数据库系统实验报告

作业名称：MINISQL

姓　　名：陈德瀚、康雅琪、庞懿非

学　　号：3190102203、3190106213、3190104534

电子邮箱：cdh573885@outlook.com

联系电话：18927403099

指导老师：孙建伶

日　　期：2021年6月28日

MINISQL

一、实验目的

设计并实现一个精简型单用户 SQL 引擎(DBMS)MiniSQL，用户可以通过字符界面输入 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/查找及退出。

通过对 MiniSQL 的设计与实现，提高系统编程能力，加深对数据库系统原理的理解。

二、系统需求

Windows10

三、实验环境

1. 开发语言：C++
2. 交互界面：字符界面
3. 开发工具：Microsoft Visual Studio / Visual Studio Code

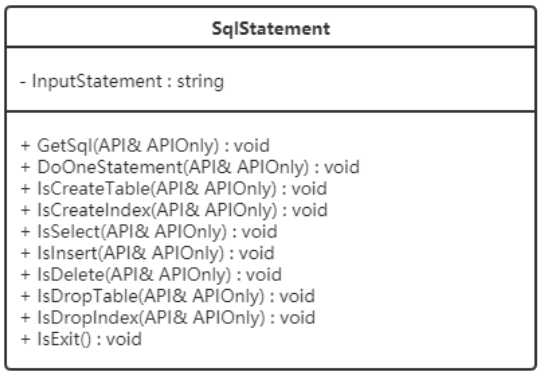
四、系统设计

1. 分工情况

- 康雅琪：Catalog Manager, Interpreter, API
- 陈德瀚：Buffer Manager, Record Manager
- 庞懿非：Index Manager

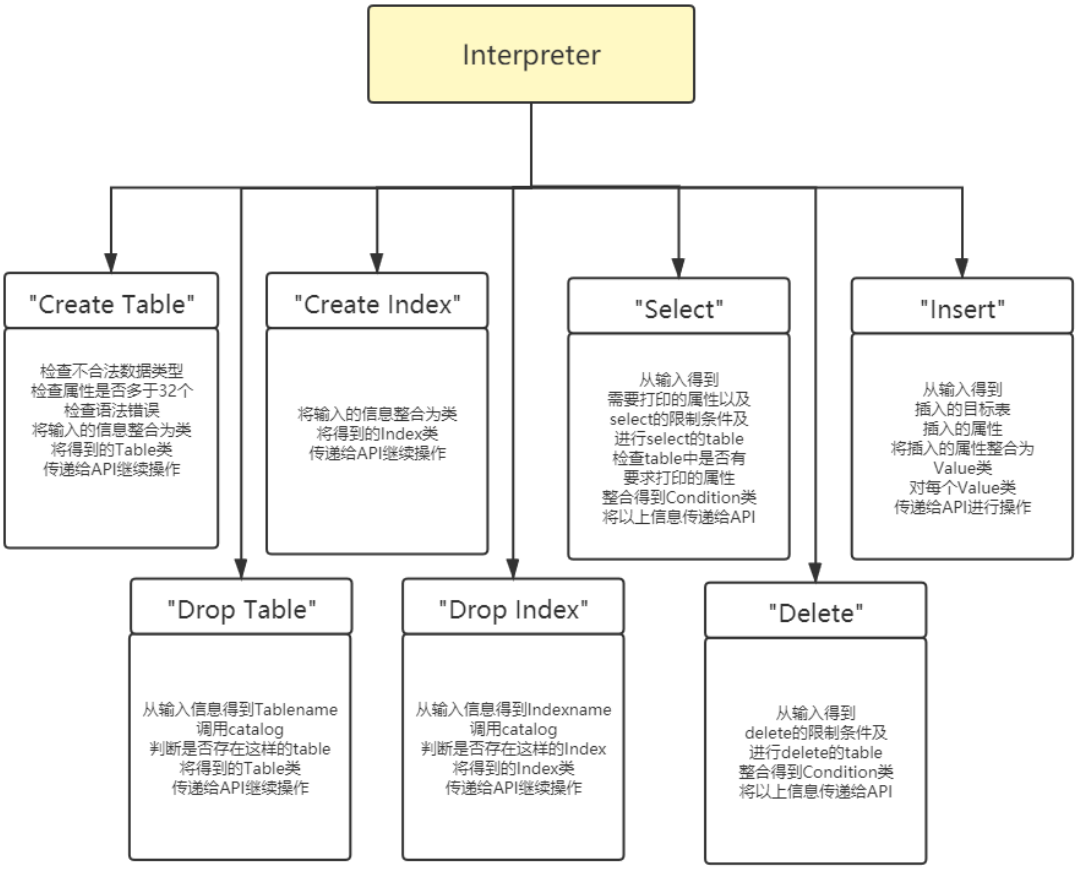
2. Interpreter

- **类图**



- **功能描述**

  Interpreter的主要功能为直接与用户交互、程序流程控制、接收并解释用户输入的命令，生成内部的类的表示、检查语句的语法正确性、显示错误信息等。如图所示：



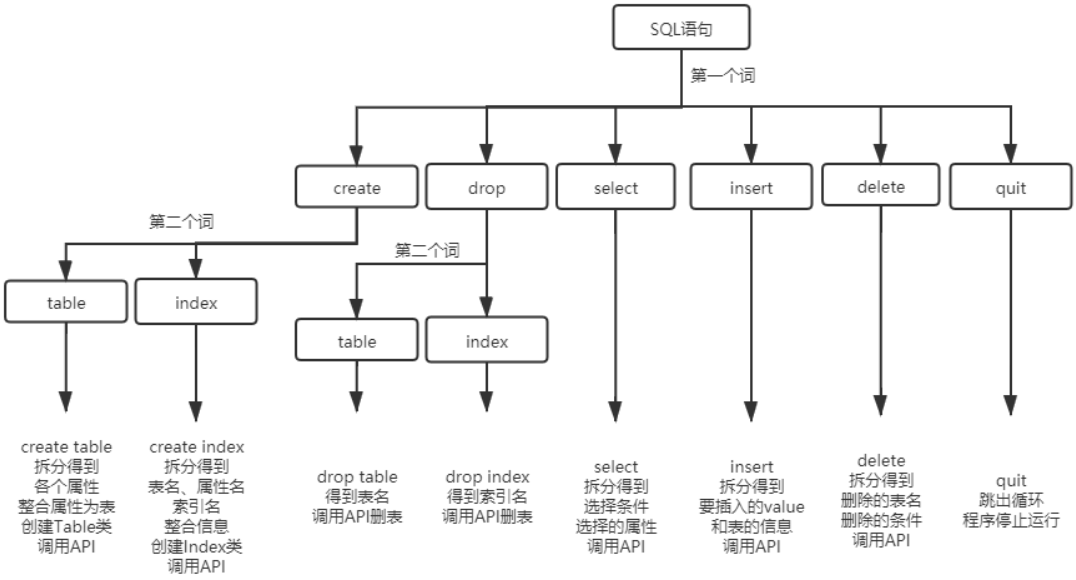
- **语句错误检测**

Interpreter对输入语句采用“特征点切割”进行检测。首先从输入读取若干行进行连接，直到出现“;”分号，停止读取并将该条语句实为一条SQL语句进行处理。

在Interpreter得到一条SQL语句后，首先对这条语句的头尾进行去空格和去换行符的操作，然后读取该条语句的第一个词，并根据这个词进行分支。若为“create”，则根据第二个词进入“create table”或“create index”的判断。若为“drop”，则根据第二个词进入“drop table”或“drop index”的判断。若为“select”、“insert”、“delete”，则各自进入信息提取。若不是以上的全部情况，则判断为语法错误。

在利用以上关键词完成对该条语句功能的判断后，利用各种符号（如“,”、“;”、“(”、“)”等）、表名、属性名、特征点（如“into”、“where”、“and”等）来由输入的SQL语句整合得到Catalog中定义的Attribute、Table、Index、InsertValue类的信息。

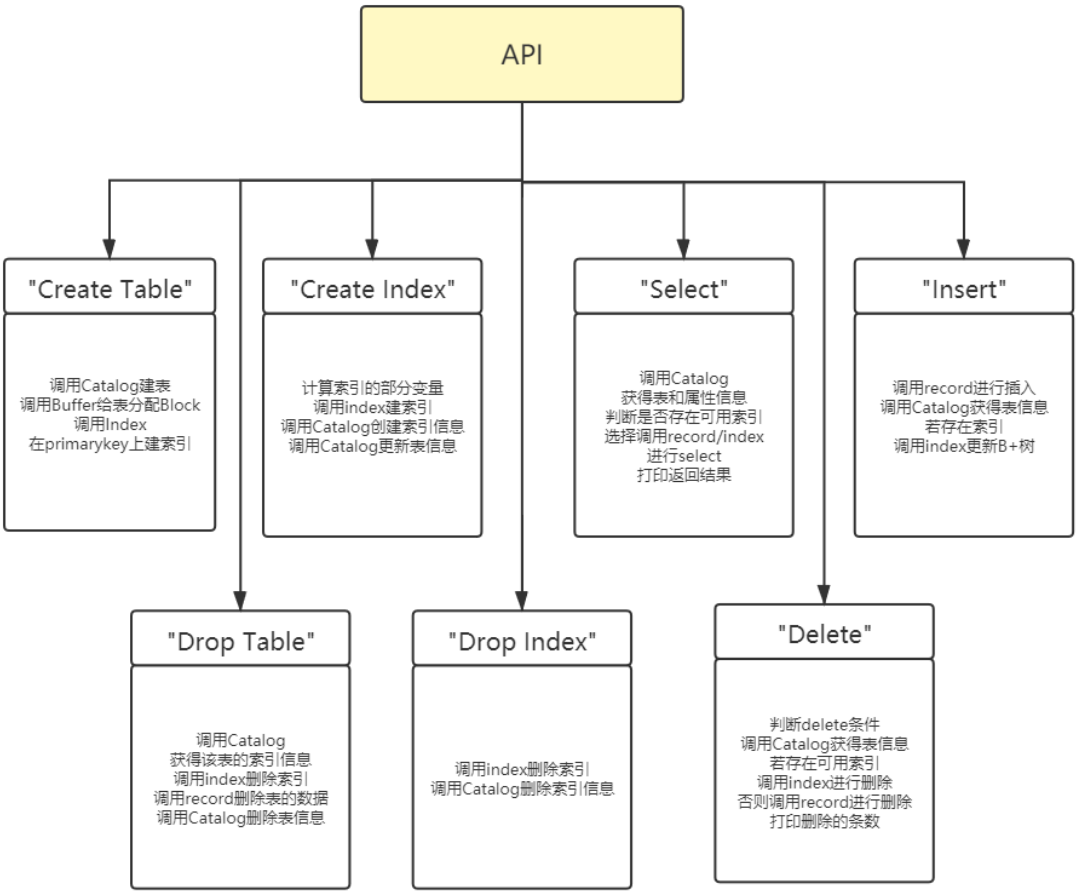
一些简单的错误判断在Interpreter中进行，如create table时的属性名重复、属性个数多于32个、数据类型错误。



3. API

- **功能描述**

  API主要功能为提供执行SQL语句的接口，供Interpreter调用、处理插入的value、向Record Manager和Index Manager提供执行select语句时需要的判断条件及通过调用Catalog Manager、Record Manager、Index Manager，来实现各个语句。

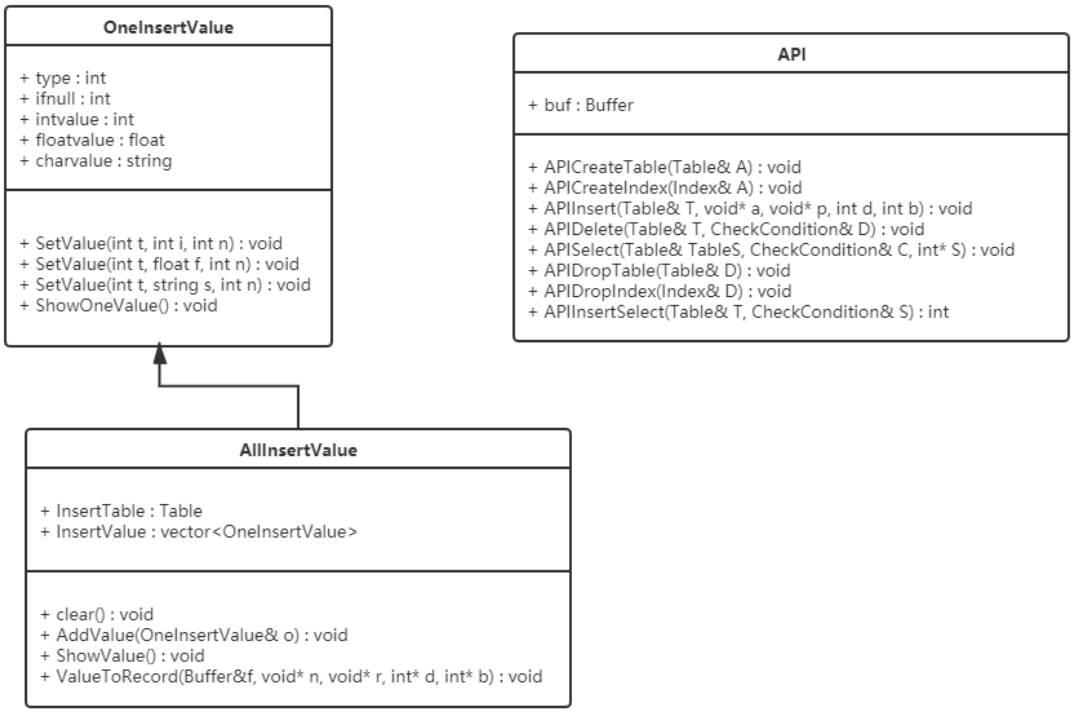


- **类图**

  API中定义OneInsertValue、AllInsertValue、API三个类。

  OneInsertValue用于存储插入时insert语句中其中一个value值，例如“insert into table testtable values(11,22,33);”，整数11将被记作一个OneInsertValue，整数22将被记作一个OneInsertValue，整数33将被记作一个OneInsertValue，这三个OneInsertValue组成一个AllInsertValue，记录该条insert语句的全部插入值。

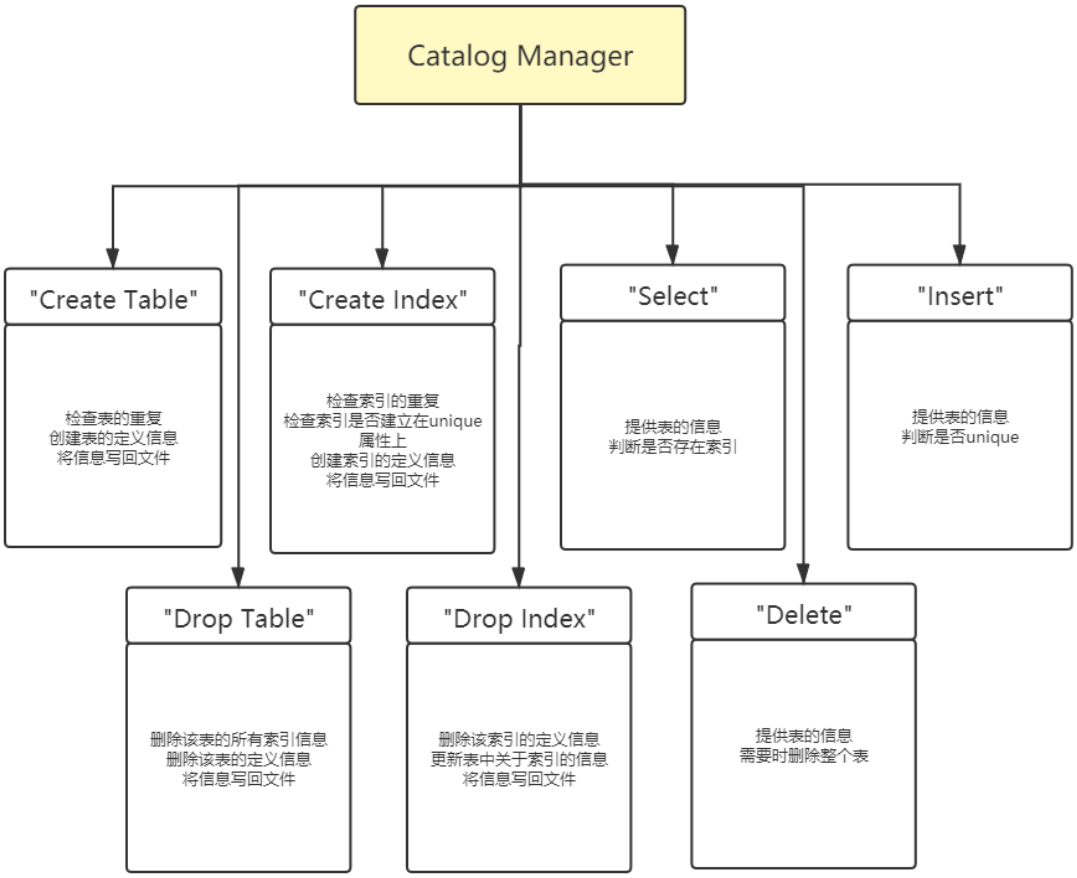
  API类有一个Buffer类的成员变量buf，用于记录当前Buffer的状态，在整个程序运行中只能存在一个Buffer。



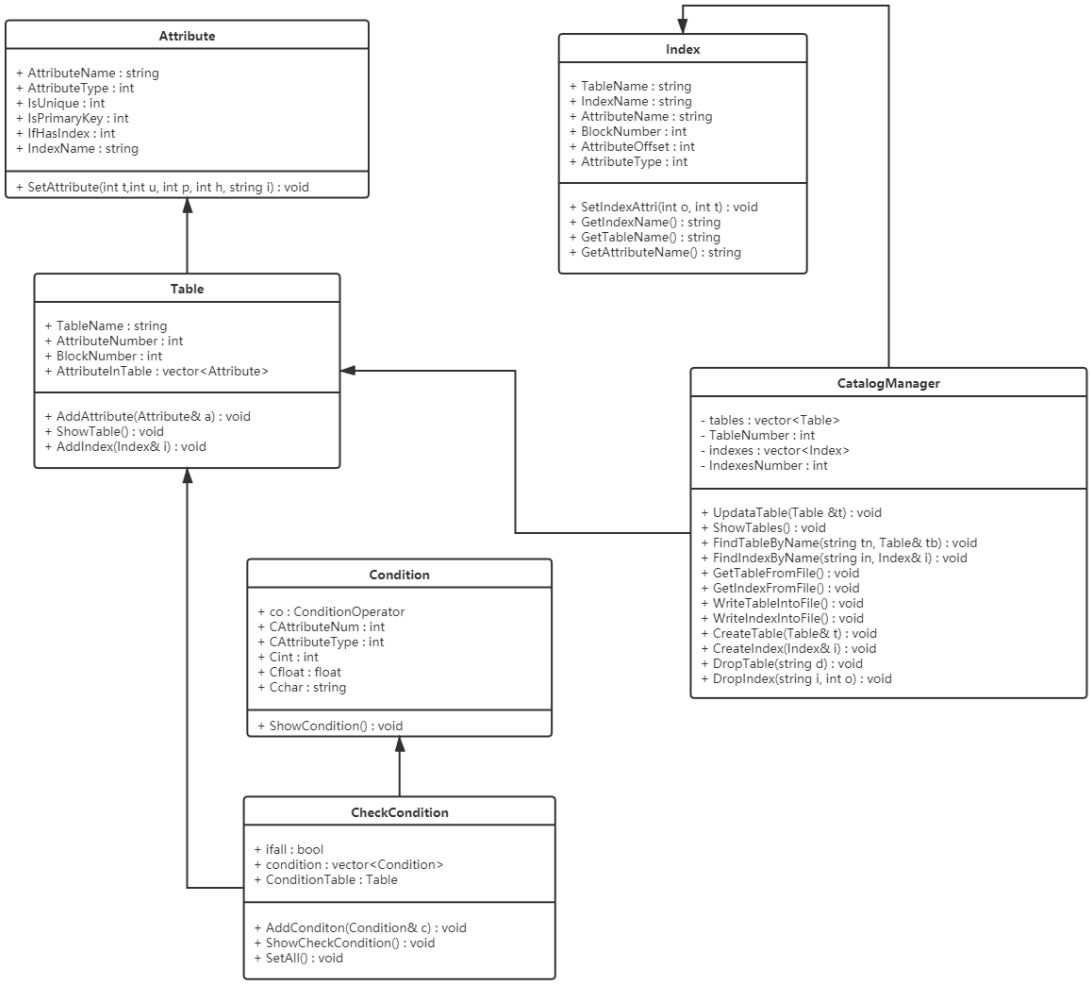
4. Catalog Manager

- **功能描述**

  Catalog Manager的主要功能为管理保存数据库的表和索引的信息的文件，使得在程序结束时能够保存表和索引的信息、给其他模块提供通用的类，如Attribute、Table、Index、Condition等、供Interpreter和API调用。



- **类图及类间关系**



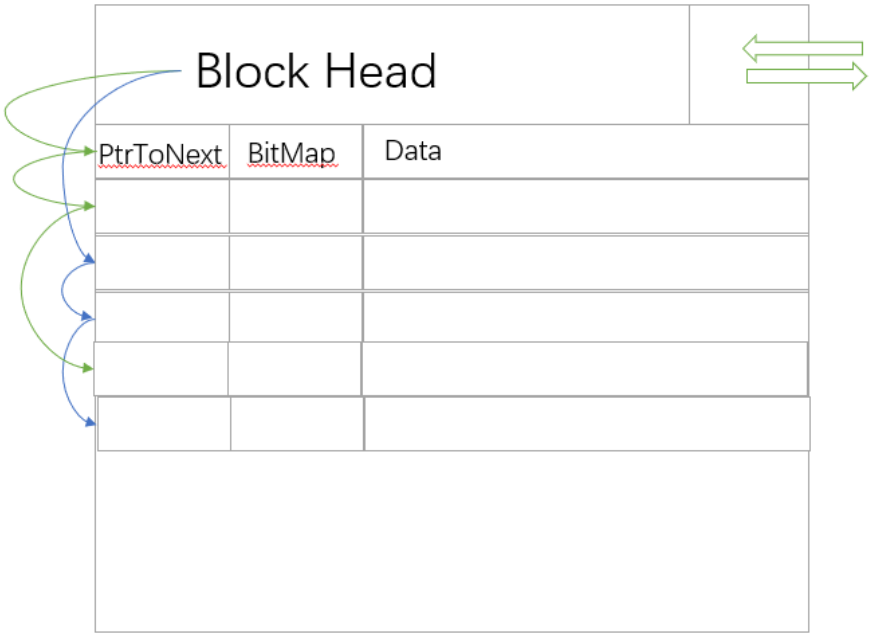
5. Record Manager

- **功能描述**

  Record模块负责管理Record对象和Block对象之间的关系，使得上层模块可以通过Record对象的方法，直接对Record进行操作，便于SQL语句的编写。同时该模块也维护Table和其数据存储的Block的绑定关系。

- **数据结构**

  Block的内部结构如图所示：



其中BlockHead维护该块的相关参数，以及两条链表：绿色箭头链表、蓝色箭头链表

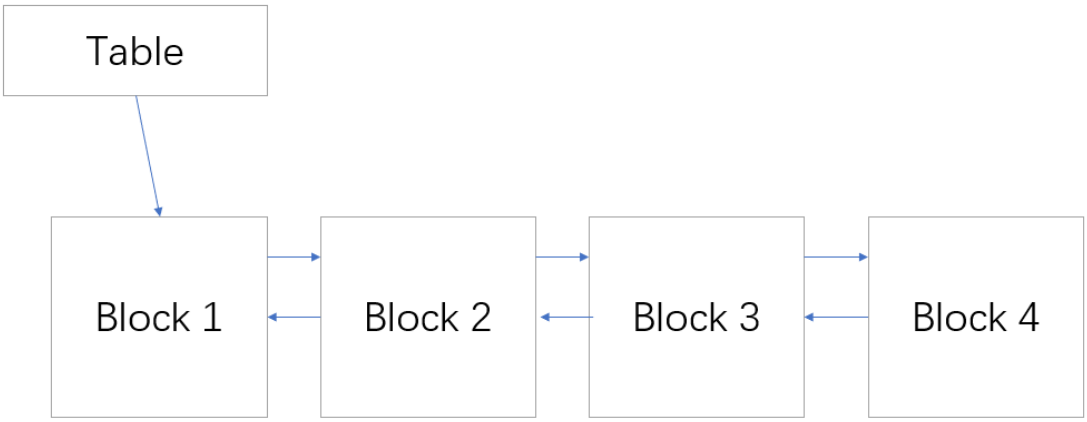
Head下方每一行代表一个Record，Record存储了next指针、Bitmap内容和实际的Data内容

绿色箭头链表代表已经存入Block的数据

蓝色箭头链表代表空的Record槽位

每当插入Record的时候，该模块会把一个空Record槽位写入数据，然后移动到绿色箭头链表中

所有的Table都对应一个Block的双向链表，如图所示：



插入数据时，需要遍历这个双向链表，并找到空槽位来插入

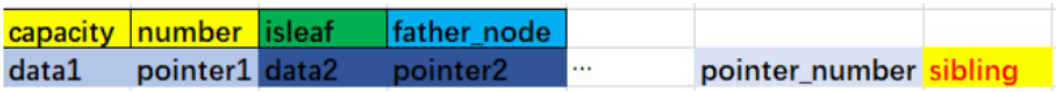
6. Index Manager

- **功能描述**

  Buffer Manager负责索引的构建、删除；含索引的表的查询、插入和删除元素

  整体上维护一个在索引属性上构建的B+树，实现了该B+树的叶子结点增删查功能

- **数据结构**
  - 非叶子结点



头部先存储4个信息数据，在之后紧接着存储数据和指针。

- capacity：该节点的最大容量；
- number：该节点现有的数据数目；
- isleaf：该节点是否为叶子节点；
- father_node：该节点的父亲节点（如果为根节点，则值为0）；
- data+pointer：每个data存储的是这个pointer节点中第一个data，和传统的树不同，这样设计是为了更好的指导该树存储信息中的最小值。
- sibling：指向兄弟节点，在遍历的时候非常方便，需要不断更新。

- 叶子结点

- 访问数据：通过RecordAccess类作为指针来访问

```
class RecordAccess {
public:
    BlockID bid;
    Offset offset;
};
```

- Value结构：通过该类进行增删查的条件传递

```
struct Value {//支持两个操作数，5种比较方式
public:
    vector<Record> search_result;
    int int_value[2];
    float float_value[2];    //上层对有效数据进行赋值，其他赋值不影响
    string string_value[2]; //根据index_info进行判断取值
    int type[2];             //-2 > ; -1 >= ; 0 == ; 1 <= ; 2 < ;
    int number;              //操作数的数量
};
```

7. Buffer Manager

- **功能描述**

  在我们实现的系统中，所有数据库的record和index都存储在一个文件里：MiniSQL_Data.dat。Buffer模块是Record、Index等上层模块和该文件进行交互的接口。其主要实现了数据文件的按块存储和按块访问，管理数据文件的打开和关闭，同时通过BlockID和BlockOffset向上层提供访问数据文件的手段。

- **数据结构**

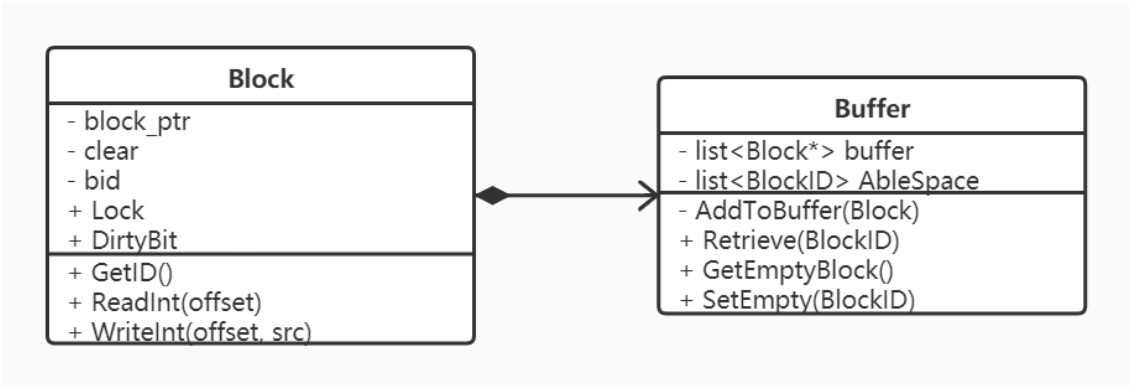
  Buffer在内存中申请八个块的空间，以链表的形式将八个块连接起来：

Buffer	0	1	2	3	4	5	6	7

  通过向Buffer提供BlockID，即可访问到对应序号的块

  通过向Block提供Offset，即可按字节访问到Block的内容(Block[offset])

- **类图**



8. DB Files

该MiniSQL程序维护三个文件：*MiniSQL_Data.dat, MiniSQL_IndexInfo.dat, MiniSQL_TableInfo.dat*

分别存储了所有record、index的元数据、table的元数据

其中*MiniSQL_Data.dat*通过Buffer按块访问，后两者由对应模块直接访问进行管理

五、系统实现分析及运行截图

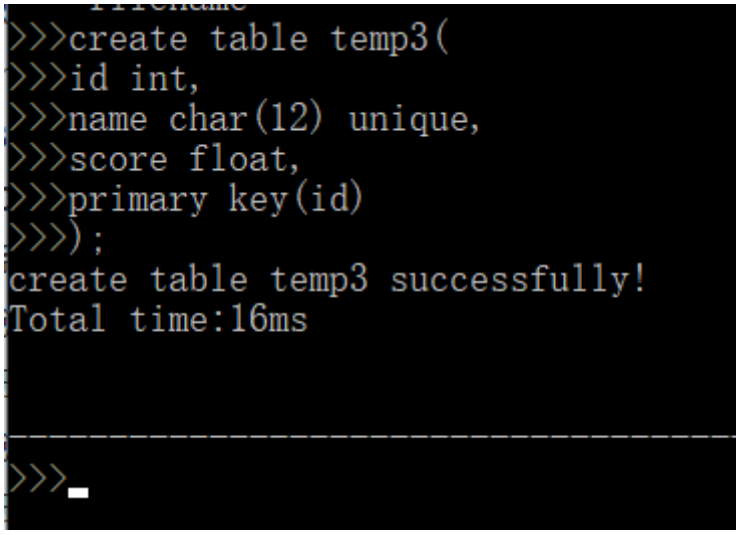
1. 创建表（create table）

- 执行流程

  Interpreter解析语句；Catalog写入table的元数据；API初始化Table对象，通过Record绑定到Buffer分配出的空块；如果需要，创建一个空的Index

- 截图验证

```
create table temp3(
    id int,
    name char(12) unique,
    score float,
    primary key(id)
);
```



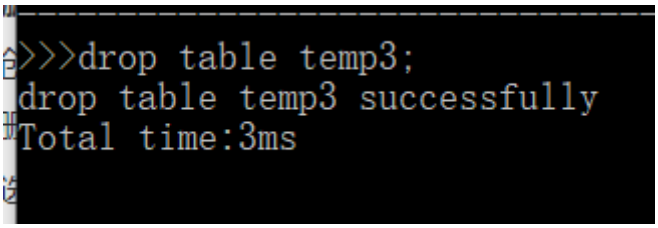
2. 删除表（drop table）

- 执行流程

  Interpreter解析语句；遍历Index存储的所有块，全部清空；遍历Record存储的所有块，全部清空；删除Index元数据和Table元数据

- 截图验证

```
drop table temp3;
```



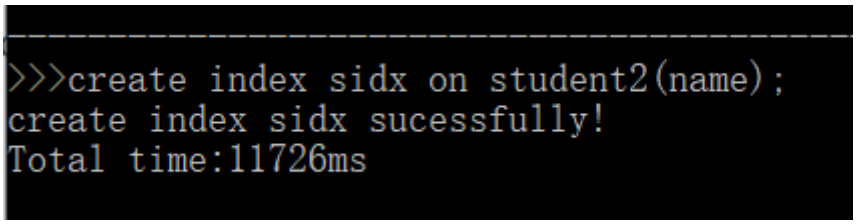
3. 创建索引（create index）

- 执行流程

  Interpreter解析语句；通过Record读取Table的对应属性列；Index进行排序，通过Buffer获得空块，建立B+树

- 截图验证

```
create index sidx on student2(name);
```



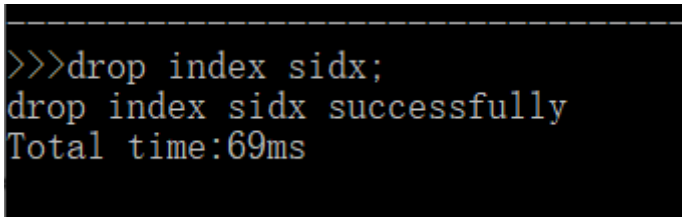
4. 删除索引（drop index）

- 执行流程

  Interpreter解析语句；遍历Index存储的所有块，全部清空；删除Index元数据

- 截图验证

```
drop index sidx;
```



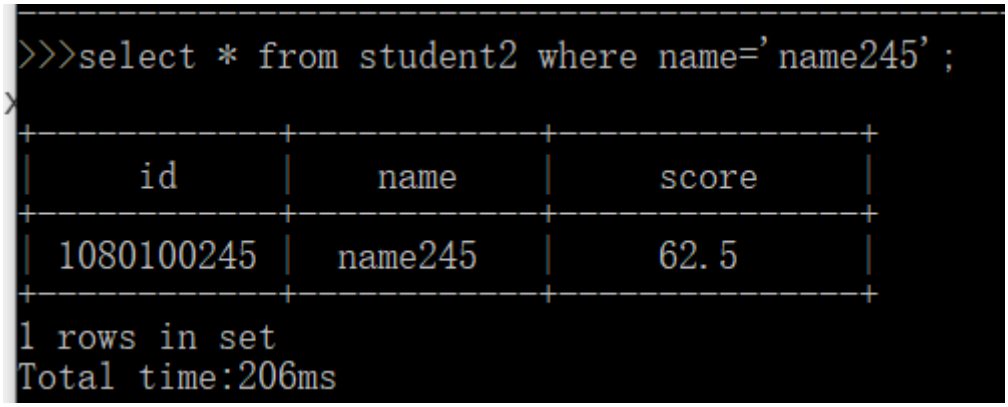
5. 选择语句（select）

- 执行流程

  Interpreter解析语句；初始化Condition状态；如果条件中包含Index，优先使用B+树检索，获取所有Record；否则调用Record模块的接口，进行线性搜索；最后调用DrawTable()

- 截图验证

```
select * from student2 where name='name245';
```



6. 插入记录（insert）

- 执行流程

  Interpreter解析语句；调用Record模块插入一条数据；如果有Index，调用Index插入函数，更新B+树

- 截图验证

```
insert into student2 values (1080100245, 'name245', 62.5);
```

```
>>>insert into student2 values (1080100245, 'name245', 62.5);
Insert Values into student2 Successfully!
Total time:207ms
```

7. 删除记录（delete）

- 执行流程

  Interpreter解析语句；调用Record模块删除一条数据；如果有Index，调用Index删除函数，更新B+树

- 截图验证

```
delete * from student2 where name='name245';
```

```
>>>delete * from student2 where name='name245';
1rows deleted
Total time:3189ms


>>>select * from student2 where name='name245';
+------------+------------+----------------+
|     id     |    name    |     score      |
+------------+------------+----------------+
|           no record returned             |
+------------+------------+----------------+
0 rows in set
Total time:234ms
```

8. 执行SQL脚本文件（execfile）

- 执行流程

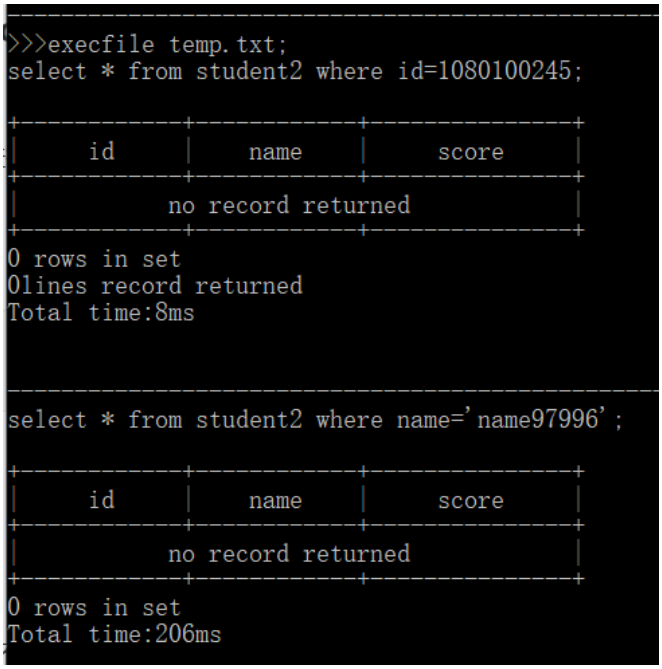
  读入所有语句，Interpreter进行解析；分步执行

- 截图验证

  temp.txt:

```
select * from student2 where id=1080100245;
select * from student2 where name='name97996';
```

  输入execfile：



9. 退出MiniSQL（quit）

- 执行流程

  析构Buffer对象，将内存中所有脏块写回硬盘；结束程序

- 截图验证