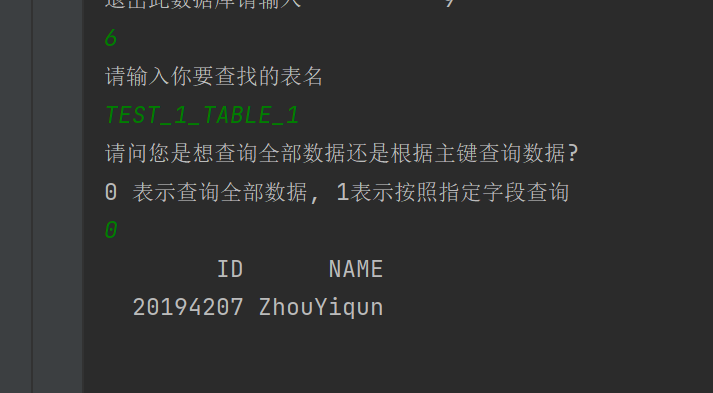
修改数据的逻辑与删除数据相似，首先进入目标数据库，找到需要修改的那条数据存储位置，即位于哪个块和行，然后调用table类的update函数。该函数实现了查找位置和更新属性值的操作，详见下面代码；

public static void update(String tableName, String pkValue, ArrayList<String> values) {  
 if (main.rManager.getRef(tableName).r.containsKey(pkValue)) {  
 String location = main.rManager.getRef(tableName).r.get(pkValue);  
 // 第一个变量记录它在哪一个块,第二个变量记录它在这个块的哪一行  
 String[] loc = location.split("\_");  
 Block temp = main.bManager.blocks.get(tableName).get(loc[0]);  
 temp.update(Integer.parseInt(loc[1]), values);  
 } else {  
 System.out.println("该对象不存在,请检查主键是否输入正确!");  
 }  
}

操作过程见下图，这里我们将TEST\_1\_TABLE\_1表中的20194207对应的姓名修改为ZhouYiqun，图2的查询结果中可以看出修改成功。



**图九：修改属性的命令行界面**

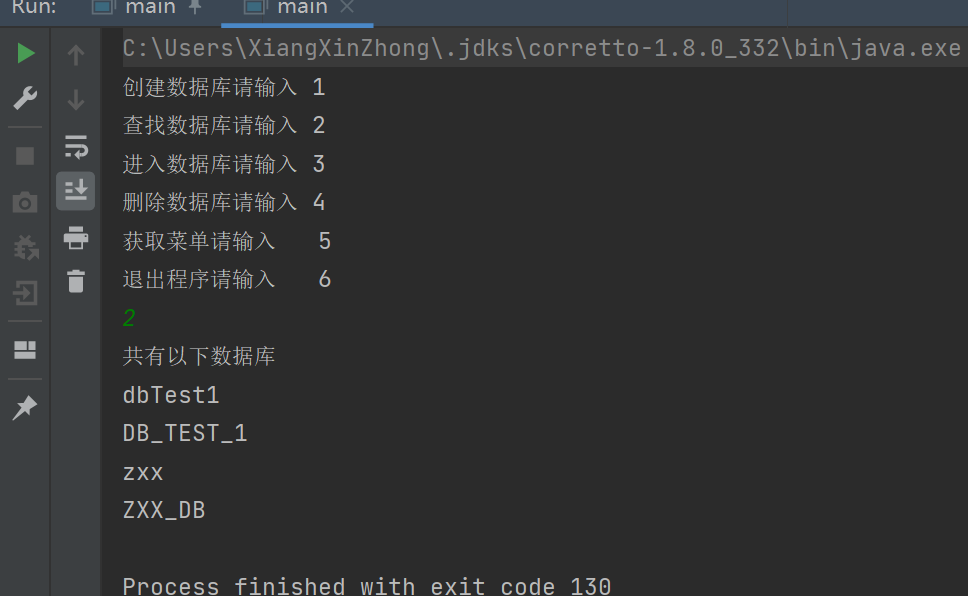


**图十：成功修改示例数据，将Zhongxiangxin修改为ZhouYiqun**

### 3.2 查找数据库

查找数据库的实现逻辑较为简单，因为数据库在文件系统中其实就是一个个文件夹，所以我们只需要在数据库的根目录下搜索文件夹，存入ArrayList中，然后打印即可。

ArrayList<String> allDB = new ArrayList<String>();  
File file = new File(rootPath);  
File[] files = file.listFiles();  
for (File f : files) {  
 if (f.isDirectory()) {  
 allDB.add(f.getName());  
 }  
}  
System.out.println("共有以下数据库");  
for (String s : allDB) {  
 System.out.println(s);  
}

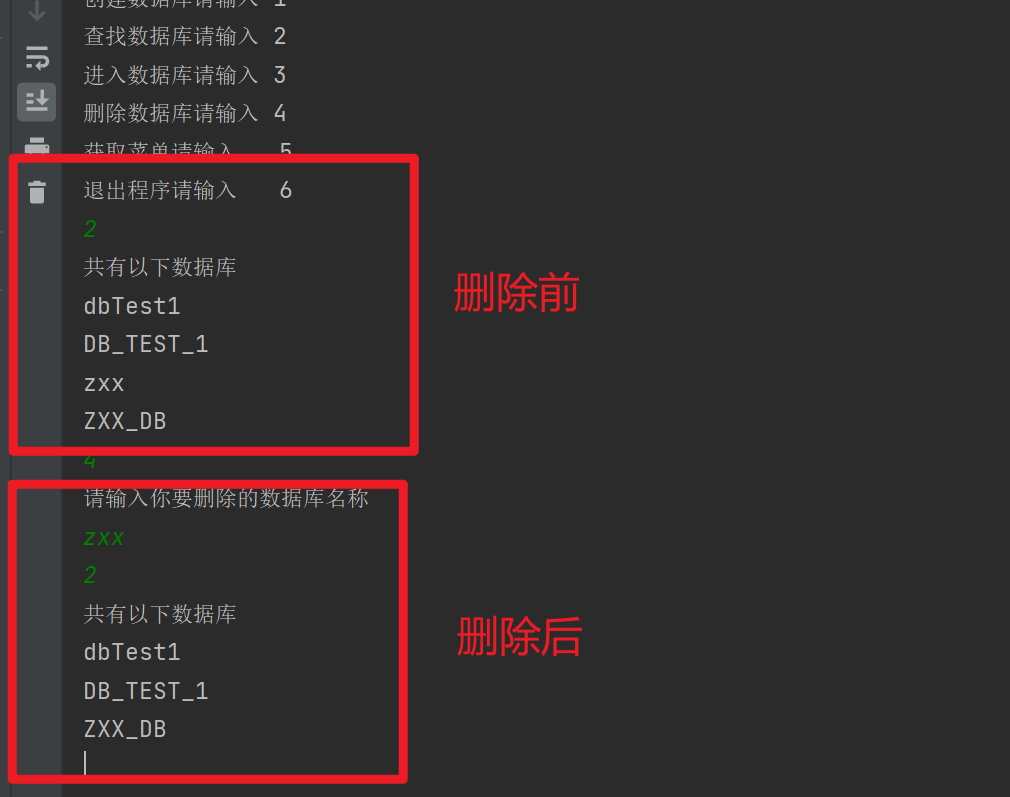


**图十一：查找所有的数据库**

### 3.3 删除数据库

和查找数据库的操作类似，首先找到该数据库位于文件系统中的位置，然后递归调用deleteDirectory函数即可删除数据库下的所有文件，就完成了数据库的删除操作。

public static void deleteDirectory(File file) {  
 if (!file.isFile()) {  
 // 首先得到当前的路径  
 String[] childFilePaths = file.list();  
 for (String childFilePath : childFilePaths) {  
 File childFile = new File(file.getAbsolutePath() + "/" + childFilePath);  
deleteDirectory(childFile);  
 }  
 }  
 file.delete();  
}



**图十二：删除名为zxx的数据库**

### 3.4索引的实现与对比

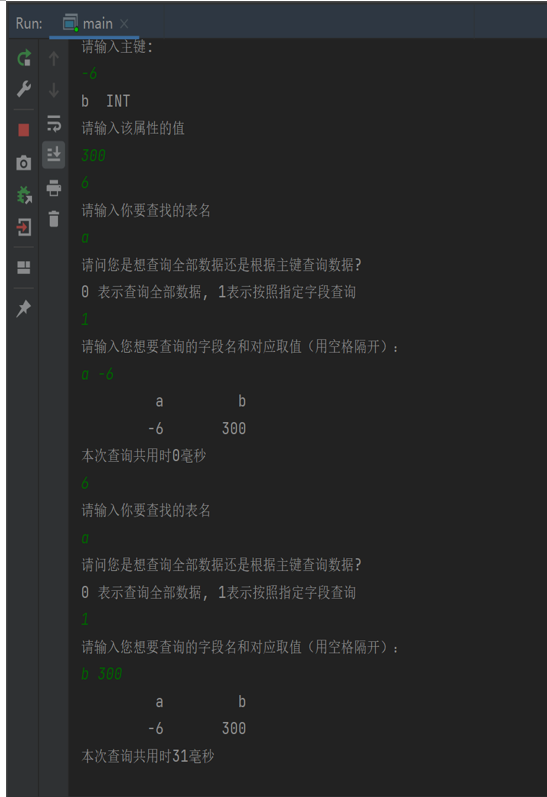
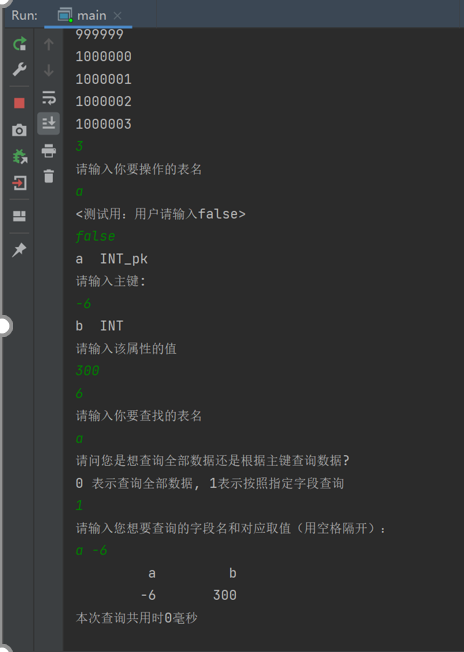
索引是一种与(数据库)文件相关联的附加结构，额外增加的一个辅助文件，可以加快数据库查找的速度。本次project我们使用Java的hashmap实现索引功能，我们为每个表建立一个索引，用主键作为搜索码。

在类refManager中我们定义了二级索引allRef，该索引中的值为当前数据库中不同表的所有索引。在ref类中为每个表的数据建立索引。

就以上述的student表为例，它有两个字段，分别是ID和name，其中ID为主键。加入我们插入该表的第一条数据（20194195，wzy），那么会形成对应于该条数据的索引，即（20194195：1.txt\_1），20194195是该数据的主键也是此索引的key，1.txt\_1是value，它表示此数据位于该表的第一个块（1.txt）中的第一个数据(\_1)。

package DataBase;  
  
import java.io.File;  
import java.util.HashMap;  
  
public class refManager {  
 public String dbName;  
 public HashMap<String, ref> allRef;  
 // 数据库中所有索引的索引  
 public refManager(String dbName) {  
 allRef = new HashMap<String, ref>();  
 this.dbName = dbName;  
 }  
  
 // 加载此数据库所有表的索引文件  
 public static refManager loadAllRef(String dbName) throws Exception {  
 refManager ret = new refManager(dbName);  
 File file = new File(dbName);  
 File[] files = file.listFiles();  
 for (File f : files) {  
 if (f.isDirectory()) {  
 ret.allRef.put(f.getName(), ref.loadRef(dbName + "\\" + f.getName() + "\\ref.txt"));  
 }  
 }  
 return ret;  
 }  
  
 // 存放所有表的索引文件  
 public static void saveAllRef(refManager rManager) throws Exception {  
 for (ref f : rManager.allRef.values()) {  
 if (f != null) {  
 ref.saveRef(f);  
 }  
 }  
 }  
  
 public ref getRef(String tableName) {  
 return allRef.get(tableName);  
 }  
}

package DataBase;  
  
import java.io.Serializable;  
import java.util.HashMap;  
  
public class ref implements Serializable {  
 public String tableName = "";  
 public HashMap<String, String> r;  
  
 public ref(String tableName) {  
 this.tableName = tableName;  
 r = new HashMap<String, String>();  
 }  
  
 public static void saveRef(ref r) throws Exception {  
 tool.saveObject(r, r.tableName + "\\ref.txt");  
 }  
  
 public static ref loadRef(String tableName) throws Exception {  
 return (ref) tool.readObject(tableName);  
 }  
  
 // 下文的id带有.txt  
 public void insert(String pk, String id) {  
 r.put(pk, id);  
 }  
  
 public String getID(String pk) {  
 if (r.get(pk) != null) {  
 return r.get(pk);  
 } else {  
 return "-1";  
 }  
 }  
}



**图十三：左图为插入1百万条数据后基于索引的查询，右图为不基于索引的查询**

本数据库系统提供了一键插入1百万条数据的性能测试脚本。为了测试性能，我们在这里建立了a数据库中的a表，包含a、b两个字段，其中a作为主键。数据库在运行过程中会自动为主键a生成索引。在测试中，我们分别对a、b两个字段执行查找操作。结果显示对主键a进行查找只需接近0毫秒，而对没有索引的字段b查找需要31毫秒。由此体现了反映了索引对数据库查询效率的提升

## 4 项目总结

通过此次项目，我们小组成员都对底层数据库的实现有了可以说是非常清晰的认识，提升了我们的代码编写能力和项目测试、调试能力，并让我们深刻地认识到索引文件带来的查询效率的巨大提升。

对于我们实现的数据库，我觉得主要**有以下优点**：

1. 使用java语言编写，**具有较高的执行效率和程序易读性**。我们与其他组交流的过程中发现他们很多是用python语言编写，并用json的格式存放数据，json格式存的是utf编码的明文数据，同等信息下它所占的大小会明显大于二进制文件。它可被直接读取，也意味着其安全性没有用java序列化得到的二进制文件好。
2. 使用交互式地操作方式**对用户更亲和**，用户可以在完全不懂sql语句的情况下即可轻松使用该数据库。
3. 代码量少而精，我们设计的**数据库模型逻辑简洁合理且性能优异**，能够经受住百万级别甚至更高量级的数据存储及访问，基于索引文件，我们可以在不到1ms的时间里精确定位到需要操作的数据。

但与此同时，我们认为此数据库也有以下待改进的地方

1. 不支持基于属性的带范围的查找，只支持输入属性的值并据此查找到单个的数据。
2. 未实现日志功能
3. 未实现数据库角色，数据库管理员，视图等身份概念

做一个数据库实在是有趣且具有重大意义的事情，关于上述不足之处的实现方式我们也都有眉目，但由于时间和精力的关系未能实现，以后若从事相关工作，再仔细研究。