MiniSQL总体设计报告

1. MiniSQL系统概述

本组完成Bonus

1. **团队人数少于等于3人完成**minisql，**且程序功能完整**

   (**验收当天下午六点前修复了**bug，**钉钉上已经把成功的演示录像发给助教，并且请求当天重新验收，报告编写时助教还未回复**)。

2. **程序能显示各条语句执行时间，验收时能解释清楚**

1.1 背景

1.1.1 编写目的

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，提高系统编程能力，加深对数据库系统原理的理解。**

1.1.2 项目背景

随着数据库系统课程的推进，在学习一定数据库系统知识的基础上，需要一次基于数据库系统的项目实践来加深知识的理解与掌握。精简型单用户SQL**引擎**MiniSQL**正满足了这一需求，符合本科学生知识储备的同时包含了**SQL**引擎的基本功能。**

1.2 功能描述

1. **数据类型**

支持三种基本数据类型： `int` **，** `char(n)` **，** `float` **，其中** `char(n)` **满足** 1<=n<=255**。**

2. **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。**

创建表语法如下：

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

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

删除表语法如下：

```
drop table 表名 ;
```

3. 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）。
创建索引语法如下：

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

删除索引语法如下：

```
drop index stunameidx;
```

4. 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。

选择语法如下：

```
select * from 表名 ;
```

或：

```
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：

`=`,`<>`,`<`,`>`,`≤`,`≥`;

5. 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。

插入记录语法如下：

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

删除记录语法如下：

```
delete from 表名 ;
```

或：

```
delete from 表名 where 条件 ;
```

6. **退出MiniSQL系统**

退出语法如下：

```
quit;
```

1.3 运行环境和配置

集成开发环境：Windows10 Visual Studio 2019

1.4 参考资料

《数据库系统概念》第六版 机械工业出版社

《C++ Primer》**第五版 电子工业出版社**

2. MiniSQL系统结构设计

2.1 总体设计

2.2 Interpreter 模块

2.2.1 概述

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。**

2.2.2 结构与接口

```
// 获得需要处理的字符串
+private std::string getword(std::string s, int *i);
// 将字符串中所有大写字母转化为小写字母
+private std::string getlowerword(std::string s);
// 处理字符串后返回一个指向Sentence类的智能指针
+public std::shared_ptr<Sentence> parseSql(std::string s);
```

2.3 API模块

概述

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。

结构与接口

```
APP();
// 开始运行数据库，包括登录，调用interpreter解析语句，调用API模块执行语句.
void run();
/// Executes the SQL.
void execSql(std::shared_ptr<Sentence> parseResult);
```

2.4 CatalogManager模块

概述

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。

结构与接口

```
+public int test();
// Initializes a new instance of the <see cref="CatalogManager"/> class.
+public CatalogManager();
// Closes this instance.
+public void close();
// Gets the attribute's cata.
+public catalog getCataByAttrName(std::string tableName, std::string attrName);
+public anyVec getCataInAnyVec(std::string tableName);
+public std::vector<std::string> getAllTableNames();
+public std::vector<std::string> getAllAttrByTableName(std::string tableName);
+public void createTable(CreateTableSentence sent);
+public void dropTable(std::string tableName);
```

2.5 RecordManager模块

概述

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。

结构与接口

```
+public int test();
+public RecordManager(std::shared_ptr<BufferManager> ptr);
// Inserts the record to table.
+public int64 insertRecordToTable(std::string tableName, anyVec values);
// Removes the record from table.
+public std::vector<int64> removeRecordsByAddressAndCondition(std::string
tableName, std::vector<int64> addresses, std::vector<condition> conds);
// Selects the records by address and condition.
+public anyVec selectRecordsByAddressAndCondition(std::string tableName,
std::vector<int64> addresses, std::vector<condition> conds);
```

2.6 Indexmanager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。**

结构与接口

```
class IndexManager {
//初始化IndexManager模块并从文件读取数据
+public IndexManager(std::shared_ptr<CatalogManager> _catalogManager);
//关闭时将数据写入文件
+public void close();
//建立B+树并返回根节点指针
+public void* createIndex(std::string indexName, std::string tableName,
std::string attrName);
//删除指定表中的指定索引
+public void dropIndex(std::string indexName, std::string tableName);
//删除指定表中的所有索引
+public void dropAllIndex(std::string tableName);
//根据指定索引和指定表向b+树中插入keyvalue对
+public insertToIndex(std::string indexName, std::string tableName, std::any
key, int64 value);
//根据条件和指定的表名、索引名获得索引
+public std::vector<int64> selectIndexsByCondition(std::string tableName,
std::string indexName, std::vector<condition> cond);
//在指定表中根据指定地址删除索引
+public void removeIndexByAddress(std::string tableName, std::vector<int64>
addresses);
//判断指定表中是否有聚集索引
+public bool hasClusteredIndex(std::string tableName);
//返回指定表中的聚集索引
+public std::string getClusteredIndex(std::string tableName);
//判断指定表中是否有非聚集索引
+public bool hasNonClusteredIndex(std::string tableName);
```

```
// 返回指定表中的非聚集索引
+public std::vector<std::string> getNonClusteredIndex(std::string tableName);
// 判断指定表中是否有索引
+public bool hasAnyIndex(std::string tableName);
// 返回指定表中的所有索引
+public std::vector<std::string> getAllIndex(std::string tableName);
// 在指定表中根据索引名返回属性名
+public std::string getAttrNameByIndexName(std::string indexName, std::string
tableName);
// 在指定表中根据属性名返回索引名
+public std::string getIndexNameByAttrName(std::string attrName, std::string
tableName);
```

2.7 BufferManager模块

概述

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**。**

结构与接口

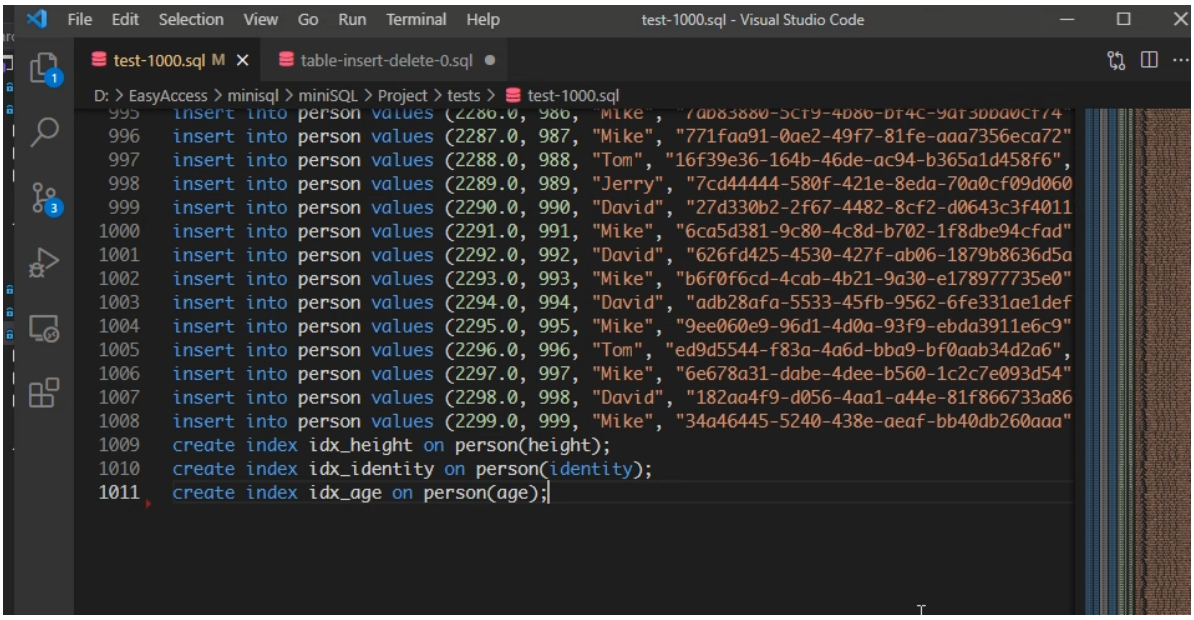
```
+private int64 getBlockNum(std::string tableName);
// 代理引用buffer,通过修改这个函数可以管理缓冲区
+private DataBlock& refBlockByLabel(std::string tableName, int64 blockSerial);
// 代理插入buffer,通过修改这个函数可以管理缓冲区
+private void insertBlock(std::string tableName, int64 blockSerial, DataBlock
d);
+public int test();
// Initializes a new instance of the <see cref="BufferManager"/> class.
+public BufferManager(std::shared_ptr<CatalogManager> ptr);
// Gets the record number.
+public int64 getRecordNum(std::string tableName);
// Inserts the record to table.
+public int64 insertRecordToTable(std::string tableName, anyVec values);
// Gets the record by address.
+public anyVec getRecordByAddress(std::string tableName, int64 addresses);
// Deletes the record by address.
+public void deleteRecordByAddress(std::string tableName, int64 addresses);
// Creates the table.
+public void createTable(std::string tableName);
// Drops the table.
+public void dropTable(std::string tableName);
```

3. 测试方案和测试样例

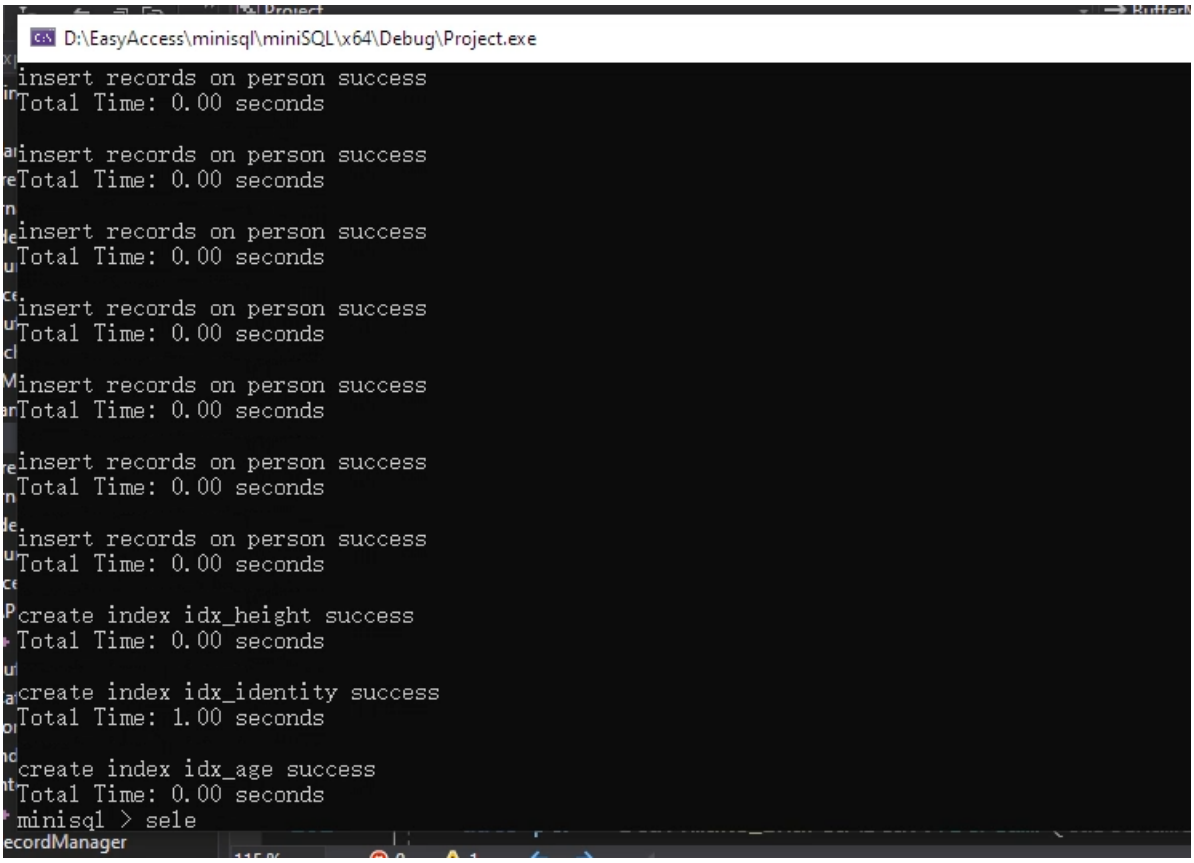
学在浙大上给出的所有test文件均可单次通过。

验收时导致失败的bug在当天下午已经修复完成并且录制了视频，视频已经在钉钉上发给助教老师，并附于压缩包内。

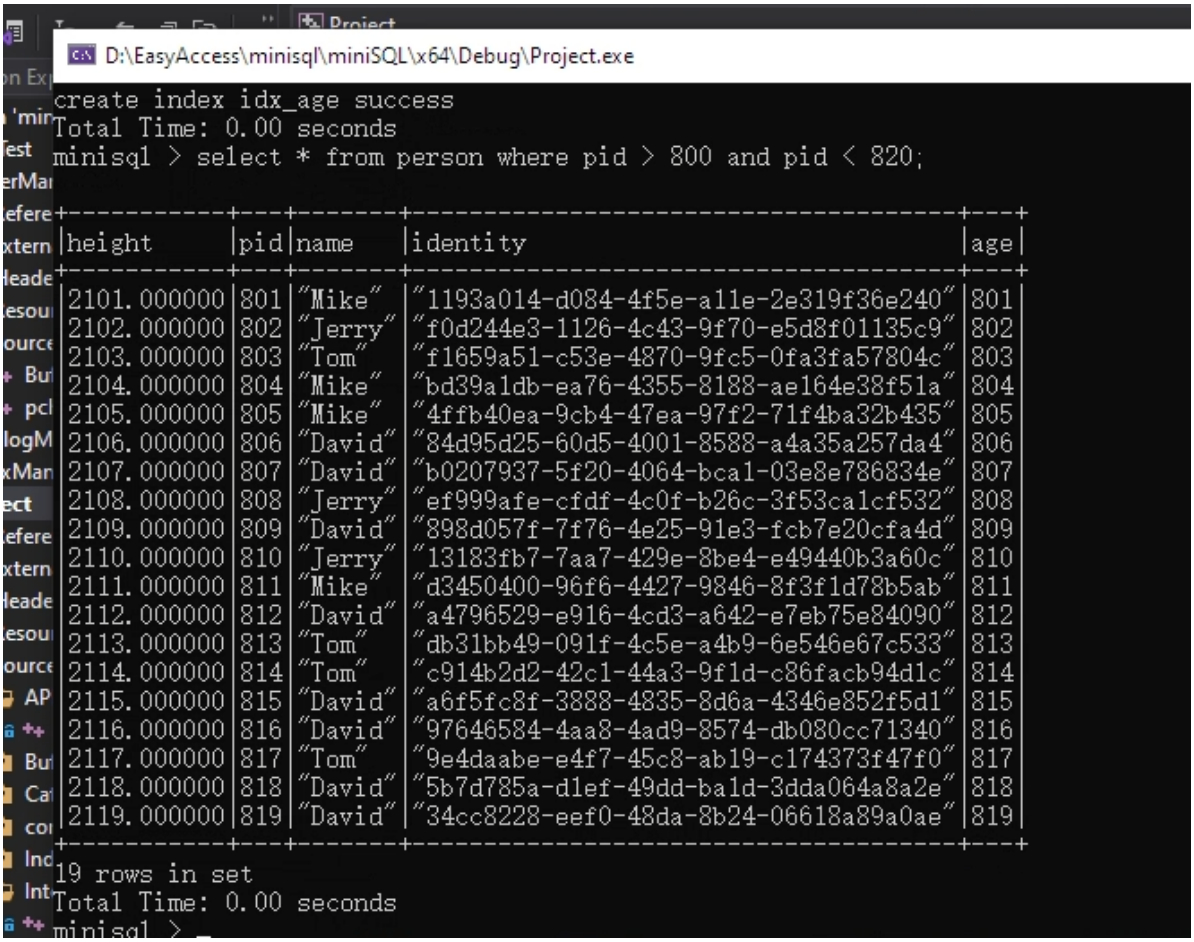
下面给出1000组数据的截图



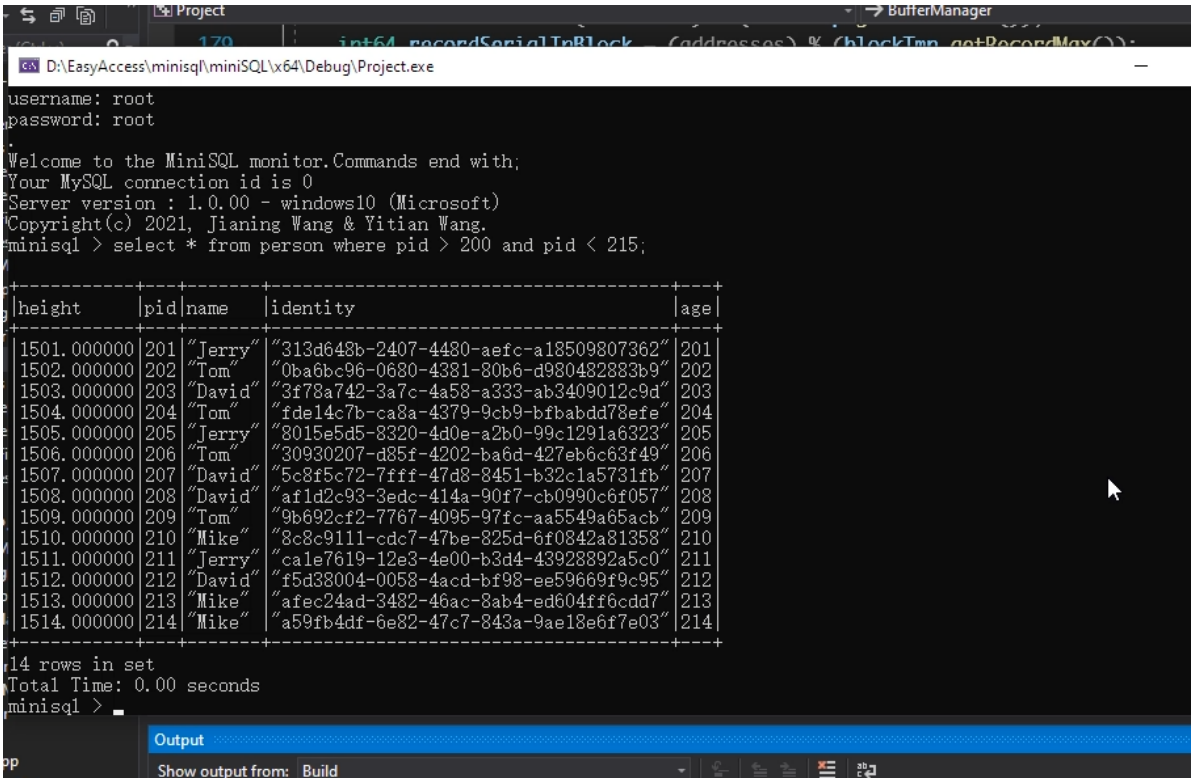
测试结果如下



试图进行查询，结果如下：



重启程序，再次查询，观察是否能够保存表，结果如下



总体上来讲，虽然验收时效果不佳，但经过下午短时间的调试，发现是interpreter部分无法识别符号和数字连在一起的情况，修改后能够正常演示，总的来说，实验结果比较成功。

4. 分组与设计分工

本组成员

姓名	学号
王佳宁	3190105248
王奕天	3190102362

本组分工

姓名	分工
王佳宁	**项目架构设计** API**模块** CatalogManager**模块** RecordManager**模块** BufferManager**模块** **调试** **报告撰写**
王奕天	Interpreter**模块** IndexManager&BplusTree**模块** **报告撰写**