MiniSQL设计报告

陈则衔、卢涛、於航

时间：2016.6.19

**目录**

[1. MiniSql总体框架 2](#_Toc30984)

[1.1 MiniSql 实现功能分析 2](#_Toc15065)

[1.2 MiniSql系统体系结构 2](#_Toc5392)

[1.3 设计语言与运行环境 2](#_Toc12316)

[2. MiniSql各模块实现功能 3](#_Toc2005)

[2.1 Interpret实现功能 3](#_Toc21568)

[2.2 API实现功能 3](#_Toc18860)

[2.3 Catalog Manager实现功能 3](#_Toc6418)

[2.4 Record Manager实现功能 3](#_Toc7374)

[2.5 Buffer Manager实现功能 3](#_Toc30976)

[2.6 Index Manager实现功能 3](#_Toc13989)

[3. 内部数据形式及各模块提供的接口 3](#_Toc11704)

[3.1 内部数据形式 3](#_Toc1130)

[3.2 主窗口及主函数设计 5](#_Toc10440)

[3.3 Catalog Manger接口 6](#_Toc28097)

[3.4 Record Manger接口 7](#_Toc17995)

[3.5 Buffer Manger接口 7](#_Toc19774)

[3.6 Index Manger接口 7](#_Toc12769)

[4. MiniSql系统测试 7](#_Toc22000)

[5. 分工说明 10](#_Toc30855)

1. **MiniSql总体框架**

**1.1 MiniSql 实现功能分析**

1）总功能：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；

2） 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 <= 1023；

3）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；

4）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）；

5）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；

6）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。

**1.2 MiniSql系统体系结构**

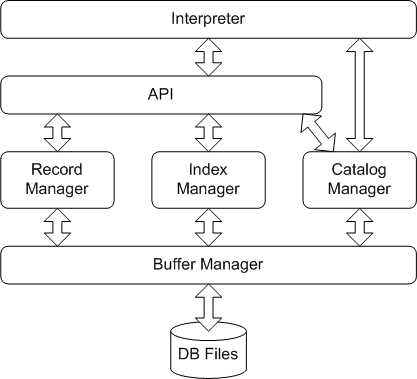
****

图 1.1 MiniSql体系结构

**1.3 设计语言与运行环境**

**工具：Microsoft Visual Studio 2015 环境：Win10**

1. **MiniSql各模块实现功能**

**2.1 Interpret实现功能**

读入SQL语句：每次读入一行，判断最后一个字符是不是‘；’，如果不是的话，就把它加入到输入的字符串中，继续读入下一行，直到读入的最后一个字符为‘；’则停止输入，最终得到的字符串即为输入的SQL语句。

解析SQL语句：将得到的输出分割成单个的单词或者是符号，如果发现语法上存在错误，就打印错误信息，如果语法没有错误，根据catalog的信号，判断是否能够执行该查询，如果不能，打印错误信息，如果可以，就调用API进行执行。

**2.2 API实现功能**

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调用。该模块接受interpreter 提供的解析命令，再根据catalog 的信息，来确定执行规则，并调用record manage，index manage，catalog manage来执行相应的操作，并将结果返回给interpreter 模块。

**2.3 Catalog Manager实现功能**

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、表中列数、primary key、定义在该表上 的索引信息。

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，包括所属表的信息、索引建立在那个字段上等。

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

**2.4 Record Manager实现功能**

Record Manager主要针对API的一些请求对Buffer中的Record进行管理，同时返回API所预设的Data类型。通过主要调用Buffer Manager的函数以及Table类参数对Buffer中指定Block的指定区间进行修改。实现了条件比较，对于int、float、char这3中变量类型的6中比较条件进行定义，并实现无条件和有条件的查找、插入、删除。

**2.5 Buffer Manager实现功能**

Buffer Manager模块是一个独立的，专门管理Buffer，从磁盘文件中读写数据的功能模块。Buffer Manager负责读取Block缓存至Buffer，同时向其他模块提供Buffer访问、读写功能，并能将Buffer中的Block写回磁盘文件中。Buffer Manger管理有限个Block存放在Buffer中，它遵循LRU（最近最少使用）原则。

**2.6 Index Manager实现功能**

Index Manager负责B+树索引的实现，可以完成B+树的创建、删除、维护、等值查找、区间查找、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，均为4K，B+树的叉数由节点大小与索引键大小计算得到）。

1. **内部数据形式及各模块提供的接口**

**3.1 内部数据形式**

**Interpreter处理数据：**

一个单词一个单词地读入

1. 首先读入operation，比如create table，select等，
2. 再读入要进行操作的table name，index name等
3. 接着读入value值
4. 最后读入操作的条件。

SQL结构大致如下：

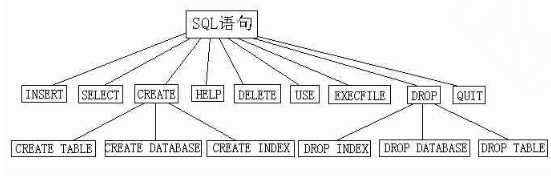


图 3.1 SQL语句结构

执行完SQL语句的分析后，返回的是一个Interpreter类，它包含以下内容：

|  |
| --- |
| class interpreter{  public:  int operation; //操作  string tablename; //要进行操作的table name  string indexname; //要进行操作的index name  string colname; //要进行操作的列  vector <Condition> conditions; //条件向量  vector <Attribute> column;  Table tableinfo; //table 信息  Index indexinfo; //index 信息  Row row;  int primarykeypos; //primary KEY的位置  int uniquekeypos; //unique KEY的位置  int wrongtype;  vector <Condition> uniqueconditions;  string filename; //要操作的文件名  ...  }; |

**B+树非叶结点结构设计：**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R | L | RecordNum | FatherPtr | LeftPtr | RightPtr | Key | ChildPtr | ... |

**B+树叶结点结构设计：**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R | L | RecordNum | FatherPtr | LeftPtr | RightPtr | Key | FileOff | BlockOff | ... |

其公共部分：

R: R--> 根节点 \_--> 非根节点

L： L--> 叶结点 \_--> 非叶结点

RecordNum(4Byte): 该结点拥有的键值数量

FatherPtr(4Byte): 指针指向父节点

LeftPtr(4Byte): 指针指向左兄弟

RightPtr(4Byte): 指针指向右兄弟

Key: 键值/关键词

其他：

ChildPtr： 指针指向子结点

FileOff: 文件中的Block位移

BlockOff： Block中的偏移量

**Catalog Manager数据：**

存放在catalog.index与catalog.table文件中，在创建Catalog实例时，便解析这两个文件并且读入到内存当中。由于这两个文件很小，而且存储形式与纯数据有些不同，用BufferManger反而会降低效率，因此由Catalog自主管理。

**BufferBlock数据结构：**

是一个数组为主的类，数组大小为1024，每个元素为占用大小为4096Byte的Block模块。于此同时，每一个Block中还记录：

该Block所对应的文件名: FileName；

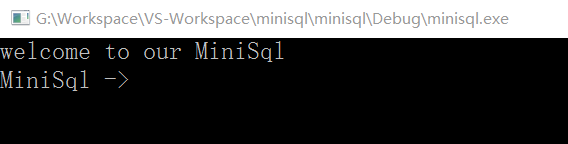
在该文件中的Block偏移量： BlockOffset；

该Block是否可用： inUse;

Block替换是的LRU值： LRUValue；

**3.2 主窗口及主函数设计**

运行时系统主界面如下：



SQL语句在MiniSql ->后输入；

**3.3 Catalog Manger接口**

1. 创建表：

|  |
| --- |
| void createtable(Table& tableinfo); |

1. 判断表是否存在：

|  |
| --- |
| bool isexisttable(string tablename); |

1. 建立索引：

|  |
| --- |
| void createindex(Index& indexinfo); |

1. 根据索引名判断索引是否存在：

|  |
| --- |
| bool existindex(string indexname); |

1. 根据表名和列数判断索引是否存在：

|  |
| --- |
| bool existindex(string tablename, int colnum); |

1. 根据表名获取table：

|  |
| --- |
| Table gettable(string tablename); |

1. 根据表信息和属性名获取列数：

|  |
| --- |
| int getcolunmnum(Table tableinfo, string word); |

1. 更新索引信息：

|  |
| --- |
| void update(Index indexinfo); |

1. 更新table信息：

|  |
| --- |
| void update(Table tableinfo); |

1. 删除table

|  |
| --- |
| void droptable(Table tableinfo); |

1. 根据索引名字获取索引信息：

|  |
| --- |
| Index getindexinfo(string indexname); |

1. 根据表信息和列数获取索引信息：

|  |
| --- |
| Index getindexinfo(Table tableinfo, int colnum); |

1. 删除索引：

|  |
| --- |
| void dropindex(string indexname); |

**3.4 Record Manger接口**

1.查找记录

|  |
| --- |
| Data select(const Table table, vector<Condition> conditions); |

2.插入记录

|  |
| --- |
| void insertRecord(Table & table, Row & splitRow); |

3.删除记录

|  |
| --- |
| int deleteRecords(Table table, vector<Condition> conditions); |

4.删除表

|  |
| --- |
| void dropTable(Table table); |

5.新建表

|  |
| --- |
| void createTable(Table table); |

**3.5 Buffer Manger接口**

1.获取块号

|  |
| --- |
| int getBlockNum(const string fileName, const int blockOffset); |

2.删除文件

|  |
| --- |
| void deleteFile(const string fileName); |

3.获取插入位置

|  |
| --- |
| InsertPos getInsertPosition(Table & table); |

1. 写回区块

|  |
| --- |
| int writeBack(const int blockNum); |

1. 读取区块

|  |
| --- |
| int readBlock(const string fileName, const int blockOffset, int blockNum); |

1. 获取空块

|  |
| --- |
| int getEmptyBlock(const string fileName = "", const int blockOffset = NOTFOUND); |

1. 更新LRU

|  |
| --- |
| void useBlock(int blockNum); |

**3.6 Index Manger接口**

1. 创建索引

|  |
| --- |
| void createIndex(const Table& tableinfor, Index& indexinfor); |

1. 删除索引，即删除索引文件

|  |
| --- |
| void dropIndex(Index& indexinfor); |

1. 查找操作

|  |
| --- |
| Data select(const Table& tableinfor, const Index& indexinfor, vector<int> num, vector<Condition> op, int blockOffset = 0); |

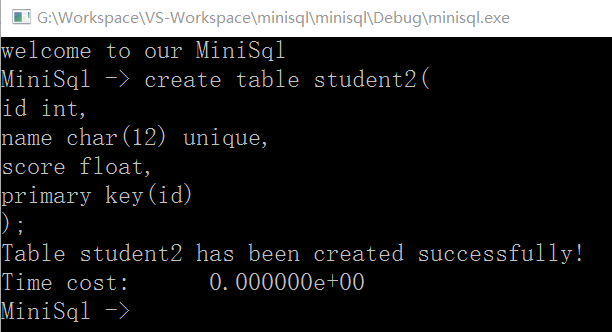
1. 插入新索引值，已有索引则更新位置信息

|  |
| --- |
| IndexBranch insertValue(Index& indexinfor, IndexLeaf node, int blockOffset = 0); |

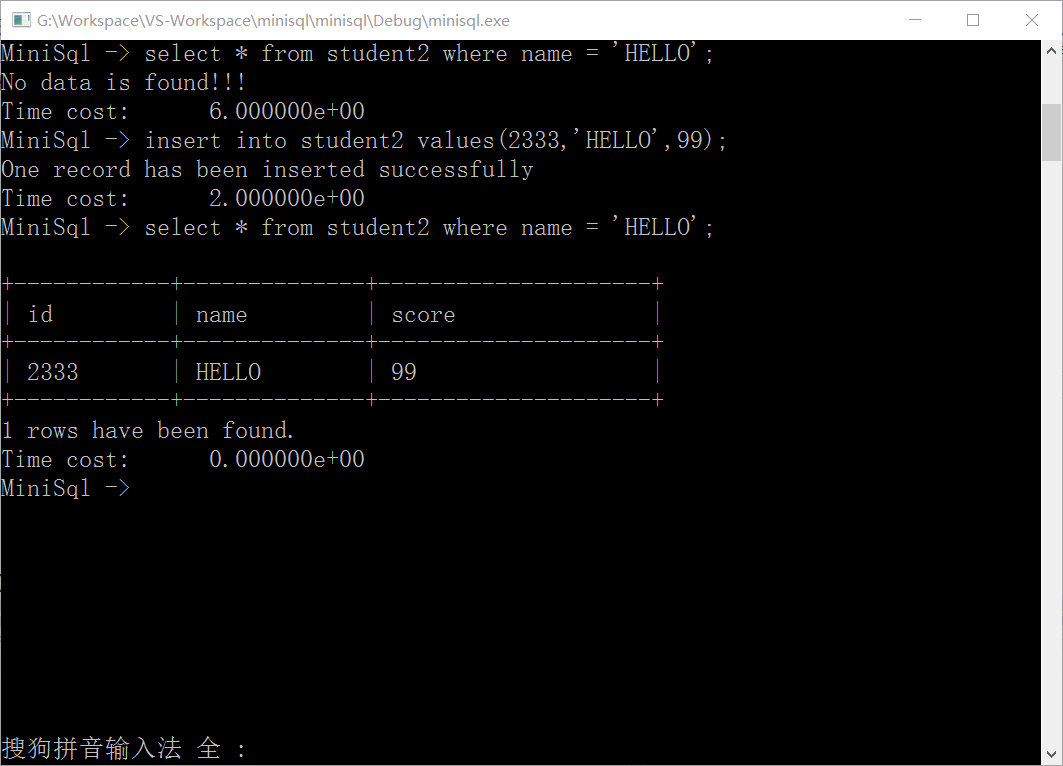
1. 删除索引值，没有该索引则什么也不做

|  |
| --- |
| void deleteValue() |

1. **MiniSql系统测试**
2. **创建表：**

****

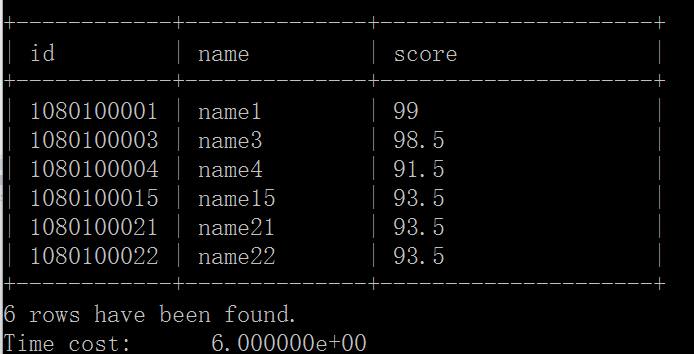
1. **插入数据：**



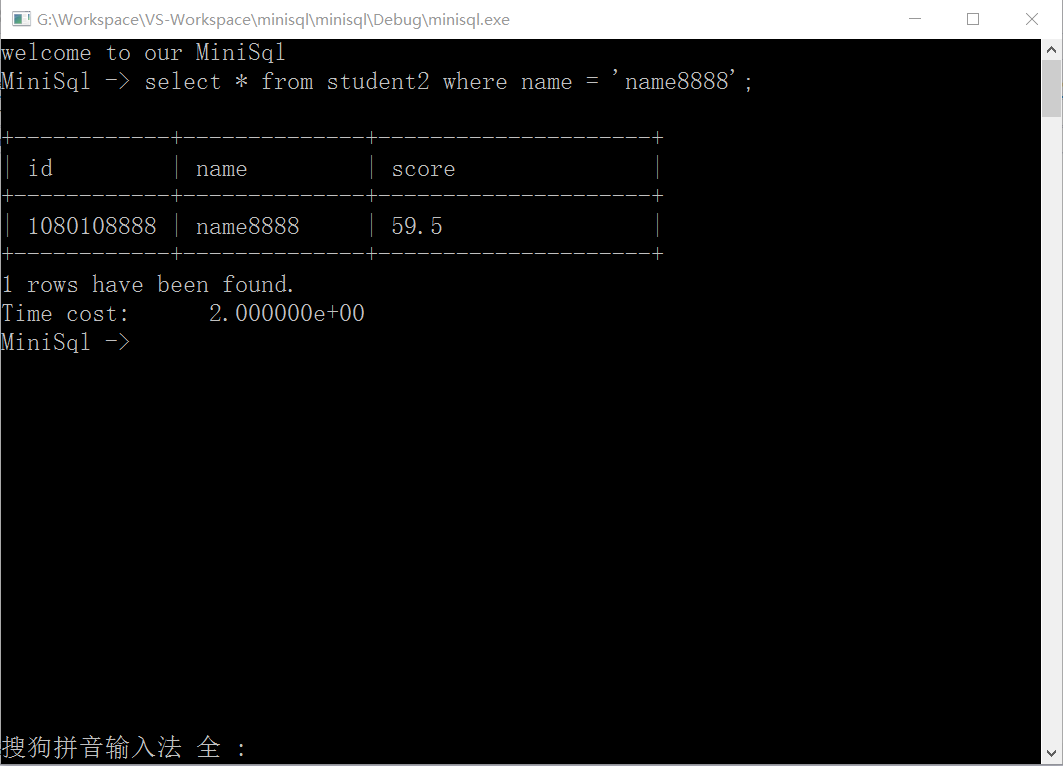
1. **多条件查找：**

**多值**

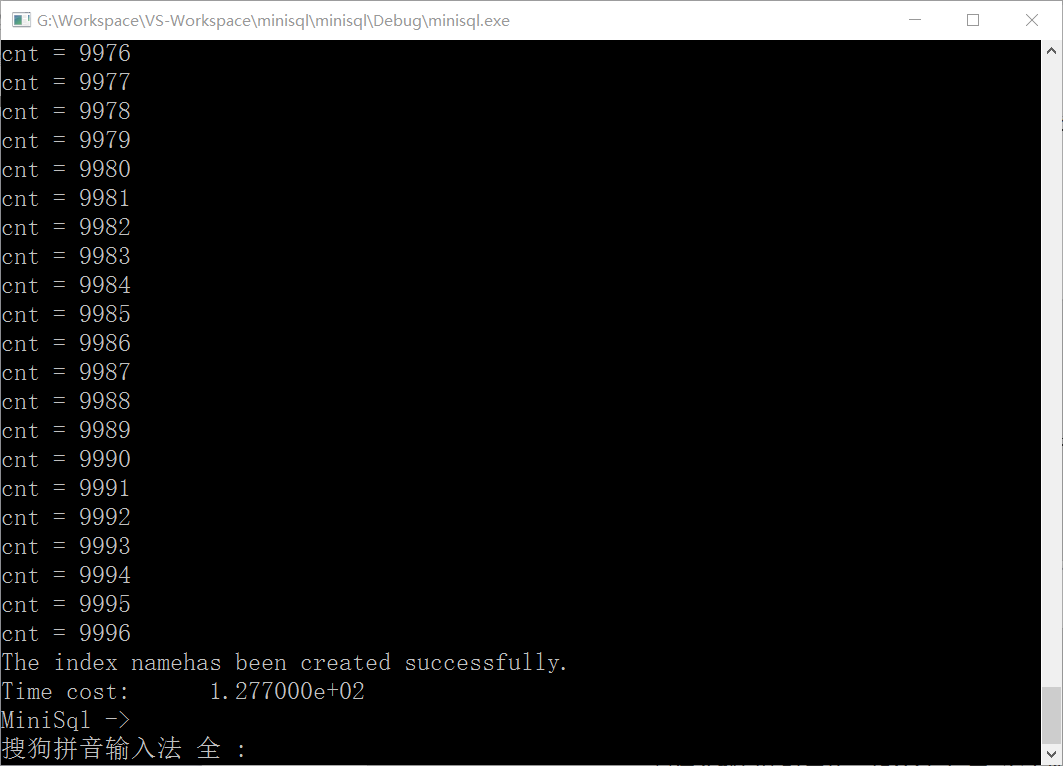
**结果：**

****

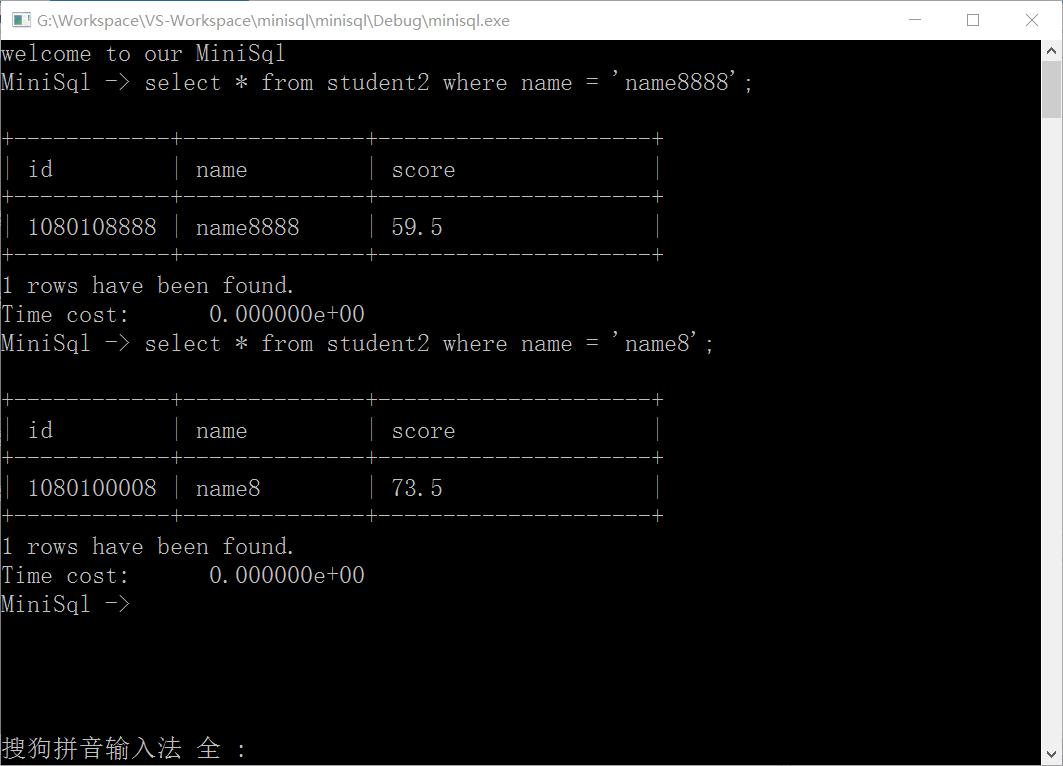
1. **无索引条件查找：**



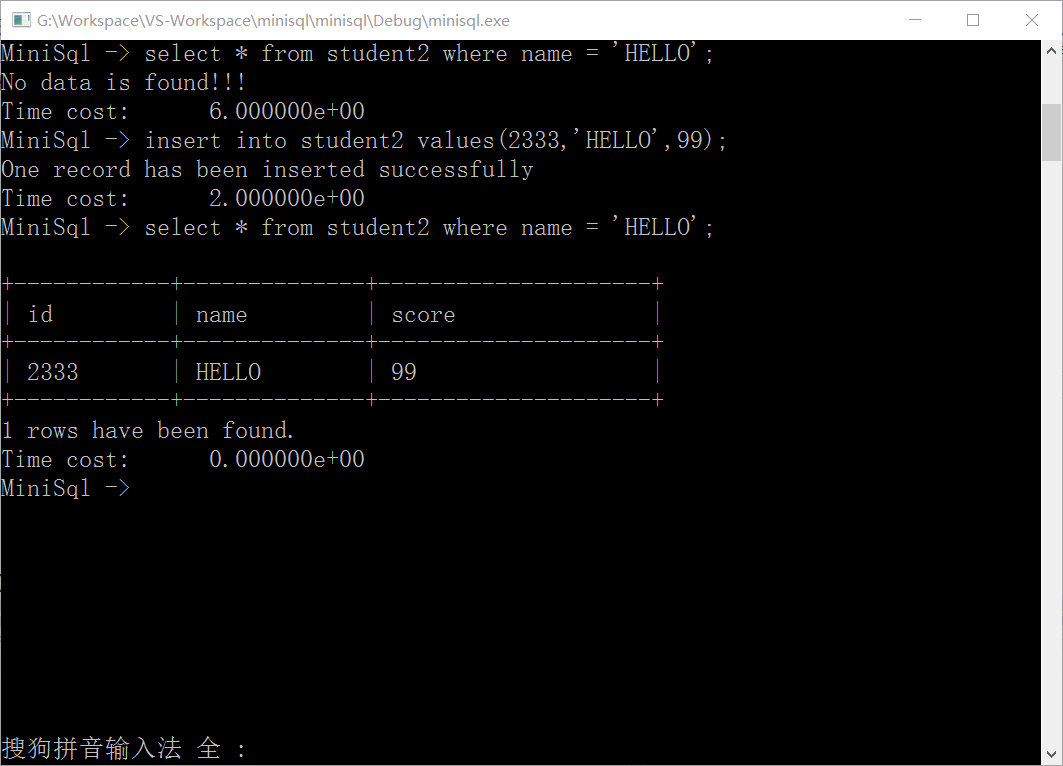
1. **创建索引：**



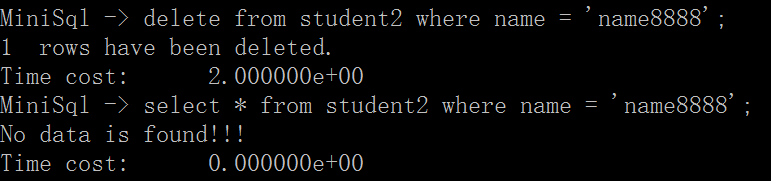
1. **有索引查找：**



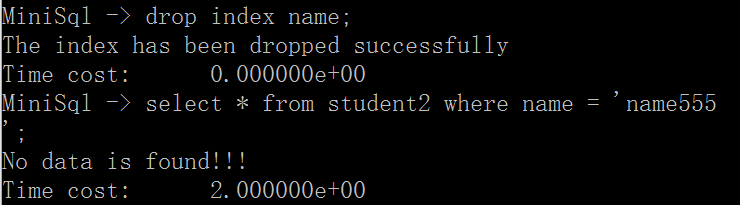
1. **索引插入：**



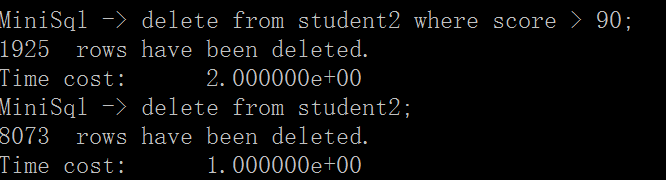
1. **索引删除：**

****

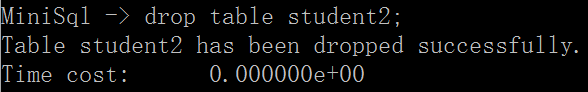
1. **删除索引：**

****

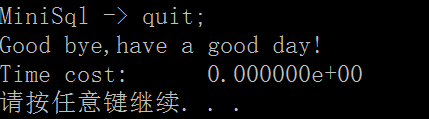
1. **删除记录：**

****

1. **删除表：**

****

1. **退出：**

****

1. **分工说明**

分工如下：

Interpret模块： 卢涛

API模块： 卢涛

Catalog Manger模块： 卢涛

Record Manger模块： 於航

Buffer Manger模块： 於航

Index Manger模块： 陈则衔

总设计报告： 陈则衔

模块汇总： 陈则衔

测试： 大家