浙江大学

数据库系统实验报告

MiniSQL详细设计报告

---Interpreter,GUI,BufferManager

姓　　名：黎睿翔

学　　号：3180106171

电子邮箱：3180106171@zju.edu.cn

联系电话：18888915594

指导老师：陈刚

目录

一、实验目的

设计并实现一个精简型单用户SQL 引擎(DBMS)MiniSQL，允许用户通过字符界面输入SQL语句实现表的建立/删除；索引的建立/删除以及表记录的插入/删除/查找。通过对MiniSQL的设计与实现，提高学生的系统编程能力，加深对数据库系统原理的理解。

本人参与设计的模块包括Interpreter、 BufferManager、 GUI(作为bonus)。

二、实验环境

采用Java进行编程。个人选择的操作系统为Windows10，IDE是IntelliJ IDEA Community Edition 2020.1.2 x64，java版本“14.0.1”。

三、模块设计

3.1 Interpreter

Interpreter的基础功能包括将用户的输入命令进行语法分析和语义解析并得到需要的命令参数,最后将该命令参数封装成对应命令的参数类对象,传到API模块；同时对于 API 返回的操作结果进行输出显示。

下面对Interpreter模块中设计的类和相关数据结构、成员函数作一个简要陈述。

词法解析

工具：Lexer 类

	名称	功能描述
成员变量	char peek	下一个读入字符
	Hashtable<String, Word> words	关键字存储
	BufferedReader reader	作为读入的文本
	Boolean isReaderEnd	判断当前是否读取到了文件的结尾

外部接口	Lexer(BufferedReader reader)	构造函数，初始化关键字，将关键字存进哈希表
外	Boolean getReaderState()	是否读取到输入流的结尾
口	Token scan()	读取字节流，返回 token，可通过 token 标签判断单词类型

相关申明的其他类

类名称	作用
Token	符文，用标签 tag 分类
Comparison 继承 Token	操作符 (<,<=,>,>=,=,<>)
Num 继承 Token	数字
Word 继承 Token	单词
Tag	Token的标签（所有关键字、 STR———字符窜INTNUM ———整数FLOATNUM———浮点数 TYPE———字段类型OP———操作符 ID———表名、索引名或字段名）

语法和语义分析

工具: Interpreter类

	名称	作用
类量成员变量	Token thetoken	下一个读入符
	boolean isSynCorrect=true;	标记当前语句是否语法正确，初始化为真
	boolean isSemaCorrect=true;	标记当前语句是否语义正确，初始化为真
	String synErrMsg;	记录当前语法错误信息
	String semaErrMsg;	记录当前语义错误信息
	long startTime	记录当前语句开始时间
	long endTime	记录当前语句结束时间
	Parsing(BufferedReader reader)	用于对输入内容逐条分析语句
	void showSelectRes (String tmpTable, Vector<String>	根据结果按照格式输出select语句的内容，tmpTable对应当前语句的表，

外部接口	tmpAttriNames, conditionNode tmpConditionNode, String tmpOrderAttriName, boolean order)	tmpAttriNames对应当前语句需要查询的类型，对应于投影操作，tmpCondetionNode对是一个表示查找条件的对象，由RecordManager产生，tmpOrderAttriName,order则是关于排序的，如果需要排序则order为true。
	Vector<String> ParsingProjection(Lexer lexer)	对project投影部分进行解析。
	conditionNode ParsingExpression(Lexer lexer,String tmpTableName)	判断条件语义错误的，具体为值属性错误和属性间比较错误判断。
	conditionNode ParsingCondition(Lexer lexer,String tmpTableName,String endtoken)	对条件部分字符串进行解析，针对相关于一错误，包括属性名不存在，value格式不对和操作符不符合。

语法分析

判断方法：通过状态机来实现状态跳转。

在读取到 lexer.getReaderState()==false(输入流末尾)前，循环读取lexer.scan()函数返回每个 token 标签进行判断，来决定下一个要进入的状态。符合正确语法的跳转到下一状态，不符合正确语法时，记录语法错误信息，将语法标记为假，同时跳过本轮循环。在下一轮循环开始前，将会输出语法错误信息，并将语法标记重新置为真。以分号作为语句的分隔。

支持语句

创建表	create table 表名(属性1,属性2,…,属性n, primary key(属性名)); 属性申明方式: 属性名 类型 (primary key) 支持类型: (int,float,char(n) (1<=n<=255)).
删除表	drop table 表名字;
创建索引	create index 索引名 on 表名 (属性名)
删除索引	drop index 索引名;

元组插入	insert into 表名 values (属性值1,属性值2,…,属性值n) (其中字符串用''或"")
元组删除	delete from 表名 where 条件 其中"条件"具有以下格式：表达式的 与或，支持括号优先级。 表达式中运算符包括<>,=,>,<,>=,<=
元组查询	支持的select语句: ① select *|(列名*) from 表名 ; ② select *|(列名*) from 表名 where 条件 ; ③ select *|(列名*) from 表名 where 条件 order by 列名; ④ select * from 表名1 join 表名2 where 表名1.列名=表名2.列名; 条件格式同delete语句。
退出系统	quit;
SQL脚本语句	execfile 完整路径+文件名;

语义分析

判断方法：在当前状态为语法正确的基础上调用CatalogManager的接口对表和索引信息进行查询并进行判断。判断为错误时，记录当前语义错误信息，将语义标记为假。在本条语句语法完全基础上，判断语法标记，若为真，调用执行命令语句，若为假，输出语义错误信息，并将语义标记重新置为假。

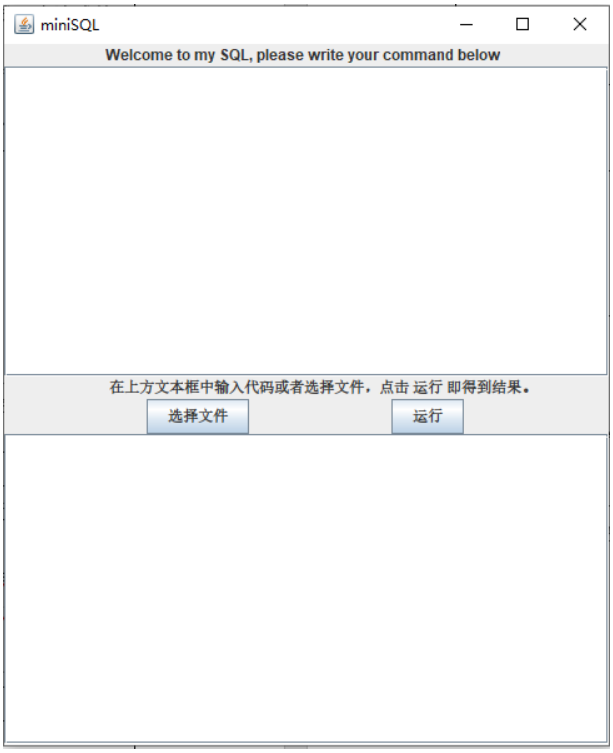
语句类型	语义错误
create table	table name 已存在 primary key 所指字段不存在 出现重复的 attribute 字段 char(n) 的 n 越界
drop table	table name 不存在
create index	index name 已存在 table name 不存在 attribute 不存在 attribute 已经是索引

	attribute 不是 unique
drop index	该 index name 不存在 该 index name 是主键不能删除
insert	table 不存在 插入的 tuple 数量不对 插入的 tuple 类型（及长度）不对 对于 unique key 字段有重复插入记录
delete	table 不存在 where 条件有误 ： 字段名不存在; value 属性与字段属性不匹配
select	table 不存在 where 条件有误: 字段名不存在; value 属性与字段属性不匹配 select 或 order 的字段名不存在 两字段属性不同无法比较

与其他模块的交互：与Interpreter交互的模块如下，在获取输入和实现输出方面是与GUI交互，在实现相关语句如select等语句与API交互，在进行逻辑信息处理的时候主要与RecordManager交互，在对表处理时，直接与CatalogManager交互。

3.2 GUI

基于interpreter的GUI设计。GUI样式如下:



功能介绍: 上方文本框用于输入对应sql代码，下方文本框用于打印相关信息。选择文件按钮可以实现选择本机中的文件，将内容拷贝到上方文本框中，运行按钮则是根据上方文本框的sql代码运行将运行输出打印到下方文本框中。

GUI类的成员变量和外部接口:

	名称	功能描述	实现原理
成员变量	static int FirstWrite0, FirstWrite1;	用于判断第一个文本框、第二个文本框是不是第一次打印。	
	static JTextArea readArea, writeArea	分别对应输入的文本框和输出的文本框。	
	void Initialize()	创建并初始化顶层容器，设定对应参数，以显示GUI。	调用swing库中的函数以及GUImake函数

外部接口	void GUImake(JFrame frame)	在顶层容器frame中添加内容，包括文本框，按钮等。	调用swing库中的button类、scrollpane类、box类、JTextArea类。
	void FilePrintText(File file, int type)	将对应文件内容打印到GUI的文本框中，由于有两个文本框，用type加以标识。	调用JTextArea类中的函数，将内容打印到对应文本框中。
	void StringPrintText (String s)	将数据库中对应语句打印到GUI中的输出文本框中。	同上，主要是调用了JTextArea类中的append函数。
	void StringPrintAttribute (String s)	在select语句中，由于输出的属性是不需要输出即换行的，与StringPrint Text函数产生了冲突，因此单独设计了StringPrintAttribute函数加以处理。	同上。
	BufferedReader GetText (int type)	与Interpreter交互的函数，在Interpreter中sql语句内容采用BufferReader流方式读入，将JTextArea中的文本内容转换为BufferedReader格式。	首先调用了JTextArea的getText函数，得到String类型的内容，再通过将其转换为ByteArrayInputStream类然后转换为BufferedReader类。

与其他模块的交互：GUI基本上都是和Interpreter交互的，为其提供输入，Interpreter的输出需要输出到GUI中。

3.3 BufferManager

BUFFERMANAGER包中包括两个类，Block类和BufferManager。其中Block块是用来记录块内4KB大小数据以及标记位的类。BufferManager是用于管理buffer的类，对应的成员变量，接口如下：

Block类

	名称	功能描述	实现原理
成员变量	byte[] data	数据区，4KB 大小	
	String filename	记录块所属文件名	
	int blockoffset	记录这个块属于这个文件 的第几个块	
	boolean dirty	是否脏数据	
	boolean valid	有效位	
	boolean fixed	是否被锁定	
	boolean reference_bit	引用位，用于 LRU 算法	
外部接口	byte[] readData()	用于读出 4KB 数据	返回内部成员 data，将引 用位置 1
	boolean writeData (int byteoffset, byte inputdata[], int size)	用于将 inputdata[]中的数据写入块中	写入数据后，将引用位置1，dirty 位置 1 以标记为脏数据
	boolean writeData()	用于在直接修改 data 之后发出脏数据信号及引用位 信号	写入数据后，将引用位置 1，dirty 位置 1 以标记为脏数据
	void fix()	把块锁定在缓冲区	fix 位置 1 以锁定
	void unfix()	把块从缓冲区解锁	fix 位置 0 以解锁
	int readInt(int offset)	从块中的指定位置读出一 个整数	读出数据后，将引用位置 1
外部接口	void writeInt(int offset, in t num)	从块中的指定位置写入一个整数	写入数据后，将引用位置1，dirty 位置 1 以标记为脏数据
	float readFloat(int offset)	从块中的指定位置读出一 个 float	读出数据后，将引用位置 1

	void writeFloat(int offset, float num)	从块中的指定位置写入一个 float	写入数据后，将引用位置 1，dirty 位置 1 以标记为脏数据
	String readString(int offse t, int length)	从块中的指定位置读出一个长度为 length 的 String	读出数据后，将引用位置 1
	void writeString(int offset, String num, int length)	从块中的指定位置写入一个长度为 length 的 String	写入数据后，将引用位置 1，dirty 位置 1 以标记为脏数据

BufferManager类

	名称	功能描述	实现原理
成员变量	Block[] blocks	用一个 block 类数组代表 b uffer,总共占用 80k 的空间	
	int pointer	用于实现时钟算法，用块 在 buffer 中的下标代替指针	
外部接口	void initialize()	初始化。在使用 Buffer Ma nager 前调用一次	为 buffer 申请内存空间
	void close()	关闭 Buffer Manager，在退 出程序之前调用	把 Buffer 中的脏数据写回 文件
	Block getBlock(String filen ame, int blockoffset)	给定文件名和块编号，返回一个 block	调用 findBlock 搜索这个块是否在 buffer 中，是则返回这个 block。否则，调用 getFreeBlockNum 得到 buff er 中可用的一个块下标， 把文件中的数

			据读入这个 块，并返回这个块。
内部函数	int findBlock(String filena me, int blockoffset)	给定文件名和块编号，返回一个block 在 buffer 中的下标，如果不在 buffer 中 则返回-1	对 buffer 中的所有块进行遍历搜索。
	int getFreeBlockNum()	返回一个可被替换出去的block 的下标。过程中使用时钟算法进行选择，并且跳过被锁定在 buffer 中的块。	将 pointer 指向 buffer 中下一个块。如果且 fixed 为 1， 将 pointer 指向下一个块并进入下一个循环。否则， 如果指向的块 reference_bi t 为 1，则把 reference_bit 位置 1；如果指向的块 refe rence_bit 为 0，则把这个块写回文件，并返回 pointer 的值。具体算法见后文。
	boolean readFromDisk(String filename, int blockoffset, int num)	给定文件名，块偏移，把数据从文件读取到 buffer 中下标为 num 的块中，并对标记位进行初始化（有效位、reference_bit 置 1，dirty、fixed 位置 0)	通过 RandomAccessFile 执行文件读取

	void writeToDisk(int num)	把 buffer 中下标为 num 的块写回到文件中，并把块的有效位置 0	如果 dirty 位为 0，则不执行写操作，否则执行写操作。

与其他模块的交互：BufferManager模块主要是和其他的三个部分Index、Record、Catalog以及文件处理File这4个Manager进行交互的，将数据写到buffer中，需要替换时则将数据写到硬盘中。

四、具体实现细节

4.1 Interpreter

对输入的处理：

输入我们采用的是java.io中的BufferedReader进行读入的，在最初的没有GUI的版本中，BufferedReader流来自于System.in和文件内容；在后面加入GUI之后，BufferedReader流来自于GUI的文本框（文本框中内容可以是来自文件也可以是输入的）。

关于对BufferedReader的处理，采用的是逐字节读入的方式，由此引入了Lexer类。下面是Lexer类中的成员变量。

```
char peek = ' ';        /* 下一个读入字符 */
Hashtable<String, Word> words =
       new Hashtable<String, Word>();

BufferedReader reader = null;
/* 保存当前是否读取到了文件的结尾  */
private Boolean isReaderEnd = false;
```

peek为读入的字符，采用BufferedReader的read函数读取。words为一个hash表，在构造函数中对该hash表进行了初始化，用途是将sql语句中的单词转化为对应Tag。reader就是在构造函数中构造的BufferedReader流。isReadEnd是用来判定输入结束的。

事实上，Lexer是用来对BufferedReader处理的类，我们在Interpreter的Parsing函数中是通过对当前字符的Token处理得到的。Token实际上是一个标记，根据之前的读入以及当前的读入，它有一个int类型的标记值，从而判断它是一个关键字，或者一个单词或者一个数字或者操作符。这个工作是在Lexer.scan函数中实现的。首先忽略\t\n空格这些无用字符，然后依次进行操作符分

析，数字分析（读到一个字符是数字后就把后面所有的数字都读进来处理，这里用到了一个继承类Num），字符串分析（读到'开始，直到'结束，这里用到了一个继承类Word），关键字或标识符分析(对于关键字，根据开始初始化的hash表转化为Word对象，如果不是关键字，就是类型名，对应构造一个新的Word对象)。

语法分析与语义分析：

这个部分主要来自于Interpreter的parsing函数，在Interpreter类中，有两个成员变量

```
private static boolean isSynCorrect=true;
private static boolean isSemaCorrect=true;
```

这两个变量用来判断是语法错误和语义错误。对于每一条语句,只有当读取到分号并且上述两个变量都为真时，才会执行对应语句，不然就会报错。

这个函数有将近1k行，但是内容还是比较容易理解的，主要就是条件语句的使用，其中对应的语义分析我在第三部分已经有写到了，每类语句单独判断。首先用一个scan读取关键字，然后根据语法依次读取，并记录相关内容，由于在判定语义错误时，需要检查比如表格是否存在，是否重名，unique内容是否相同，因此需要直接调用CatalogManager和RecordManager的相关成员函数。在执行语句的时候就是调用的API提供的接口。

这个函数难度不大，也很容易理解，就是比较繁琐，每种语义正确判断要单独处理。

parsing函数实现的是基础功能，在后面我们又添加了投影、or等相关操作，判断起来更复杂了，于是在Interpreter类中添加了对select投影的判断，对应于函数ParsingCondition，对where后面表达式的处理，对应于函数ParsingCondition(ParsingCondition又涉及到了字符串的处理)，对应于函数ParsingExpression。

语句时间测定：

在Interpreter类中调用了startTime和endTime两个long类型变量，然后在parsing的开始记录startTime，只有在语句正确的时候才会记录时间，语句输出结果时，记录endTime，计算该语句的运行时间即可。输出运行时间后，重新记录startTime。

下面是运行时间的计算方法:

```
String duration =
        String.format("%.2f",(double)(endTime-startTime) /1000.0);
```

记录时间用的是System.currentTimeMillis。

和GUI的交互：

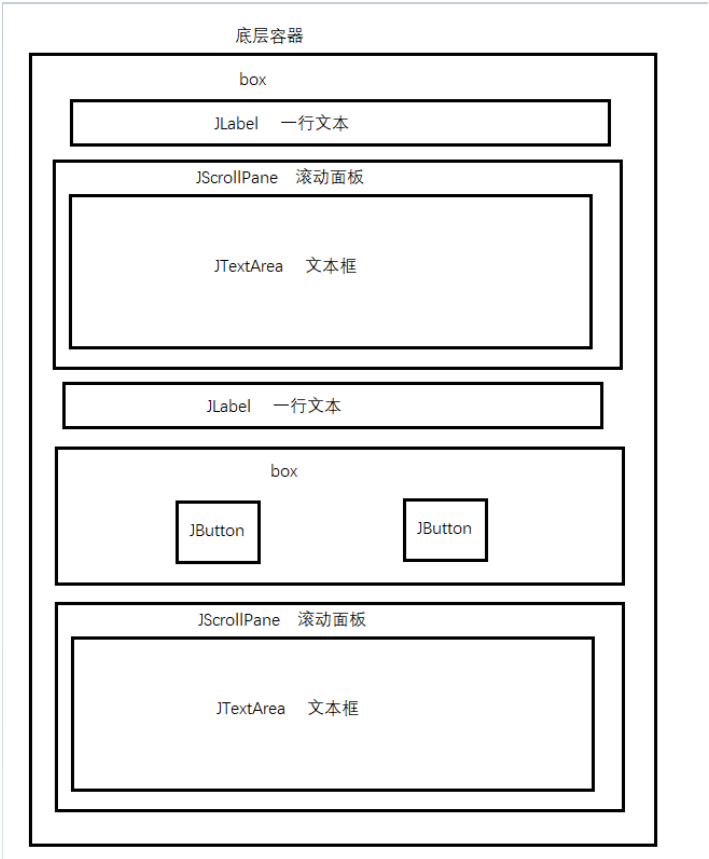
在没用GUI的版本中，输出采用的是System.out.println或者System.out.print函数，将这些函数替换为GUI.StringPrintText即可。其中在输出查询结果的时候，首先要输出表格属性，在这里输出与该函数不是很兼容，因为该函数输出时自动换行，因此专门为表格属性输出涉及了GUI.StringPrintAttribute函数。

4.2 GUI

初始化界面：

GUI设计采用的是swing库，swing库的设计是有底层容器，然后往上面添加中间容器，中间容器中添加组件形成的。从界面可以分析，我们的GUI是由两行文本，两个文本框和两个按钮构成的。

其中文本框、按钮、文本均为基本组件，采用中间容器box对界面进行排版，两个按钮横向排列，采用横向的box进行存储，也就是一个box套box的结构。而文本框最后呈现的是可滚动界面，调用到了JScrollPane这个中间容器，向JScrollPane容器中添加JTextArea，以实现文本框的滚动，最终实现了GUI的设计，对应图如下:



按钮监听器：

第一个按钮是用来选取文件的，采用的是监听器addActionListener，按下后产生一个新窗口Filechooser，选择对应文件，如果不选择文件则报错，提示没有任何文件。相关源码如下：

```
button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        JFileChooser fc = new JFileChooser();
        String defaultDirectory = "\\\\Mac\\Home\\Desktop\\tests";
        fc.setCurrentDirectory(new File(defaultDirectory));
        fc.showOpenDialog(frame);
        try{
            File file = fc.getSelectedFile();
            FilePrintText(file,0);
        } catch(Exception e1){
            JPanel panel1 = new JPanel();
            JOptionPane.showMessageDialog(panel1,"没有选择任何文件","
提示",JOptionPane.WARNING_MESSAGE);
        }
    }
});
```

此外运行按钮同样处理，设置监听器，然后对上面文本框的信息进行处理。

文件处理：

由于Filechooser选择的文件是File格式，而文本框中的内容是String格式，在Interpreter.parsing中采用的是BufferedReader格式。因此在输入输出过程中需要对应类型的转换。

将文件内容输出到文本框中(FilePrintText函数)是现根据File文件的路径转换为FileInputStream，然后转换为InputStreamReader，再转换为BufferedReader。然后对BufferedReader执行readline操作循环，用swing.JTextArea.append将其添加到文本框后面。

获取文本框内容(GetText函数)是将文本框中的内容转换为BufferedReader供Interpreter.parsing使用。实现方式是先将其转换ByteArrayInputStream，然后转换为InputStreamReader进而转换为BufferedReader。具体代码如下：

中文处理：

由于采用swing编写的GUI的编码方式是GBK，而java的默认编码方式是UTF-8，因此在导入含有中文的文件时，会产生乱码。

解决方法：在文件处理部分我提到了内容的相互转化，在转化过程中只需

要添加编码方式转化即可，在FileInputText函数中，对应代码如下：

```
reader = new BufferedReader(new InputStreamReader(new FileInputStream
(path),"UTF-8"));
```

在GetText函数中，对应代码如下：

```
InputStream is = new ByteArrayInputStream(content.getBytes("GBK"));
BufferedReader reader = new BufferedReader(new InputStreamReader
(is));
```

输出到文本框中：

输出到文本框中是不会自动换行的，为了解决这个问题，需要在输出之前判定是否是第一次输出，如果不是第一次输出，需要现输出一个换行符再输出。因此定义了GUI类的成员变量，FirstWrite0和FirstWrite1用于判断两个文本框是不是第一次输出。

二次输入问题的解决：

在GUI的初始版本中，发现了一个问题，在第一次运行后产生的结果是正确的，可在第二次运行后产生的结果就不正确了。后面发现是第二次以后的每次运行结果只有第一句话运行不正确。报错结果为语法错误，错误的原因是一个诡异的方框。

在经过GUI和Interpreter的综合分析后，发现原因是第二次以后的每次读入都会读入到上一次的最后一个字符0xFFFF，所以第一句话总是不对的。

解决方案：第一句话去掉第一个字符。

4.3 BufferManager

Block的相关处理：

设定一个Block的大小为4096B，即4KB。由于采用LRU算法处理Buffer，因此需要类似于计算机组成cache原理类似。设定引用位，脏位，有效位，锁定位。定义语句如下：

```
    public static final int BLOCKSIZE = 4096;
    static final boolean EMPTY = true;

    String filename="";//记录块所属文件名
    public int blockoffset=0;//记录这个块属于这个文件的第几个块
    boolean dirty = false;//是否脏数据
    public boolean valid = false;//有效位
    boolean fixed = false;//是否被锁定
    boolean reference_bit = false;//引用位，用于 clock 算法

    public byte[] data = new byte[BLOCKSIZE];//数据区，4KB 大小
```

对应的函数，readData读取数据，返回数据的同时引用位置1。writeData写入操作，从指定位置写入给定的内容，由于java中有数据结构byte[]，操作起来是非常容易的。引用位置1，脏位置1。由于在MiniSQL中只有三种数据类型int、float和char[]，因此处理起来还是很容易的，32位的int、float对应存在连续的4个bytes中，而字符串可以直接调用java的btye[]与String相互转换函数。

BufferManager的相关操作：

定义BufferManager大小为80KB，即20个Block。

首先是初始化和结束处理函数，初始化调用Block的构造函数即可，结束处理则是将各个block写到硬盘中。

然后是对block的相应操作。

1）删除block，直接将需要删除的block有效位置0即可。

2）查找对应文件filename第blockoffset块，如果该块在Buffer中存在（有效位为1）则返回该块；如果不存在，则首先利用时钟算法找到第一个可以用的块，如果文件不存在，则建立一个新block，内容为0，否则将对应文件的块取出来放到新block中。这些对应了getBlock函数和findBlock函数。

时钟算法的实现（getFreeBlockNum）：在置换策略中，我们选择了时钟算法，即当某一页装入主存时，将use bit置成1；如果该页之后又被访问到，使用位也还是标记成1。对于页面置换算法，候选的帧集合可以看成是一个循环缓冲区，并且有一个指针和缓冲区相关联。遇到页面替换时，指针指向缓冲区的下一帧。如果这页进入主存后发现没有空余的帧(frame)，即所有页面的使用位均为1，那么这时候从指针开始循环一个缓冲区，将之前的使用位都清0，并且留在最初的位置上，换出该帧对应的页。实际运行的话就是一个循环。

文件操作：

由于Block中有一个成员变量filename，用于记录该block存储的文件名，因此读取和储存都挺容易的。本模块中调用java相关文件操作的接口实现将Buffer的Block存到文件中和读取文件到Buffer。

读取操作：其实也很容易，就是把需要读入数据的block初始化，然后读取文件的对应块，如果读取失败就为0，采用的是RandomAccessFile类。

写入操作：也是用RamdomAccessFile类的函数，需要注意，如果脏位为0，不需要执行。相关文件操作的代码

```
raf.seek(blocks[num].blockoffset * Block.BLOCKSIZE);
raf.write(blocks[num].data);
```

读取就把write改成read就行。

五、总结

本次项目难度还是很大的，基本上在ddl的最后一段时间都每天都挺晚的，相关模块的交互使得debug难度增大了。这个MiniSQL是用java写的，我个人不是很擅长java，但是在查阅各种资料并和组长王俊同学交流后，我最终还是完成了我个人的部分。我个人主要是写了BufferManager模块和一个bonus GUI，此外和王俊同学共同完成了Interpreter，之后我完成了报告的测试部分。关于BufferManager，主要是对Buffer的管理，用到了在实现方面比较容易的时钟算法，此外关于写入和读取操作都还是比较容易实现的，这个部分也主要是对DB文件处理的理解和对计算机系统的理解。在Interpreter方面，我配合王俊共同完成了该部分，我自己完成了一个GUI，和他共同调试运行，解决了很多问题包括读入异常、输出异常、支持中文等。

当然我想从我的部分谈一下我们的工程还有一些可以改进的地方，首先是语言选择方面，选用java在处理byte等方面可以简化代码，降低代码量，但是在性能上不如C++，从性能上考量可以选择C++。在BufferManager部分，我可以改用LRU算法或者其他置换算法来提高效率。在Interpreter方面，还存在小规模的bug，比如在添加过多内容后在进行操作会报错，不过这也是可以避免的。另外是添加的bonus中join操作和其他操作并不兼容，这个需要进一步的改进。在GUI方面，由于是第一次使用 swing库，写的GUI比较简陋，字体也比较小，而会出现文本框大小变化（在最大化时）的问题。

总体来说，我们的项目还是不错的，有GUI支持比命令行好多了，外加一些额外的关键字，也为我们的项目锦上添花。感谢我的小组成员这段时间的陪伴，让我们能够共同完成这个具有挑战性的project。