MiniSQL

【设计说明书】

Author: 陈锰 刘邵轩 赵子瑜

Date: Jun.24,2019

目录

1 项目背景

1.1 项目概述

基于数据库原理和数据库设计相关知识，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。

1.2 项目意义

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，在实践中学习和应用数据库设计的基本方法。

2 需求分析

2.1 功能需求

2.1.1 需求概述

- **数据类型：**只要求支持三种基本数据类型：int,char(n),float,其中 char(n) 满足 1 <= n <= 255 。
- **表定义：**一个表最多可以定义 32 个属性，各属性可以指定是否为 unique；支持单属性的主键定义。
- **索引的建立和删除：**对于表的主属性自动建立 B+树索引，对于声明为 unique 的属性可以通过 SQL 语句由用户指定建立/删除 B+树索引（因此，所有的 B+树索引都是单属性单值的）。
- **查找记录：**可以通过指定用 and 连接的多个条件进行查询，支持等值查询和区间查询。

✧ **插入和删除记录：**支持每次一条记录的插入操作；支持每次一条或多条记录的删除操作。

2.1.2 语法说明

MiniSQL 支持标准的 SQL 语句格式，每一条 SQL 语句以分号结尾，一条 SQL 语句可写在一行或多行。为简化编程，要求所有的关键字都为小写。在以下语句的语法说明中，用黑体显示的部分表示语句中的原始字符串，如 **create** 就严格的表示字符串“create”，否则含有特殊的含义，如 表名 并不是表示字符串 “表名”，而是表示表的名称。

➢ **创建表语句**

```
create table 表名 (
    列名 类型 ,
    列名 类型 ,

    列名 类型 ,
    primary key ( 列名 )
);
```

若该语句执行成功，输出执行成功信息；若失败，必须告诉用户失败的原因。

➢ **删除标语句**

```
drop table 表名 ;
```

若该语句执行成功，输出执行成功信息；若失败，必须告诉用户失败的原因。

➢ **创建索引语句**

```
create index 索引名 on 表名 ( 列名 );
```

若该语句执行成功，输出执行成功信息；若失败，必须告诉用户失败的原因。

➢ **删除索引语句**

```
drop index 索引名 ;
```

若该语句执行成功，输出执行成功信息；若失败，必须告诉用户失败的原因。

- **选择语句**

```
select * from 表名 ;
//or
select * from 表名 where 条件 ;
```

其中“条件”具有以下格式：列 op 值 and 列 op 值 … and 列 op 值。

op 是算术比较符：= <> < > <= >=

若该语句执行成功且查询结果不为空，则按行输出查询结果，第一行为属性名，其余每一行表示一条记录；若查询结果为空，则输出信息告诉用户查询结果为空；若失败，必须告诉用户失败的原因。

- **插入记录语句**

```
insert into 表名 values ( 值1 , 值2 , … , 值n );
```

若该语句执行成功，输出执行成功信息；若失败，必须告诉用户失败的原因。

- **删除记录语句**

```
delete from 表名 ;
//or
delete from 表名 where 条件 ;
```

若该语句执行成功，则输出执行成功信息，其中包括删除的记录数；若失败，必须告诉用户失败的原因。

- **退出系统语句**

```
quit;
```

- **执行脚本文件语句**

```
execfile 文件名 ;
```

SQL 脚本文件中可以包含任意多条上述 8 种 SQL 语句，MiniSQL 系统读入该文件，然后按序依次逐条执行脚本中的 SQL 语句。

2.2 性能需求

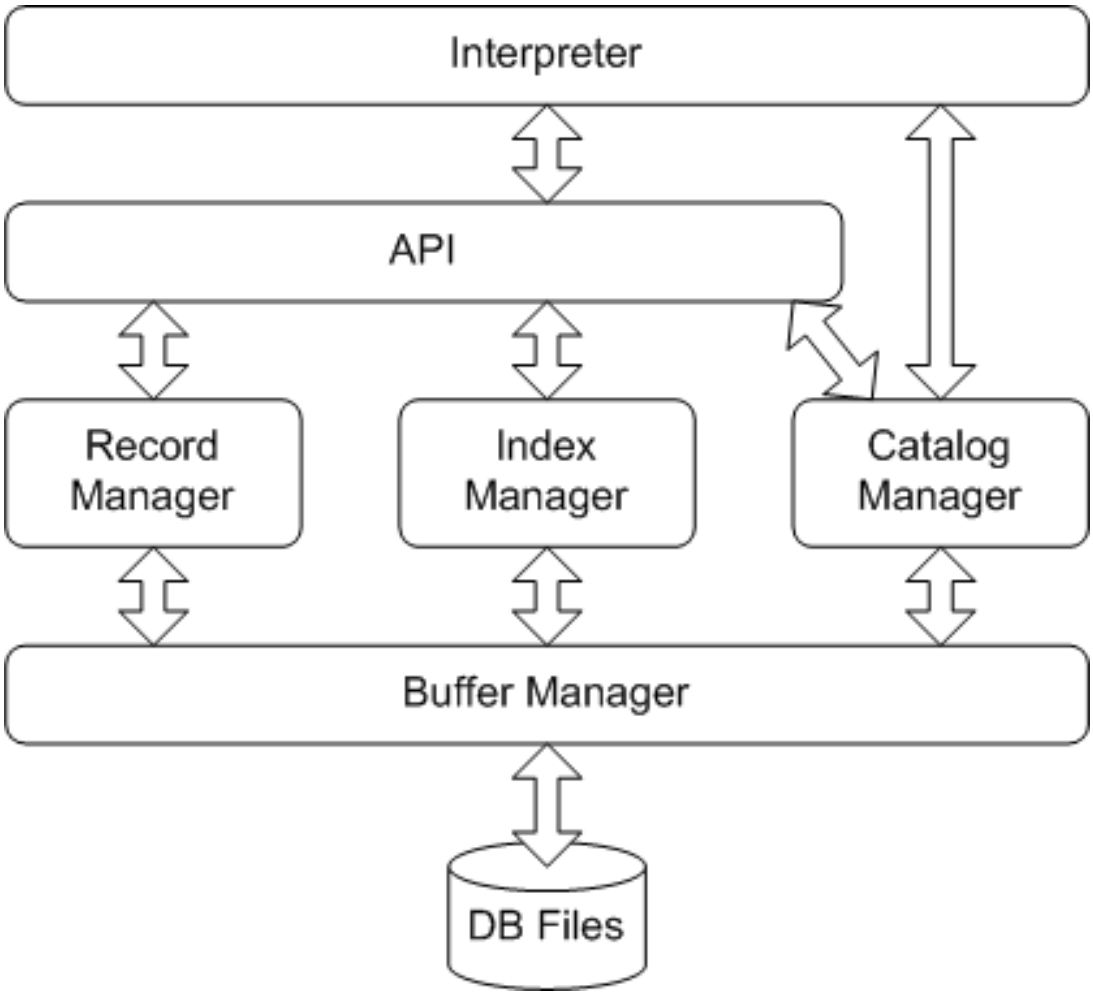
- 单表属性：支持最多 32 个

- char(n)类型：支持范围[1，255]
- B+树索引：支持单属性单值索引
- 查询/删除：支持复合条件操作，支持等值和区间查询
- 插入记录：支持一次性操作至少 5000 条

3 项目设计

3.1 总体设计

3.1.1 系统架构



✧ Interpreter

Interpreter 模块直接与用户交互，主要实现以下功能：

1. 程序流程控制，即“启动并初始化 → ‘接收命令、处理命令、显示命令结果’循环 → 退出”流程。
2. 接收并解释用户输入的命令，生成命令的内部数据结构表示，同时检查命令的语法正确性和语义正确性，对正确的命令调用 API 层提供的函数执行并显示执行结果，对不正确的命令显示错误信息。

✧ API

API 模块是整个系统的核心，其主要功能为提供执行 SQL 语句的接口，供 Interpreter 层调用。该接口以 Interpreter 层解释生成的命令内部表示为输入，根据 Catalog Manager 提供的信息确定执行规则，并调用 Record Manager、Index Manager 和 Catalog Manager 提供的相应接口进行执行，最后返回执行结果给 Interpreter 模块。

✧ Catalog Manager

Catalog Manager 负责管理数据库的所有模式信息，包括：

1. 数据库中所有表的定义信息，包括表的名称、表中字段（列）数、主键、定义在该表上的索引。
2. 表中每个字段的定义信息，包括字段类型、是否唯一等。
3. 数据库中所有索引的定义，包括所属表、索引建立在那个字段上等。

Catalog Manager 还必需提供访问及操作上述信息的接口，供 Interpreter 和 API 模块使用。

✧ Record Manager

Record Manager 负责管理记录表中数据的数据文件。主要功能为实现数据文件的创建与删除（由表的定义与删除引起）、记录的插入、删除与查找操作，并对外提供相应的接口。其中记录的查找操作要求能够支持不带条件的查找和带

一个条件的查找（包括等值查找、不等值查找和区间查找）。

数据文件由一个或多个数据块组成，块大小应与缓冲区块大小相同。一个块中包含一条至多条记录，为简单起见，只要求支持定长记录的存储，且不要求支持记录的跨块存储。

✧ Index Manager

Index Manager 负责 B+树索引的实现，实现 B+树的创建和删除（由索引的定义与删除引起）、等值查找、插入键值、删除键值等操作，并对外提供相应的接口。B+树中节点大小应与缓冲区的块大小相同，B+树的叉数由节点大小与索引键大小计算得到。

✧ Buffer Manager

Buffer Manager 负责缓冲区的管理，主要功能有：

1. 根据需要，读取指定的数据到系统缓冲区或将缓冲区中的数据写出到文件
2. 实现缓冲区的替换算法，当缓冲区满时选择合适的页进行替换
3. 记录缓冲区中各页的状态，如是否被修改过等
4. 提供缓冲区页的 pin 功能，及锁定缓冲区的页，不允许替换出去

为提高磁盘 I/O 操作的效率，缓冲区与文件系统交互的单位是块，块的大小应为文件系统与磁盘交互单位的整数倍，一般可定为 4KB 或 8KB。

✧ DB Files

DB Files 指构成数据库的所有数据文件，主要由记录数据文件、索引数据文件和 Catalog 数据文件组成。

3.2 模块设计

3.2.1 Interpreter 模块

✧ **模块概述**

Interpreter 模块的主要功能是实现 miniSQL 与用户之间的交互，将用户在控制台输入的指令进行解析，从而实现对 miniSQL 中各个模块的控制，完成各种操作。

根据 miniSQL 的具体要求，Interpreter 模块能够实现如下功能：创建表、删除表、创建索引、删除索引、选择、插入记录、删除记录、执行脚本文件语句、退出。在能够识别正确的操作语句并将最终操作成功的提示返回的同时，Interpreter 还要做到能够识别相应的语法错误，若存在语法错误的情况，将错误信息返回给用户。

✧ **主要功能**

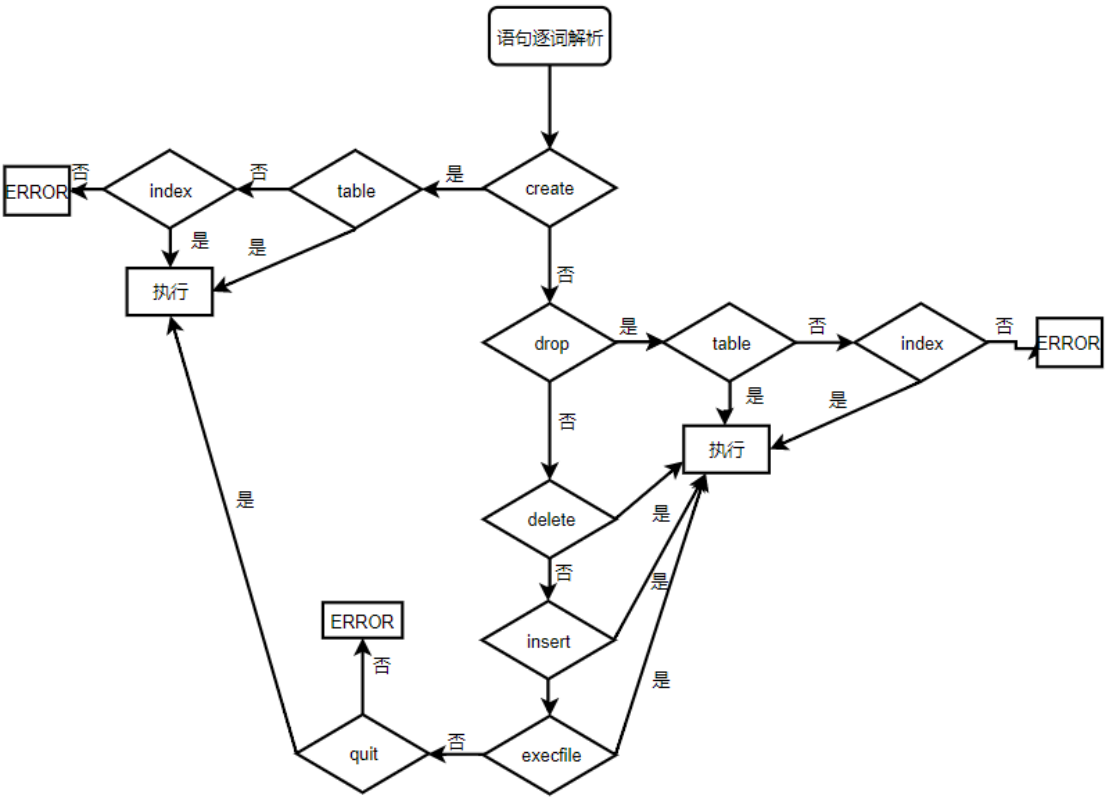
Interpreter 的功能主要是句法解析、生成包含相应指令的 Request 对象以及将最终结果传回主函数进行打印。

➢ **指令解析**

调用 getWord 函数将 Interpreter 得到的指令逐词解析，得到每一个单词。取词函数：

```
1    static String getWord(String sql);
```

Interpreter 具体流程如下图所示：



在解析的过程当中，若出现了不合法的字符或是缺少关键字符，都会直接返回错误提醒。并继续进行下一条指令的执行。

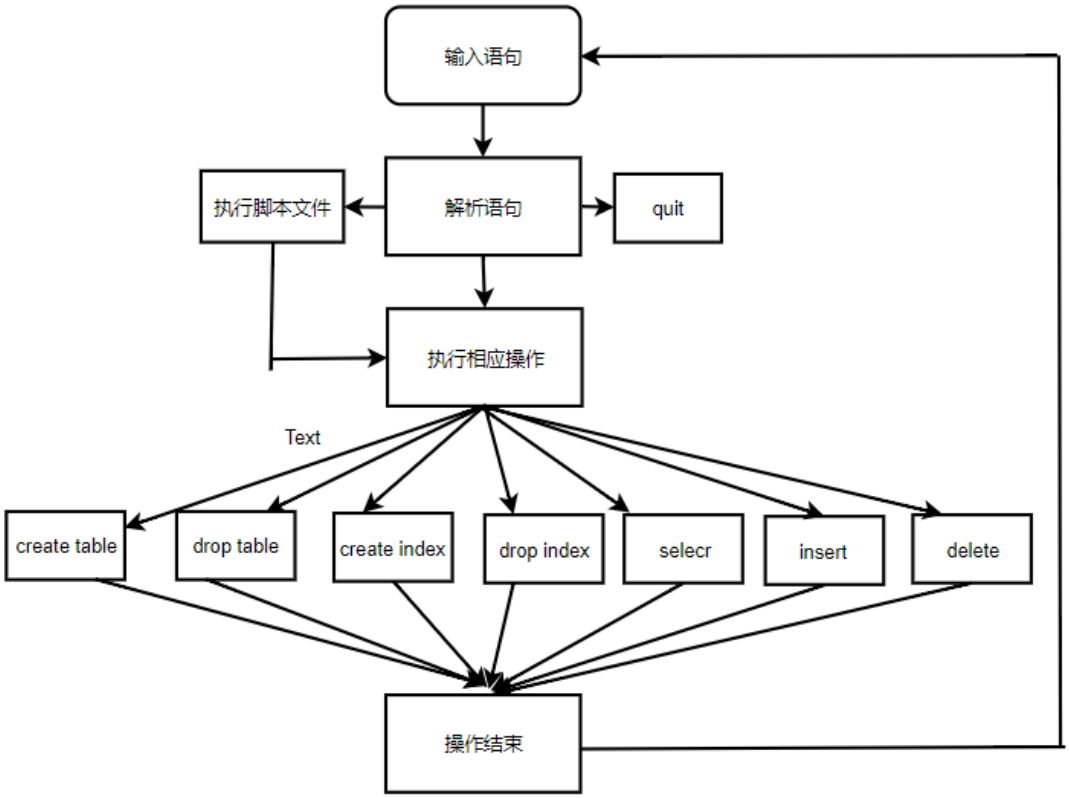
- **生成 request 对象**

编写 Request 类，在语句解析完成且判定该语句合法后，会生成一个 Request 对象。该对象包含执行指令所需的信息，具体分支如下：

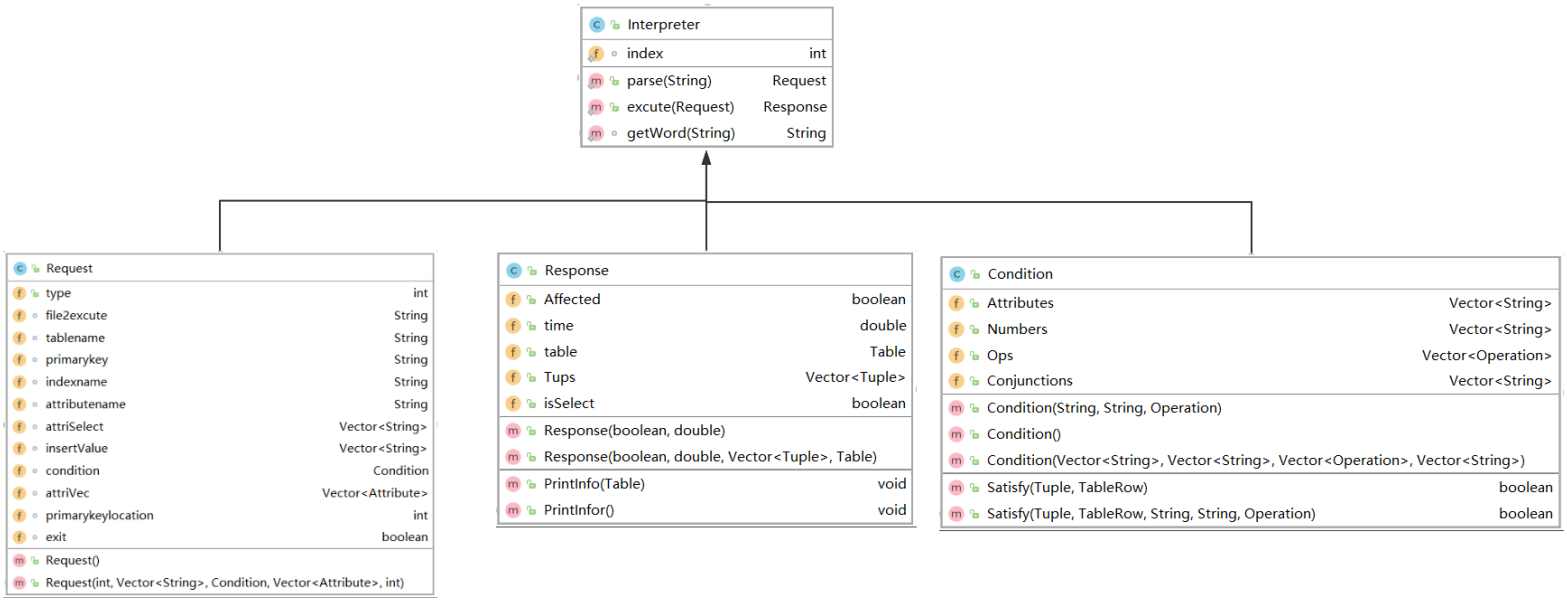
```
- create table: 表名、列 (Attributes) 名称，列属性。
- drop table: 表名。
- create index: 表名称，列名称，索引名称。
- drop index: 索引名称。
- insert ：表名称，插入值。
- delete: 表名称，删除条件。
- select: 表名称，选择条件。
- execfile: 待执行文件名。
```

最终生成的 Request 对象供 API 模块调用进行具体操作。

✧ 工作流程图


输入语句
解析语句
执行脚本文件
quit
执行相应操作
Text
create table
drop table
create index
drop index
selecr
insert
delete
操作结束

✧ 类设计图


Interpreter
index int
parse(String) Request
excute(Request) Response
getWord(String) String
Request
type int
file2excute String
tablename String
primarykey String
indexname String
attributename String
attriSelect Vector<String>
insertValue Vector<String>
condition Condition
attriVec Vector<Attribute>
primarykeylocation int
exit boolean
Request()
Request(int, Vector<String>, Condition, Vector<Attribute>, int)
Response
Affected boolean
time double
table Table
Tups Vector<Tuple>
isSelect boolean
Response(boolean, double)
Response(boolean, double, Vector<Tuple>, Table)
PrintInfo(Table) void
PrintInfor() void
Condition
Attributes Vector<String>
Numbers Vector<String>
Ops Vector<Operation>
Conjunctions Vector<String>
Condition(String, String, Operation)
Condition()
Condition(Vector<String>, Vector<String>, Vector<Operation>, Vector<String>)
Satisfy(Tuple, TableRow) boolean
Satisfy(Tuple, TableRow, String, String, Operation) boolean

3.2.2 API 模块

API 模块是整个 miniSQL 系统的核心。

✧ **主要功能**

其主要功能提供执行 SQL 语句的接口，供 Interpreter 调用。API 接受来自 Interpreter 的 Request 对象，调用 BufferManager、IndexManager、RecordManager 等模块，实现所需执行的各项功能。在语句执行完毕之后，将最终的结果返回给 Interpreter 模块进行输出。

✧ **接口设计**

➢ 创建表

```
1 static public Response createTable(Request request);
```

➢ 删除表

```
1 static public Response dropTable(Request request);
```

➢ 创建索引

```
1 static public Response createIndex(Request request);
```

➢ 删除索引

```
1 static public Response dropIndex(Request request);
```

➢ 记录查找

```
1 static public Response select(Request request);
```

➢ 记录插入

```
1 static public Response insert(Request request);
```

➢ 记录删除

```
static public Response delete(Request request;
```

调用 API 提供的各个接口之后，无论执行成功与否，都将返回给 Interpreter 一个 Response 对象。该对象内包含了指令执行的结果，若存在记录的查找，Response 对象中还将包含选择出来的记录信息，供 Interpreter 打印。

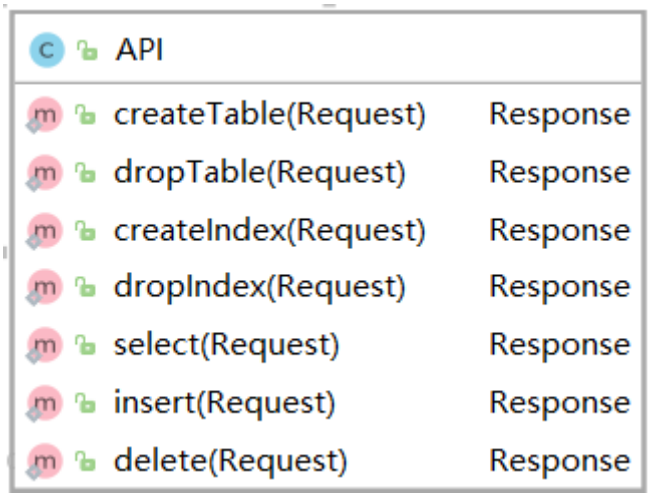
✧ **返回 response 对象**

➢ 基础变量定义

```
    public boolean Affected; /*执行是否成功*/

    public double time; /*执行耗时*/

    public Table table = new Table();  /*查询模式下返回的表*/

    public Vector<Tuple>Tups = new Vector<Tuple>();
    /*选择记录操作时挑选出的信息*/
```

在 API 中生成并返回到 Interpreter 中，最终在 main 函数中调用函数 Response.PrintInfo()，打印相关信息。

✧ **类设计图**



3.2.3 Catalog Manager 模块

✧ **模块概述**

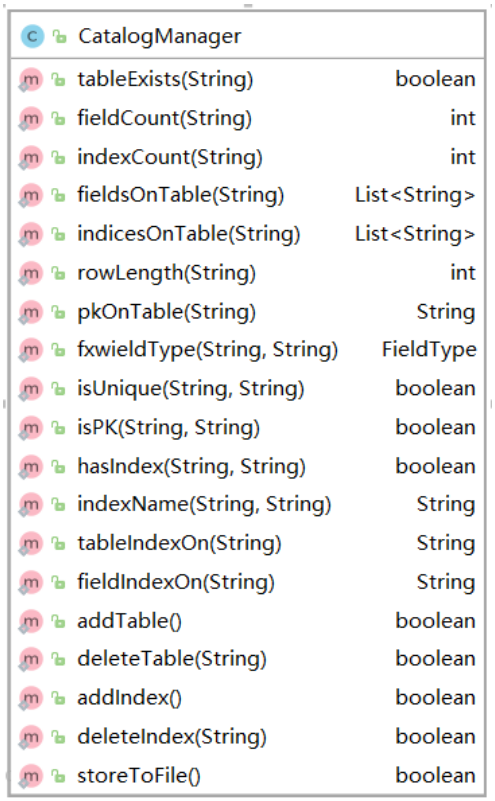
Catalog Manager 负责管理数据库的所有模式信息，并提供访问及操作这些

信息的接口，供 Interpreter 和 API 模块使用。

✧ **接口设计**

```java
//判断表格是否存在
public static boolean tableExists(String table);
//返回表中所有记录的个数
public static int fieldCount(String table);
//返回表的属性列表
@SuppressWarnings("null")
public static List<String> fieldsOnTable(String table);
//判断属性是否为unique
public static boolean isUnique(String table, String field);
//判断属性是否具有Index
public static boolean hasIndex(String table, String field);
```

✧ **类设计图**



3.2.4 Record Manager 模块

✧ **模块概述**

本模块负责管理数据的组织方式，同时 API 会调用 BufferManager 中对应的函数进行增删改查以及索引的建立和删除。同时在这个模块中需要对 Buffer 中

block 的数据进行进一步解析，来使数据能够以合理的方式被读取传递。在这个模块中也需要对条件进行判定。

✧ 接口设计

➢ FieldType

此模块为 enum 类型，定义了 Attribute 的类型。

```
enum FieldType {INT, FLOAT, STRING, Empty;}
```

➢ Attribute

此模块定义了表的属性。

```
FieldType Type;
// 对应Attribute的类型
String attriName;
// Attribute对应的名称
int lengtg;
// 字符串长度
int offset;
// 偏移量
boolean isPrimary
// 属性是否为主键
boolean isUnique;
// 属性是否为Unique
boolean hasInex
// 属性上手否有索引
```

```
Attribute(FieldType Type, String attriName);
// 构造函数，用于INT、FLOAT类型属性的构造
Attribute(FieldType Type, String attriName, int length, int offset);
// 构造函数，用于STRING类型属性的构造
void SetPrimary();
// 设置属性为主键
void SetUnique();
// 设置属性为Unique
void SetIndex();
// 设置属性有索引
```

➢ TableRow

此模块定义了表属性的集合。

```
List<Attribute> attlist;
// 表属性的集合
int attrinum;
// 属性个数
```

```
public TableRow( List<Attribute> attlist, int attrinum)
// 构造函数
FieldType GetType(String AttName);
// 获得对应属性名的类型
int GetIndex(String AttName);
// 获得对应属性名在List中的索引
Attribute GetAtt(String AttName);
// 获得属性名所对应的属性
int size();
// 获得table一行的大小 (byte)
```

- Table

此模块定义了表的形式。

```
String TableName;
// 表的名称
TableRow Row;
// 表的属性集合
int RecordNum;
// 表的记录数
```

```
Table(String TableName, TableRow Row);
// 构造函数
Table(String TableName, TableRow Row, int RecNum);
// 构造函数
Attribute GetAttribute(String AttriName);
// 获得对应属性名的属性
```

- Tuple

此模块定义了表中元组的组织方式，Tuple 将作为表中带有数据的元组的传递单位。

```
int valid;
// 是否可用
Vector<String>Data;
// 数据存储
// 以String来存储数据，不同属性的数据有不同的解析方式
```

```
Tuple();
// 构造函数
Tuple(int valid, Vector<String> data);
// 构造函数
int size();
// 获取元组的大小
String GetData(int index);
// 获取Vector中对应下标的数据
byte[] GetBytes(Table table);
// 将Tuple中的数据按照table中的解析方式来解析成byte[]
// 使Tuple中的信息可以以byte写到磁盘上去

```

- Condition

此模块记录了所要判断的条件，其中有函数能够判断传入的 Tuple 是否满足条件，用于条件的判断，支持介词判断。

```
enum Operation{EQUAL, NOT_EQUAL, LESS, MORE, LESS_EQUAL,
               MORE_EQUAL, EMPTY;}
// enum类型，枚举比较的符号
Vector<String> Attributes;
// 传入的属性名
Vector<Operation> Ops;
// 传入的比较符号
Vector<String> Numbers;
// 传入的数值
Vector<String>Conjunctions;
// 传入的介词
```

```
boolean createTable(String Name, List<Attribute> Atts);
// 传入表的名称、表的属性
// 传出表是否建立成功
boolean dropTable(Table table);
// 删除表
// 传入要删除的表，将其在内存和磁盘上均删除
boolean dropTable(String TableName);
// 重载函数，删除表
// 根据表的名称，在内存和磁盘上删除对应的表
void insert(Table table, Tuple tup);
// 插入数据
// 往对应的table中插入Tuple所传入的数据
// 需要根据table中的信息来对Tuple进行一定的解析
void insert(Table table, List<Tuple> tups);
// 重载函数，插入多个数据
// 多次调用函数insert(Table table, Tuple tup);
Vector<Tuple> select(Table table, Condition condition);
// 查询函数
// 根据对应的条件Condition，从table所对应的数据中进行查找
// RecordManager中的查找为遍历查找
Vector<Tuple> SelectAll(Table table);
// 查询函数
// 将table中的所有记录传出
// 用于语句select * from <table>;
Tuple select(Table table, int Offset);
// 挑选出对应Offset的Tuple
// 主要用于上层模块的调用(使用Index来进行查找)
Vector<Tuple>select(Table table, Vector<Integer>Offsets);
// 重载函数
// 挑选出Offsets所对应的所有Tuple
// 用于上层模块的调用 (使用Index进行查找)
int delete(Table table, Condition condition);
// 删除满足对应条件的元组
// 传出删除的元组数
int deleteAll(Table table)
```

➢ RecordManager

此模块负责记录查询、插入、删除以及表的建立与删除，提供相应的函数来给 API 模块调用。

✧ 类设计图

3.2.5 Index Manager 模块

✧ **模块概述**

Index Manager 负责 B+树索引的实现，实现 B+树的创建和删除（由索引的定义与删除引起）、等值查找、插入键值、删除键值等操作，并对外提供相应的接口。B+树中节点大小应与缓冲区的块大小相同，B+树的叉数由节点大小与索引键大小计算得到。

✧ **接口设计**

➢ TreeNode

TreeNode 定义了一个节点的泛型类，每个节点存储该节点的键值对，键可

以是 int,float,string 三种类型，故采用泛型类实现。节点分为两种类型：内部节点和叶子节点。内部节点存储的是键和指向孩子节点的引用。模块中还提供了节点内的诸多操作接口（如节点内的搜索、插入、删除），是 B+树实现的基础。

```
public final T initKey = null;
public static int degree;                    //Degree of node
public int numOfKeys;                        //Number of keys
public NodeType nodeType;                    //Type of node
public TreeNode <T> parent;                  //Parent node
public TreeNode <T> nextLeafNode;            //Next leaf node
public Vector <T> keys;                      //Search keys
public Vector < TreeNode<T> > children; //Children nodes
public Vector <Integer> values; //in-block address
```

```
//constructor
public TreeNode(NodeType nodeType);
//Search key in the tree node, and remian
//the index in "result"
public boolean searchKey(T key, Result<T> result);
//Split the node into 2 sub-nodes,Return the new node
public TreeNode<T> splitNode(Result<T> result);
//Add key-child pair to internal node
public int addKey(T key);
//Add key-reference pair to internal node
public int addKeyValue(T key, int value);
//Remove a key from the node
public boolean removeKey(int index);
```

- BPlusTree

BPlusTree 是基于 TreeNode 实现的 泛型类，可以建立 int,float,string 类型的 B+树。此模块提供了 B+树的所有操作，如树的建立、搜索、插入、删除操作，以及与 buffer 的读写交互。

```
public int degree;                  //Degree of tree
public int keySize;                 //Size of key
public int height;                  //Height of tree
public int numOfNodes;              //Number of nodes
public FieldType keyType;           //Type of key
public String fileName;             //Path of index file
public TreeNode<T> root;            //Root node
public TreeNode<T> headLeafNode;//First leaf node
public TreeNode<T> lastLeafNode;//Last leaf node
```

```
//Constructor
public BPlusTree(String fileName, int keySize,
                 int degree, FieldType keyType);
//Search the tree to find key until leaf
public boolean findAtLeaf(TreeNode<T> node, T key,
                          Result<T> result);
//Search key throuth the tree
public int search(T key);
//Insert key-value pair into tree
public boolean insert(T key, int value);
//Adjust the tree after insertion
public void insertAdjust(TreeNode<T> node);
//Delete key-value based on key
public void delete(T key);
//Adjust the tree after deletion
public void deleteAdjust(TreeNode<T> node);
//Find the most left key of node
public T mostLeftKey(TreeNode<T> node)
//Display the tree for debug
public void printTree();
```

- IndexManager

IndexManager 基于 BPlusTree 实现对索引的管理以及运用，包括新建索引、删除索引、插入键值、基于索引搜索等，通过调用 BPlusTree 来维护 index file，以供 API 调用。

```
public static BPlusTree<Integer> iTree;//An int tree
public static BPlusTree<Float> fTree;//A float tree
public static BPlusTree<String> sTree;//A string tree
public static String indexFileName;//File path for index
```

```
//Initialize a tree
public static void InitTree(FieldType type, int keySize)
//Create index on table
public static boolean createIndex(String tableName, String
attributeName, String indexName) {
//Drop a index with index name
public static boolean dropIndex(String indexName) {
//Single-equivalent select
public static int select(Table table, Attribute attribute,
String key) {
//Insert key into index
public static boolean insert(String tableName, String
attributeName, String key, int addr) {
```

```
14 //Delete key from index
15 public static boolean delete(String tableName, String
16 attributeName, String key) {
17 //Delete all the keys, clear the index file
18 public static boolean deleteAll(String tableName, String
19 attributeName) {
20 //Read index from buffer
21 public static void readFromBuffer(FieldType type)
22 //Write information into buffer
23 public static void writeToBuffer(FieldType type)
```

✧ 类设计图

3.2.6 Buffer Manager 模块

✧ **模块概述**

本模块负责缓冲区的管理，在设计时选择 1K 的块大小，Buffer 中存在 1K 个 block。在实现 miniSQL 时，需要读取磁盘中的数据并实现内存与磁盘的交流，故所有与磁盘读取和写出的功能均在此实现。

✧ **接口设计**

➢ Block

用于记录块的信息，包括是否为脏块、是否被锁，是否可用，数据以 byte 数组来进行存储，同时可直接调用函数来进行块的写入写出。在 Block 中也有对于对于其中信息的解析方式(INT, FLOAT, STRING)。

```
int Size;  //页的大小
String file; //文件名
int fileOffset; //块在文件中的偏移量
boolean isDirty; //块是否为脏
boolean isPined; //块是否被锁
boolean isValid; //块是否可用
byte[] data; //数据
```

```
public void SetBlock(String file, int fileOffset);
//设置块的文件名和偏移量
public void LoadBlock();
//从磁盘中将对应块的数据读入
public void WriteData(byte []data2write,int length,int off);
//在块中写入对应的数据
public void WriteBack();
//将块中信息写会到对应文件
public int GetInt(int offset);
//从块中数据偏移量为offset起的四个byte解析为int传出
public float GetFloat(int offset);
//从块中数据偏移量为offset起的四个byte解析为float传出
public String GetString(int offset, int length);
//从块中数据偏移量为offset起的length个byte解析为String传出
public void WriteInt(int num, int offset);
//将传入num写入在块中数据偏移量为offset起的四个byte
public void WriteFloat(float num, int offset);
//将传入num写入在块中数据偏移量为offset起的四个byte
public void WriteString(String S, int offset);
//将传入S写入在块中数据偏移量为offset起的length(S)个byte
```

- BufferManager

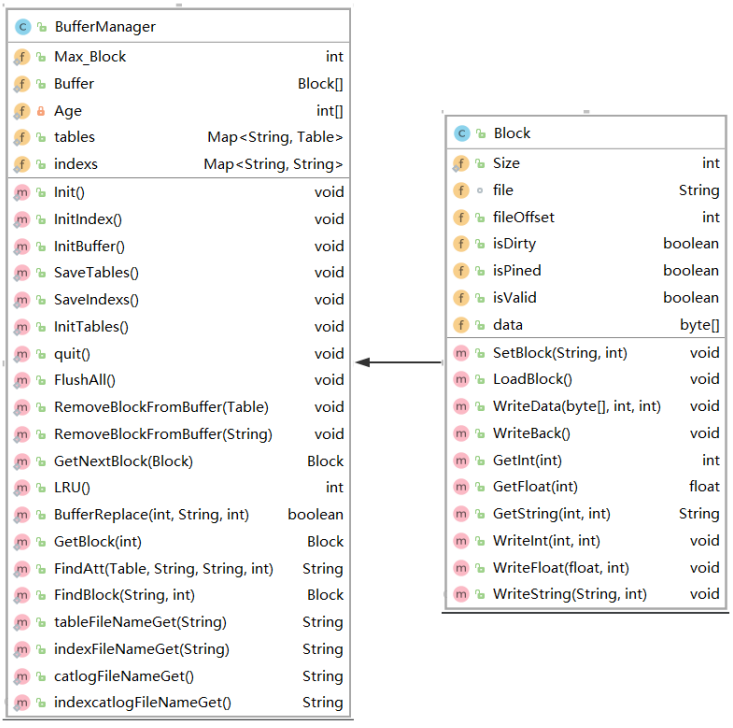
BufferManager 主要实现了 buffer 的替换算法、信息的初始化、信息往磁盘上存储、并实现了部分解析的功能来方便上层模块的调用。

```
int Max_Block;
//Buffer的大小
Block []Buffer;
int []Age;
// 记录块最近一次调用时间，用于实现LRU算法
Map<String, Table> tables;
// <String TableName, Table table> 记录所建立的所有Table
Map<String, String> indexs;
// first is the IndexName, second is the "table_attribute"
// 用于记录索引的名字以及索引所对应的table以及attribute需要再次解析
```

```
void Init();
// 初始化内存中的内容
// 在每次miniSQL刚启动之时调用
void InitIndex()
// 初始化Index内容
// 在Init()函数中调用
// 从文件"Index_Cat_File.SQLCAT"中进行解析
void InitBuffer()
// 初始化Map<String, String> indexs内容
// 在Init()函数中调用
// 将Age初始化为0，Buffer中所有Block设置isValid为true
void InitTables();
// 初始化Map<String, Table> tables的信息
// 在Init()函数中调用
// 从文件"CATLOG_FILE.SQLCAT"中进行解析
void quit()
// 将内存中的内容写入磁盘之中
// 在每次miniSQL关闭之时调用
void FlushAll();
// 将所有脏块强制写出
// 在quit()中调用
void SaveTables();
// 将table信息存储在磁盘上
// 最终存储文件名为"CATLOG_FILE.SQLCAT"
// 在quit()中调用
void SaveIndexs();
// 将Index信息存储在磁盘上
// 最终存储文件名为"Index_Cat_File.SQLCAT"
// 在quit()中调用
void RemoveBlockFromBuffer(Table table)
```

```
// 将对应table的块在Buffer中的块移除
// 在删除表时进行调用
static public void RemoveBlockFromBuffer(String FileName)
// 将对应文件名的块在Buffer中的块移除
// 在删除表时进行调用
Block FindBlock(String FileName, int offset)
// 获得对应参数的块
// 若块在Buffer中则之间将这个块传出
// 若块不再Buffer中，使用LRU算法进行块的替换
static public Block GetNextBlock(Block b)
// 获得这个块所对应文件的下一个块
// 此函数主要用于遍历时从文件对应的第一个块遍历到最后一个快
// 实际调用函数为：FindBlock(b.file, b.fileOffset + Max_Block);
Block GetBlock(int index);
// 获得Buffer[index]
boolean BufferReplace(int index, String file, int offset);
// 进行Buffer的替换
// 将Buffer[index]所对应的块替换成文件名为file，偏移量为offset的块
// 在进行块的替换时调用
public int LRU()
// LRU算法
// 根据Age[]来找寻要替换的块
// 传出要替换的块在buffer中的索引
String FindAtt(Table table,String field,String file,int offse
// 在Block中解析出对饮filed的值
// table中存储着解析byte[]数据的信息
// 该函数主要在上层模块中调用
```

✧ 类设计图

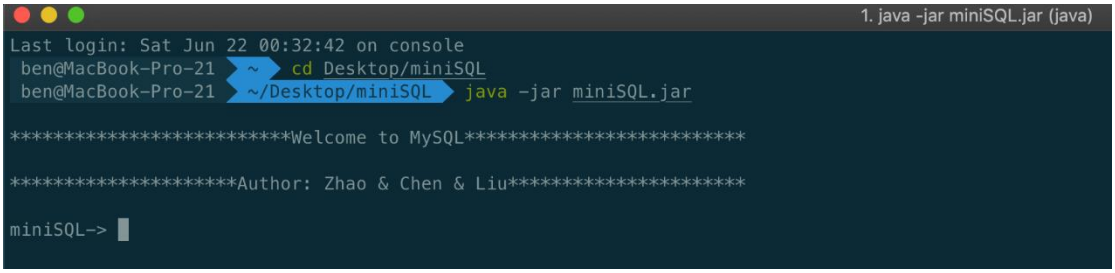


4 测试结果

4.1 测试方法

在终端切入对应的文件夹　输入如下指令：java -jar miniSQL.jar

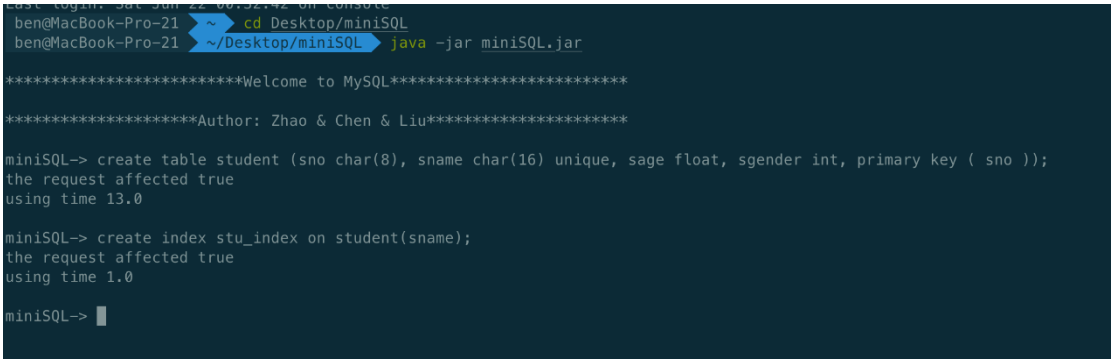
运行界面如下：



4.2 功能测试

- 创建新的表

- 创建索引



- 插入数据

```
miniSQL-> insert into student values('12345629', '1234567890123451', 12.231, 1);
the request affected true
using time 7.0

miniSQL-> insert into student values('12345672', '1234567890123452', 13.241, 1);
the request affected true
using time 1.0

miniSQL-> insert into student values('12345673', '1234567890123453', 14.251, 1);
the request affected true
using time 0.0

miniSQL-> insert into student values('12345675', '1234567890123456', 15, 1);
the request affected true
using time 2.0

miniSQL-> insert into student values('10345610', '1234567890123457', 15, 1);
the request affected true
using time 1.0
```

- 查询数据

```
miniSQL-> select * from student;
+---------+----------------+----------+----------+
|sno      |sname           |sage      |sgender   |
+---------+----------------+----------+----------+
|12345629|1234567890123451|12.231    |1         |
+---------+----------------+----------+----------+
|12345672|1234567890123452|13.241    |1         |
+---------+----------------+----------+----------+
|12345673|1234567890123453|14.251    |1         |
+---------+----------------+----------+----------+
|12345675|1234567890123456|15.0      |1         |
+---------+----------------+----------+----------+
|10345610|1234567890123457|15.0      |1         |
+---------+----------------+----------+----------+
the request affected true
using time 1.0
```

- 执行脚本（插入 5000 条数据）

```
insert into student values('57940000', '5794000000000000', 12, 1)
insert into student values('67940000', '6794000000000000', 12, 1)
insert into student values('77940000', '7794000000000000', 12, 1)
insert into student values('87940000', '8794000000000000', 12, 1)
insert into student values('97940000', '9794000000000000', 12, 1)
insert into student values('08940000', '0894000000000000', 12, 1)
insert into student values('18940000', '1894000000000000', 12, 1)
insert into student values('28940000', '2894000000000000', 12, 1)
insert into student values('38940000', '3894000000000000', 12, 1)
insert into student values('48940000', '4894000000000000', 12, 1)
insert into student values('58940000', '5894000000000000', 12, 1)
insert into student values('68940000', '6894000000000000', 12, 1)
insert into student values('78940000', '7894000000000000', 12, 1)
insert into student values('88940000', '8894000000000000', 12, 1)
insert into student values('98940000', '9894000000000000', 12, 1)
insert into student values('09940000', '0994000000000000', 12, 1)
insert into student values('19940000', '1994000000000000', 12, 1)
insert into student values('29940000', '2994000000000000', 12, 1)
insert into student values('39940000', '3994000000000000', 12, 1)
insert into student values('49940000', '4994000000000000', 12, 1)
insert into student values('59940000', '5994000000000000', 12, 1)
insert into student values('69940000', '6994000000000000', 12, 1)
insert into student values('79940000', '7994000000000000', 12, 1)
insert into student values('89940000', '8994000000000000', 12, 1)
insert into student values('99940000', '9994000000000000', 12, 1)
the request affected true
using time 31108.0
```

- 根据索引查询

```
miniSQL-> select * from student where sname = '9000000000000000';
+--------+---------------+----------+----------+
|sno     |sname          |sage      |sgender   |
+--------+---------------+----------+----------+
|90000000|9000000000000000|12.0      |1         |
+--------+---------------+----------+----------+
the request affected true
using time 5.0
```

- 主键等值查询

```
miniSQL-> select * from student where sno = '90000000';
+--------+---------------+----------+----------+
|sno     |sname          |sage      |sgender   |
+--------+---------------+----------+----------+
|90000000|9000000000000000|12.0      |1         |
+--------+---------------+----------+----------+
the request affected true
using time 3.0
```

- 主键非等值查询

```
miniSQL-> select * from student where sage = 15;
+--------+----------------+----------+----------+
|sno     |sname           |sage      |sgender   |
+--------+----------------+----------+----------+
|12345675|1234567890123456|15.0      |1         |
+--------+----------------+----------+----------+
|10345610|1234567890123457|15.0      |1         |
+--------+----------------+----------+----------+
the request affected true
using time 18.0
```

- 非键值属性等值查询

```
miniSQL-> select * from student where sno >= '99900000';
+--------+----------------+----------+----------+
|sno     |sname           |sage      |sgender   |
+--------+----------------+----------+----------+
|99900000|9990000000000000|12.0      |1         |
+--------+----------------+----------+----------+
|99910000|9991000000000000|12.0      |1         |
+--------+----------------+----------+----------+
|99920000|9992000000000000|12.0      |1         |
+--------+----------------+----------+----------+
|99930000|9993000000000000|12.0      |1         |
+--------+----------------+----------+----------+
|99940000|9994000000000000|12.0      |1         |
+--------+----------------+----------+----------+
the request affected true
using time 18.0
```

- 单个条件查询

```
miniSQL-> select * from student where sno = '90000000' or sage = 15 or sno = '01000000';
+--------+----------------+----------+----------+
|sno     |sname           |sage      |sgender   |
+--------+----------------+----------+----------+
|12345675|1234567890123456|15.0      |1         |
+--------+----------------+----------+----------+
|10345610|1234567890123457|15.0      |1         |
+--------+----------------+----------+----------+
|90000000|9000000000000000|12.0      |1         |
+--------+----------------+----------+----------+
|01000000|0100000000000000|12.0      |1         |
+--------+----------------+----------+----------+
the request affected true
using time 6.0
```

- 多个条件查询

```
miniSQL-> select * from student where sno = '90000000' or sage > 12.23;
+--------+---------------+----------+----------+
|sno     |sname          |sage      |sgender   |
+--------+---------------+----------+----------+
|12345629|1234567890123451|12.231    |1         |
+--------+---------------+----------+----------+
|12345672|1234567890123452|13.241    |1         |
+--------+---------------+----------+----------+
|12345673|1234567890123453|14.251    |1         |
+--------+---------------+----------+----------+
|12345675|1234567890123456|15.0      |1         |
+--------+---------------+----------+----------+
|10345610|1234567890123457|15.0      |1         |
+--------+---------------+----------+----------+
|90000000|9000000000000000|12.0      |1         |
+--------+---------------+----------+----------+
the request affected true
using time 8.0
```

- 条件删除

```
miniSQL-> select * from student where sno = '09920000' or sage = 15;
+--------+---------------+----------+----------+
|sno     |sname          |sage      |sgender   |
+--------+---------------+----------+----------+
|12345675|1234567890123456|15.0      |1         |
+--------+---------------+----------+----------+
|10345610|1234567890123457|15.0      |1         |
+--------+---------------+----------+----------+
|09920000|0992000000000000|12.0      |1         |
+--------+---------------+----------+----------+
the request affected true
using time 6.0

miniSQL-> delete from student where sno = '09920000' or sage = 15;
the request affected true
using time 34.0

miniSQL-> select * from student where sno = '09920000' or sage = 15;
+--------+---------------+----------+----------+
|sno     |sname          |sage      |sgender   |
+--------+---------------+----------+----------+
the request affected true
using time 9.0
```

- 删除所有记录

```
miniSQL-> delete from student;
the request affected true
using time 2.0

miniSQL->
miniSQL-> select * from student;
+--------+----------------+----------+----------+
|sno     |sname           |sage      |sgender   |
+--------+----------------+----------+----------+
the request affected true
using time 0.0
```

- 删除索引

```
miniSQL-> drop index stu_index;
the request affected true
using time 0.0
```

- 删除表

```
miniSQL-> drop table student;
the request affected true
using time 0.0
```

4.3 压力测试

执行脚本一次性插入 5000 条记录来测试系统性能，根据 5.2 节执行脚本的结果来看，系统一次性能处理至少 5000 条数据，抗压能力较强。

5 总结展望

通过一段时间的共同努力，我们基本实现了 miniSQL 的主要功能，并对这些功能进行了详尽的测试，测试结果反映良好。

在编写 miniSQL 的过程中，我们更加深入和细致的认识了数据库的工作原理，对 API、Interpreter、BufferManager、RecordManager、IndexManager 等

模块的有了深刻的理解。在实现这些模块的基本功能时，学习了数据库系统的基本概念；而在实现各个模块之间互相调用的过程中，体验了一个完整的数据库系统的来之不易，锻炼了我们 debug 与合作互助的能力。

目前我们完成的 miniSQL 仍然只是一个功能相对单一的系统，一部分现有的功能不完整或是有差错，例如在验收过程中出现了 drop index 时 Index 对应文件并未被删除的问题。与此同时，还有很多新的功能亟待开发与实现。希望以后我们能够制作一个完成度更高的 miniSQL 系统，学以致用。

6 分工说明

- 陈锰：Index Manager 模块、DBFiles 模块、部分 API 模块，整合测试，报告汇总
- 刘邵轩：Interpreter 模块、Catalog Manager 模块、部分 API 模块，整合测试
- 赵子瑜：Buffer Manager 模块、Record Manager 模块、部分 API 模块，整合测试