

# 浙江大学



题    目：MiniSQL 数据库系统设计报告

授课老师：陈刚

课    程：数据库系统设计

MiniSQL 数据库系统设计报告.....	3
1. 引言.....	3
1.1 项目名称.....	3
1.2 项目背景和内容概要.....	3
2. 系统结构.....	3
2.1 系统功能.....	3
2.2 系统结构图.....	4
2.3 系统目录结构.....	4
2.4 基本设计概念.....	5
2.5 程序模块说明.....	6
Interpreter.....	6
词法解析.....	6
语法和语义分析.....	6
语法分析.....	7
语义分析.....	8
API.....	9
函数接口介绍.....	9
Catalog Manager.....	10
index 结构.....	10
table 结构.....	10
attribute 结构.....	11
CatalogManager 类.....	11
Buffer Manager.....	13
Block 类.....	13
Buffer Manager 类.....	13
Record Manager.....	15
tuple 类.....	16
conditionNode 类.....	16
RecordManager 类.....	18
File Manager.....	19
IndexManager.....	19
IndexManager 类.....	20
BPlusTree 类.....	20
B+树结构.....	24
3.测试结果.....	25
4.Bonus功能.....	40

# MiniSQL 数据库系统设计报告

## 1. 引言

### 1.1 项目名称

MiniSQL 数据库系统设计与实现。

### 1.2 项目背景和内容概要

数据库系统设计与实现实验。

**主要目的：**

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。

### 1.3 环境配置

在系统中新建环境变量为JAVA\_HOME，值设为C:\Program Files\Java\jdk.1.8.0\_212，再将PATH变量改为%JAVA\_HOME%\bin

## 2. 系统结构

### 2.1 系统功能

最终设计出来的MiniSQL 除了支持基本的数据库系统功能以外，还拓展设计了一些**附加功能（蓝色标记）**，具体支持的功能列表如下：

- Table 操作：包括表定义、表新建与表删除
- Insert 操作：单条数据的插入
- Select 基本操作：基本的表内容查询功能，显示所有的记录。
- **Projection 功能：支持查询结果的Projection。**

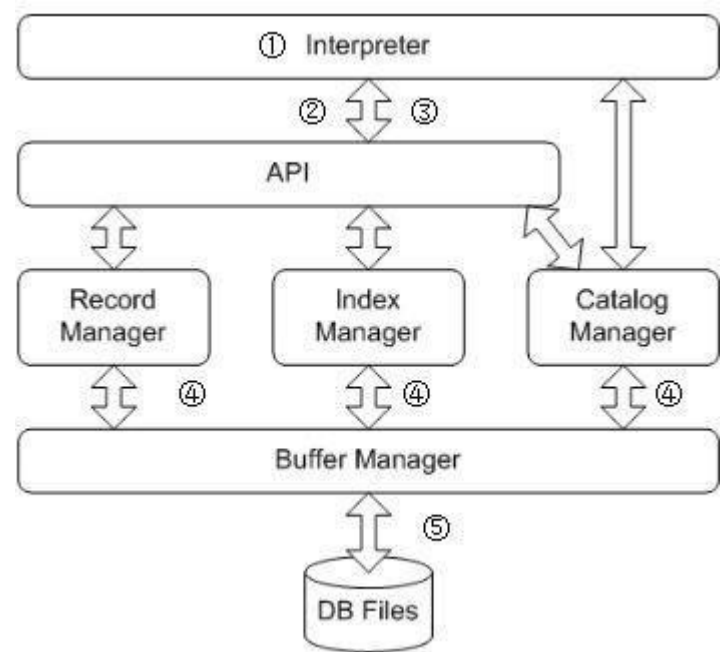
查询结果条件判断功能：

- 普通单属性条件语句判断。
- 带and 的条件语句：支持与条件查询。
- 带有or 的条件语句：支持或条件查询。**
- 带有括号的条件语句：支持带括号的条件有限查询功能。**
- Order by 排序功能：支持结果的排序功能。**
- Join 功能：支持表的连接功能。**

Delete 操作：数据的条件删除，支持多条。

Index 操作：支持 unique 属性的索引建立、删除以及数据更新时的维护。

## 2.2 系统结构图



图中各标号简明解释：

- ①判断并接受用户字符输入，使做为解释器的输入。
- ②解释器对用户输入进行翻译，调用 API 接口。
- ③执行选定的 API，返回用户所需的输出。
- ④BPlus、Record、Catalog 类调用 Buffer 类的方法实现自己各自的方法。
- ⑤Buffer 类方法对数据库文件进行直接操作。

## 2.3 系统目录结构

整个MiniSQL 的文件目录主要包括了 bin 和src 两个文件夹。

bin 目录下存放 MiniSQL 的 class 类文件，主要执行的都是这里的文件；

src 目录下存放 MiniSQL 的程序源文件，所有的功能更改与添加都通过修改其中的模块来完整，里面的程序清单如下表所示：

	模块名	程序文件名	语言	简要描述
用户接口	解释器	Interpreter.java	java	解释器模块实现文件
	API 集成	API.java	java	API 实现文件
系统内核	IndexManager	BPlusTree.java	java	B+树实现文件
		IndexManager.java	java	索引模块实现文件
		offsetInfo	java	索引结果文件
	RECORDMANAGER	conditionNode.java	java	查询条件类实现文件

		tuple.java	java	记录类实现文件
		RecordManager.java	java	RecordManager 实现文件
	CATALOGMANAGER	attribute.java	java	表属性类文件
		index.java	java	索引信息类文件
		table.java	java	表信息类文件
		CatalogManager.java	java	CatalogManager 实现文件
	BUFFERMANAGER	Block.java	java	块信息类文件
		BufferManager.java	java	BufferManager 实现文件
	FILEMANAGER	FileManager.java	java	FILEMANAGER 实现文件
	lexer	词法分析器		

其余的数据文件类型主要由以下几种：

**Catalog 文件：**主要有 index catalog.txt 以及 table catalog.txt，主要是为整个 MiniSQL 的数据库管理服务，保存了建立的表以及表信息、索引信息等总的数据库资料。

**.index 文件：**主要保存了相关的 B+树索引文件结构。

**Table 文件：**保存表中插入的相关 record 的信息。

## 2.4 基本设计概念

### 1. 系统目标

设计并实现一个精简单用户 SQL engine，并在其中实现表定义、索引、表记录操作功能。其中：

(1) 表定义中列（属性）的类型至少支持三种：integer、char、float（其中 char(n) 满足  $1 \leq n \leq 255$ ）；

(2) 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。

(3) 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）；

(3) 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，并能即时更新相应的索引；

(4) 记录的搜索至少实现按主键查找，支持主键上的范围查找，包含遍历。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。

### 2. 结构清晰

(1) 用户模块和内核模块完全分开。

(2) 内核部分 IndexManager、RecordManager、CatalogManager 三个模块不能访问物理文件，而由 BufferManager 模块实现物理文件操作的所有细节。（CatalogManager 模块在系统初始化以及退出时要读取和更新 Catalog 相关文件，这部分操作由于比较特殊所以独立于 BufferManager）。

(3) API 根据 IndexManager、RecordManager、CatalogManager 三个模块的方法整合生成，适合用户模块调用的 API。

## 2.5 程序模块介绍

### ● Interpreter

基本功能：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。

### 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	
	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
	Boolean getReaderState()	是否读取到输入流的结尾
	Token scan()	读取字节流，返回token，可通过token 标签进行判断单词类型判断

相关数据结构

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——字符串 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	功能描述
类变量	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

## 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。

支持语句

### 1. 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 )
);
```

### 2. 删除表语句

```
drop table 表名;
```

### 3. 创建索引语句

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

### 4. 删除索引语句

```
drop index 索引名;
```

### 5. 选择语句

```
select *(列名*) from 表名;
```

或：

```
select *(列名*) from 表名 where 条件;
```

或：

```
select *(列名*) from 表名 where 条件 order by 列名;
```

或：

```
select * from 表名1,表名2 where 表名1.列名=表名2.列名;
```

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

或：列op 值… and/or (列op 值and/or 列op 值) 支持括号优先级

其中op 可以为：<、<=、>、>=、=

### 6. 插入记录语句

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

### 7. 删除记录语句

```
delete from 表名;
```

或：

**delete from** 表名 **where** 条件；

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

或：列op 值… and/or （列op 值and/or 列op 值） 支持括

号优先级

其中op 可以为：<、<=、>、>=、=

## 8. 退出系统语句

**quit;**

## 9. 执行SQL 脚本语句

**execfile** 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。

### 1. create table 语义错误种类

- 1) table name 已存在
- 2) primary key 所指字段不存在
- 3) 出现重复的attribute 字段
- 4) char(n) 的n 越界

### 2. create index 语义错误种类

- 1) index name 已存在
- 2) table name 不存在
- 3) attribute 不存在
- 4) attribute 已经是索引
- 5) attribute 不是unique

### 3. drop table 语义错误种类

- 1) table name 不存在

### 4. drop index 语义错误种类

- 1) 该index name 不存在
- 2) 该index name 是主键不能删除

### 5. insert into 语义错误种类

- 1) table 不存在
- 2) 插入的tuple 数量不对
- 3) 插入的tuple 类型（及长度）不对
- 4) 对于unique key 字段有重复插入记录

### 6. delete 语义错误种类

- 1) table 不存在
- 2) where 条件有误：字段名不存在；value 属性与字段属性不匹配

### 7. select 语义错误种类

- 1) table 不存在



- 2) where 条件有误：字段名不存在；value 属性与字段属性不匹配
- 3) select 或order 的字段名不存在\*
- 4) 两字段属性不同无法比较\*

## API

### 函数接口介绍

	名称	功能描述	内部实现
数据	void Initialize()	Minisql 的初始化与退出	BufferManager 和 Catalog Manager 从文件中读取或数据库
	void close()		
	void showCatalog() void showTableCatalog() void showIndexCatalog()	显示 Catalog 信息	调用 CatalogManager 接口即可
sql 语句操作	boolean createTable(String tableName, table newTable)	创建表格	CatalogManager 添加表格和主键索引的定义信息；Recordmanager 创建存储记录的文件；Index Manager 对主键创建索引
	boolean dropTable(String tableName)	删除表格	CatalogManager 删除表格和表中所有索引的定义信息；Recordmanager 删除存储记录的文件；IndexManager 删除表中所有索引
	boolean createIndex(index newIndex)	创建索引	CatalogManager 添加索引定义信息；IndexManager 创建索引
	boolean dropIndex(String indexName)	删除索引	CatalogManager 删除索引定义信息；IndexManager 删除索引
	boolean insertTuples(String tableName, tuple theTuple)	插入记录	CatalogManager 更新表和索引信息；Recordmanager 添加记录；IndexManager 对 B+树中做添加

int deleteTuples(String tableName, conditionNodes)	删除记录	CatalogManager 更新表和索引信息；Recordmanager 删除记录，返回删除记录数；IndexManager 在 B+树中做删除
Vector<tuple> selectTuples(String tableName, Vector<String> attriNames, conditionNodes)	查询记录	CatalogManager 更新表信息（记录数改变）Recordmanager 返回查找结果
Vector<tuple> selectTuples(String tableName, Vector<String> attriNames, conditionNodes, String orderAttri, boolean ins)	查询记录（含 order 指令）	
Vector<tuple> join(String tableName1,String attributeName1,String tableName2,String attributeName2)	查询记录（含 join 指令）	

## 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。

## index 结构

功能描述：用于索引中的所有定义信息以及索引文件信息。其内部成员为：

```
public class index{
    public String indexName;//索引名，唯一标记索引
    public String tableName;//表名
    public String attriName;//字段名
    public int column;        //字段列数
    public int columnLength;//字段长度
    public int rootNum;       //根节点数目
    public int blockNum=0;    //index_name.table 占用block 数
}
```

## table 结构

功能描述：用于存储表的所有定义信息，包括表的名称、表中字段（列）数、主键、定义在该表上的索引。

其内部成员为：

```
public class table{
    String tableName;           //表名
    String primaryKey;          //主键名
    Vector<attribute> attributes; //以vector方式存放字段
    Vector<index> indexes;       //以vector方式存放表上的索引
    int indexNum;                //索引数量
    int attriNum;                //属性数量
    int tupleNum;                //记录条数
    int tupleLength;             //单条记录总字节数
}
```

attribute 结构

功能描述：用于存储Table中的每个字段信息

其内部成员为：

```
public class
    attribute{ String
        attriName; //字段名称
        String type; //字段类型：int/float/char
        int length; //字段字节数
        boolean isUnique; //字段是否为布尔型
    }
```

CatalogManager 类

	名称	功能描述	作用方法
成员变量	Hashtable<String,table> tables	表和索引的对象	哈希表容器存放表和索引，实现主键名和实例的一一对应关系
	Hashtable<String,index> indexes		
	String tableFilename="table catalog"	表和索引	
	String indexFilename="index catalog"	文件名	
函数接口	void InitialCatalog() void InitialIndexCatalog() void InitialTableCatalog()	初始化 Catalog	minisql 初始化时调用从文件中读取 Catalog 信息
	void storeCatalog() void storeIndexCatalog() void storeTableCatalog()	存储 Catalog	退出minisql 时调用将内存中的Catalog 信息写入文件中
	void showCatalog() void showIndexCatalog() void showTableCatalog()	显示 Catalog 信息	方便 Interpreter 调用查看当前 Catalog 信息
	boolean createTable(table newTable)	添加表格	Create table 操作调用
	boolean dropTable(String tableName)	删除表格	Drop table 操作调用

数据库操作接口		boolean createIndex(index newIndex)	添加索引	Create index 操作以及 Create table 的自动调用
		boolean dropIndex(String indexName)	删除索引	Drop index 操作以及 Drop table 的自动调用
		void addTupleNum(String tableName)	增加记录数	insert 操作调用更新表格记录数信息
信息交互接口		void deleteTupleNum(String tableName,int num)	减少记录数	delete 操作调用更新表格记录数信息
		boolean updateIndexTable(String indexName,index indexinfo)	更新索引信息	Insert 和delete 操作要更新索引信息(block 数等)
		boolean isTableExist(String tableName)	获取表的定义信息	主要用于 Interpreter 的语义判断以及 API、RecordManager 和 IndexManager 对 Catalog 信息的访问
		int getAttriNum(String tableName)		
		int getTupleLength(String tableName)		
		int getTupleNum(String tableName)		
		boolean isIndexExist(String indexName)	获取索引的定义信息	
		index getIndex(String indexName)		
		String getIndexName(String tableName,String attriName)		
		boolean isAttributeExist(String tableName,String attriName)	获取表中的字段定义信息	
		boolean inUniqueKey(String tableName,String attriName)		
		boolean isIndexKey(String tableName,String attriName)		
		int getAttriOffest(String tableName,String attriName)		
		String getType(String tableName,String attriName)		
		int getLength(String tableName,String attriName)		
		String getAttriName(String tableName,int i)		
		String getType(String tableName,int i)		
		int getLength(String tableName,int i)		

## Buffer Manager

代码结构：

BUFFER MANAGER 包中包含两个类。Block 类以及 Buffer Manager 类。

Block 是用来记录块内 4KB 大小的数据以及标记位的类。

Buffer Manager 类是用于管理buffer 的类。具体见下文。

### Block 类

功能：作为直接返回给Record Manager 或Index Manager 使用的数据块。记录了块内4KB 大小的数据以及标记位（脏数据位、有效位、reference 位、锁定位）。

类内各部分的功能描述及实现原理如下表所示：

	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
	void writeInt(int offset, int 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 offset, 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

## Buffer Manager 类

功能概述：

1. 根据需要，读取指定的数据(块)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（LRU），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，如是否是脏数据等
4. 提供缓冲区页的fix 功能，及锁定缓冲区的页，不允许替换出去，以提高效率。

类内各部分的功能描述及实现原理如下表所示：

	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 buffer,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 Manager 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 filename, 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 buffer 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。
	int findBlock(String filename, 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。

内部函数	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_bit为1，则把reference_bit位置1；如果指向的块reference_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。

## Record Manager

代码结构：

RECORD MANAGER 包中包含三个类。tuple 类, ConditionNode 类以及 recordManager 类。tuple 是用来存储单条记录的类。ConditionNode 类是存储条件语句的类。recordManager 类是 Record Manager 的核心，用于管理表。

## tuple 类

功能描述：用于存储Table中的一条记录

实现原理：用一个String的Vector以字符串格式一个一个地存储每个属性对应的值。

其内部成员为：

```
public class tuple {
    public Vector<String> units;
}
```

## conditionNode 类

整体功能描述：以二叉树结构记录并计算sql语句中的where条件语句。计算时输入一条tuple,返回true表示符合条件,false表示不符合条件。

支持以下基础功能：

条件语句中指定属性与常数比较

用 AND 连接条件语句。  
支持以下扩展功能：

用 OR 连接语句。如：

```
select * from student2 where score>90 and id<=1080100003 or name='嬴政';
```

用括号指定优先级。如：

```
select * from student2 where score>90 and (id<=1080100003 or name='嬴政');
```

同一个表中两个不同属性之间的数据进行比较。如：

```
select * from student2 where ChineseScore>MathScore;
```

类内各部分的功能描述及实现原理如下表所示：

	名称	功能描述	实现原理
成员变量	String tablename	在非叶节点中为空。在叶节点中表示运算符左侧涉及的属性属于哪个表。	
	String attriName	在非叶节点中为空。在叶节点中表示运算符左侧涉及的属性的名字。	
	String tablename2	在非叶节点中为空。在叶节点中表示运算符右侧涉及的属性属于哪个表。如果运算符右侧为常数时该值为空。	
	String attriName2	在非叶节点中为空。在叶节点中表示运算符右侧涉及的属性的名字。如果运算符右侧为常数时该值为空。	
	String conjunction	在叶节点中为空。表示多个条件之间用and 还是 or 来进行连接。	
	Comparison op	在非叶节点为空。在叶节点中表示运算符。	
	String value	在非叶节点中为空。在叶节点中如果运算符右侧为常数时，记录在该值中。如果运算符右侧涉及的是属性数据而不是常数时该值为空。	
	conditionNode left	左儿子	
	conditionNode right	右儿子	



	boolean constantFlag	如果是和常数比较则为 true，如果是和另一个 attribute 比较则置 false	
外部接口	conditionNode(String attriName, Comparison op, String value,boolean constantFlag)	用于构造属性与常量比较的叶节点	根据输入进行赋值
	conditionNode(String conjunction)	用于构造属性与属性之间比较的叶节点	根据输入进行赋值
	conditionNode linkChildNode( conditionNode l, conditionNode r) {	用于把一个父节点链接到两个叶节点	根据输入进行赋值
	boolean calc(String tablename, tuple T)	输入一个 Tuple，判断这条记录是否满足这个条件。	从根节点开始，递归地进行计算。具体算法详见后文。

## RecordManager 类

功能概述：

基于**Free List**，对一个表中的所有记录按块进行存取。每个表存储在一个独立的文件中。

支持以下基础功能：

select, delete, drop, create table, insert 的基本功能。

支持以下扩展功能：

**排序功能: Order by 如：**

```
select * from student2 order by score;
```

**连接功能: Join。如：**

```
select * from student2 join student_department where student2.id=student_department.id;
```

**Projection 功能。如：**

```
select name from student2;
```

名称	功能描述	实现原理
boolean createTable(String tableName)	给定表名，创建表文件，初始化表头。	调用 FileManager 创建文件。调用 bufferManager 在表头指针处写 0。
boolean dropTable(String tableName)	给定表名，删除表文件。	调用FileManager 删除文件。调用bufferManager 把该表的脏数据清除掉。

int insert(String tablename, tuple Tuple)	给定表名，及一个tuple，插入表中，并返回所插入位置的tupleOffset。	先从表头查询 FreeList,如果有被删除后留下来的空位，则插入表头指向的空位，并让表头指向下一个空位。如果 FreeList 中没有空余，则通过 Catalog Manager 获得表当前的tuple 数量，以此计算出表末尾的块号 blockOffset 及字节偏移 byteOffset，插入对应位置。
Vector<tuple> project(Vector<tuple> res, String tablename, Vector<String> attriNames)	用于做 select 时的 projection。给定表名，及要选出来属性名称，输入 select 函数返回的结果，返回经 projection 后的结果。	根据要选出来属性名称，把 tuple 中没有被选择的属性值删去。
Vector<tuple> select(String tablename, conditionNode condition)	用于普通select 语句。输入表名和判断条件，选出符合条件的记录存储在 Vector<tuple>中并返回	对表中的有效记录进行逐个遍历，读出记录至 tuple T 并使用 condition.c alc(tuple T)去判断每条记录是否符合条件，符合则放入 Vector<tuple>中，遍历完后返回结果。
Vector<tuple> select(String tablename, conditionNode condition, String orderAttriName, boolean isInc)	用于带有排序功能 order 的select 语句。输入表名、判断条件、排序依据属性名、升序排序或降序排序，选出符合条件的记录存储在Vector<tuple>中，排序后返回	对表中的有效记录进行逐个遍历，读出记录至 tuple T 并使用 condition.c alc(tuple T)去判断每条记录是否符合条件，符合则放入 Vector<tuple> 中，遍历完后，调用vector 中的sort 函数进行排序，返回结果。
int delete(String tablename, conditionNode condition)	用于delete 语句。输入表名和判断条件，删除符合条件的记录，返回被删除的记录数	对表中的有效记录进行逐个遍历，读出记录至 tuple T 并使用 condition.c alc(tuple T)去判断每条记录是否符合条件，符合则把该记录的标记为标为已删除。
Vector<tuple> join(String tableName1, String attributeName1, String tableName2, String attributeName2)	用于 select * join from A.a=B.b 指令，指定两个表及其对应属性进行 join，返回对应结果。	用select 函数获得两个表中的所有记录，用两重循环进行两两匹配，如果指定的属性值相等则加入结果中。

	Vector<tuple> getTuple(String tablename, Vector<Integer>tupleOffsets)	给定表名和多个tupleOffsets 所构成的 vector ，返回对应多个 tuple 所构成的 vector 。如果给定 tupleOffset 处数据为空或数据已被删除，则对应结果中为null。	先根据tupleOffset 和每个记录的长度，计算出对应记录在文件对应的块号 blockoffset 以及块内字节偏移 byteoffset，从对应块中读出数据，再根据每个attribute的类型进行转换，逐个存入 tuple 中。
	tuple getTuple(String tablename, int tupleOffset)	给定表名和多个tupleOffsets 所构成的 vector ，返回对应多个 tuple 所构成的 vector 。如果给定 tupleOffset 处数据为空或数据已被删除，则对应结果中为null。	先根据tupleOffset 和每个记录的长度，计算出对应记录在文件对应的块号 blockoffset 以及块内字节偏移 byteoffset，从对应块中读出数据，再根据每个attribute的类型进行转换，逐个存入 tuple 中。
内部类	static class MyCompare implements Comparator<tuple>	用于带有排序功能 order 的select 语句，通过指定要比较属性名称来进行比较。	通过属性名称取出数值进行比较。

## File Manager

功能概述：用于创建、删除、管理文件。

	名称	功能描述	实现原理
对外接口	void creatFile(String filename)	给定文件名，创建文件	调用 File.createNewFile()
	void dropFile(String filename)	给定文件名，删除文件	调用 File.delete()
	boolean findFile(String filename)	给定文件名，判断文件是否存在	调用 File.exist 函数

## IndexManager

代码模块：

IndexManger 包中包含三个类。BPlusTree、offsetInfo 类以及 IndexManager 类。

BPlusTree 类是用来包装新建或者已存在的索引结构的 B+ 树结构类。并且为 IndexManager 提供 B+树的更删改查等维护功能。

IndexManager 类是用来给API 提供相应的索引函数接口，比如索引的建立、删除、插入等操作。

offsetInfo 是用来记录 B+树中的键值的数据结构。

下面详细介绍各个模块内的功能。

IndexManager 类

功能：作为封装好的对索引进行操作的模块，通过API 提供给Interpreter 进行调用。

类内各部分的功能描述及实现原理如下表所示：

	名称	功能描述	实现原理
方法描述	<b>boolean createIndex(index indexInfo)</b>	用于针对一个表及指定属性创建索引	返回布尔变量，表示创建是否成功
	<b>boolean dropIndex(String filename)</b>	用于针对文件名字删除指定索引	返回布尔变量，表示创建是否成功
	<b>void deleteKey(index indexInfo,String deleteKey)</b>	用于删除指定索引中的指定键值	无返回值
	<b>Integer searchEqual(index indexInfo, byte[] key)</b>	用于等值的查找，通过索引查找指定键值的位置	返回所要找的tuple 在文件中的偏移量
	<b>Vector&lt;Integer&gt; searchRange(index indexInfo,String startkey, String endkey)</b>	用于范围的查找，通过索引范围来查找在范围内的一系列键值的位置	返回所要找的一系列tuple 在文件中的偏移量数组
	<b>void insertKey(index indexInfo,String key,int blockOffset,int offset)</b>	用于将键值插入指定的索引中。	无返回值
	<b>byte[] StringInttoByte</b>	将用字符串表示的整数转	返回转换后的 Byte 型数组

	<b>e(String num)</b>	换为 Byte 型数组	
	<b>byte[] StringFloattoByte(String num)</b>	将用字符串表示的浮点数转换为Byte 型数组	返回转换后的 Byte 型数组

## BPlusTree 类

功能：作为对硬盘上的索引文件抽象成B+树结构的类供IndexManager 调用，同时可以对物理存储上的索引文件进行维护：

	名称	功能描述	实现原理
成员变量	<b>final int <del>POWER</del></b> <b>H= 4</b>	常量，用于记录指针的长度，由于使用的是整形所以长度是 4	
	<b>final double <del>BOKE</del></b> <b>= 4096.0</b>	常量，用于记录一个树节点的大小，默认是 4KB	
	<b>int</b> <b>MIN_CHILDREN_FOR_INTERNAL;</b>  <b>int</b> <b>MAX_CHILDREN_FOR_INTERNAL;</b>  <b>int MIN_FOR_LEAF;</b> <b>int MAX_FOR_LEAF;</b>	常量，用于记录中间节点的最小最大路标个数，以及叶子节点的最小最大索引个数。	
	<b>String filename;</b>	索引文件名	
	<b>Block myRootBlock;</b>	包装好的根块Block 信息体	
	<b>index myindexInfo;</b>	索引的信息体，由外部传入，可实时更新	
	<b>BPlusTree(index indexInfo)</b>	构造函数，用于从无到有地生成新的一颗B+树。	①通过FileManager 新建索引文件，并通过BufferManager 获取第一块节点大小的Block； ②实时计算树的叉数； ③初始化根节点。
	<b>BPlusTree(index indexInfo,int rootBlockNum)</b>	构造函数，用于读取已存在的索引并包装成 B+树。	①计算树的叉数； ②通过BufferManager 以及根块的位置获取根节点的
方法描述			

			BBlock ; ③初始化根节点。
	<b>void insert(byte[] originalkey,int blockOffset, int offset)</b>	插入函数，以树为单位的插入。	①判断根块的属性并用中间节点和叶子节点的构造方法进行包装； ②调动节点的插入方法进行插入操作； ③判断节点的返回值，如果有返回值说明根块有更新，要更新树的rootNum变量。
	<b>offsetInfo searchKey (byte[] originalkey)</b>	等值查找函数，以树为单位的范围查找。返回一个保存返回信息的类结构。	①判断根块的属性并用中间节点和叶子节点的构造方法进行包装； ②调动节点的查找方法进行查找。
	<b>offsetInfo searchKey (byte[] originalkey,byte[] endkey)</b>	范围查找函数，以树为单位的范围查找。返回一个保存返回信息的类结构。	①判断根块的属性并用中间节点和叶子节点的构造方法进行包装； ②调动节点的查找方法进行查找。
	<b>void delete(byte[] originalkey)</b>	删除函数，以树为单位的删除。删除一个树中与给定值相等的索引值。	①判断根块的属性并用中间节点和叶子节点的构造方法进行包装； ②调动节点的删除方法进行插入操作； ③判断节点的返回值，如果有返回值说明根块有更新，要更新树的rootNum变量。

其中BPlusTree 类还包含了一个Node 的节点虚类，它有InternalNode 和LeafNode 两个子类。它们的功能为对节点为单位的更删改查以及节点调整中的删除、合并、增添的操作。具体的函数如下所示：

	名称	功能描述	实现原理
抽象类 Node			
成员变量	<b>Block block;</b>	存放该节点对应磁盘中的Block 块。	

方法描述	<b>Node createNode(Block blk)</b>	初始化成员变量block。	直接通过参数赋值。
	<b>abstract Block insert (byte[] inserKey,int blockOffset, int offset)</b>	抽象方法：插入函数	
	<b>abstract Block delete (byte[] deleteKey)</b>	抽象方法：删除函数	
	<b>abstract offsetInfo searchKey(byte[] Key)</b>	抽象方法：等值查找函数	
	<b>abstract offsetInfo searchKey(byte[] skey, byte[] ekey)</b>	抽象方法：范围查找函数	
	<b>int compareTo(byte[] buffer1,byte[] buffer2)</b>	比较函数，定义统一的根据Byte 型数组互相之间的比较，能够适配所有的树内索引排序。	和字典排序类似，只需要从开始比较数组中两个数的大小关系，并在不等的时候返回大小关系。
<b>class InternalNode extends Node 中间节点类</b>			
方法描述	<b>InternalNode(Block blk)</b>	构造函数，申请新建一个新的块。	对新块进行信息头的标记。具体可以看后面B+树的数据结构。
	<b>InternalNode(Block blk,boolean t)</b>	构造函数，已有的块进行包装。	对参数block 赋值。
	<b>Block insert(byte[] insertKey,int blockOffset, int offset)</b>	索引插入函数，对指定的键值进行插入。	查找键值所在的路标，然后递归到下一个子节点调用它的插入函数。如果根块有更新，会返回一个指针，如果没有返回 null。
	<b>Block branchInsert(byte[] branchKey,Node leftChild,Node rightChild)</b>	路标插入函数，供子节点分裂后调用插入路标。插入过程中有时需要对中间节点进行分支，branchKey 为要新插入的路标，leftChild 为新路标的左子节点（也就是已经存在的节点），rightChild 为新路标的右子节点	①如果要分裂的情况下要申请新的节点空间，并分配好路标，同时继续往上调用 branchInsert 进行插入更新； ②对该节点的插入位置键值、插入左右子节点、节点总路标数更新。



	<b>offsetInfo searchKey</b>	以中间节点为单位的等值	遍历整个节点，找到该键
	<b>(byte[] key)</b>	查找	值后递归调用子节点的 <b>searchKey</b> 方法。
	<b>offsetInfo searchKey (byte[] skey,byte[] ek ey)</b>	以中间节点为单位的范围查找	只是提供给范围查找调用，方法与上一个完全一样。
	<b>Block delete(byte[] deleteKey)</b>	以中间节点为单位的删除	与查找类似，递归调用子节点删除方法。
	<b>Block union(byte[] unionKey,Block afterBlock)</b>	删除过程中产生的节点合并，this 块和 after 块以及它们之间的unionKey	合并两个节点，并找到合并的路标，调用父节点进行块删除。
	<b>byte[] rearrangeAfter (Block siblingBlock,byte[] InternalKey)</b>	删除过程中产生的兄弟块内容重排，this 块和after 块以及它们之间的internal Key,返回的 changeKey 是为了更新父块中它们两指针中间的键值。这是兄弟节点在其后的方法。	①找到兄弟节点要转移的一条指针内容； ② 将 internalKey 和兄弟块的第一条指针复制到this 块的尾部，路标数加1； ③兄弟块的路标数减 1，获取兄弟块的第一条键值作为更新父块的键值，再将兄弟块后面的信息调整
方法描述	<b>byte[] rearrangeBefore(Block siblingBlock,byte[] internalKey)</b>	同上，是兄弟节点在其前的方法。	同上。
	<b>void exchange(byte[] changeKey,int posBlock Num)</b>	修改 posBlockNum 标号后面的路标。	直接修改对应位置的键值。
	<b>Block delete(Block blk)</b>	在中间节点中删除一个子块信息（以及它前面的那条路标）	删除后查看是否需要调整中间节点进行合并，如果需要则向父亲查找前后兄弟节点进行rearrange；不需要合并则直接结束。
<b>class LeafNode extends Node 叶子节点类</b>			
方	<b>LeafNode(Block blk)</b>	构造函数，申请新建一个新的块。	对新块进行信息头的标记。具体可以看后面B+树的数据结构。
	<b>LeafNode(Block blk,boolean t)</b>	构造函数，已有的块进行包装。	对参数block 赋值。



法 描 述	<b>Block insert(byte[] insertKey,int blockOffset, int offset)</b>	索引插入函数，对指定的键值进行插入。	查找键值所在的路标并插入，同时判断是否需要分裂叶子节点。如果需要更新，会返回一个根块指针，如果没有返回 null。
	<b>offsetInfo searchKey (byte[] key)</b>	以叶子节点为单位的等值查找。	遍历整个节点，找到该键值后返回该记录的偏移
			量，如果需要合并或借值再分别判断完成。
	<b>offsetInfo searchKey (byte[] skey,byte[] ek ey)</b>	以叶子节点为单位的范围查找。	只是提供给范围查找调用，方法基本与上一个完全一样。找到skey后往后遍历找到最后不满足为止，返回记录偏移量的数组。
	<b>Block delete(byte[] deleteKey)</b>	以叶子节点为单位的删除	与查找类似，递归调用子节点删除方法。
	<b>Block union(byte[] unionKey,Block afterBlock)</b>	删除过程中产生的节点合并，this 块和 after 块以及它们之间的unionKey	合并两个节点，并找到合并的路标，调用父节点进行块删除。
	<b>byte[] rearrangeAfter (Block siblingBlock,byte[] InternalKey)</b>	删除过程中产生的兄弟块内容重排，this 块和after 块以及它们之间的internal Key,返回的 changeKey 是为了更新父块中它们两指针中间的键值。这是兄弟节点在其后的方法。	①找到兄弟节点要转移的一条指针内容； ② 将 internalKey 和兄弟块的第一条指针复制到this 块的尾部，路标数加1； ③兄弟块的路标数减 1，获取兄弟块的第一条键值作为更新父块的键值，再将兄弟块后面的信息调整
	<b>byte[] rearrangeBefore (Block siblingBlock, byte[] internalKey)</b>	同上，是兄弟节点在其前的方法。	同上。

## B+树结构

B+树采用实时计算的特点得出最终每个节点的最大最小路标数和最大最小索引树。其中 每个节点的结构如下所示：

长度	1	4	4	4	4	记录长度	...	4
类型	标记	个数	偏移	偏移	偏移	键值	...	偏移
Internal Node	I	路标个数	父亲节点偏移	子块偏移		路标值	...	子块偏移
Leaf Node	L	索引个数	父亲节点偏移	记录文件偏移量	记录块内偏移量	索引键值	...	尾指针（兄弟节点）

对应的每个节点都用Byte[]型数组来进行存储，长度均为Byte 型，这样可以保证适配所有类型属性的索引建立。同时使用文件内的偏移量来代表数组，这样保证每次搜寻时都需要 通过BufferManager 去Disk 里寻找新的节点块，保证了内存方面的管理。

### 3. 测试结果

本组miniSQL采用提供的相关文件与自行设计的数据进行测试，分为基础功能测试、压力测试、bonus功能测试、语义功能报错测试。

#### A. 基础功能测试

##### 1. 表的创建删除功能

对应文件的运行结果截图如下：

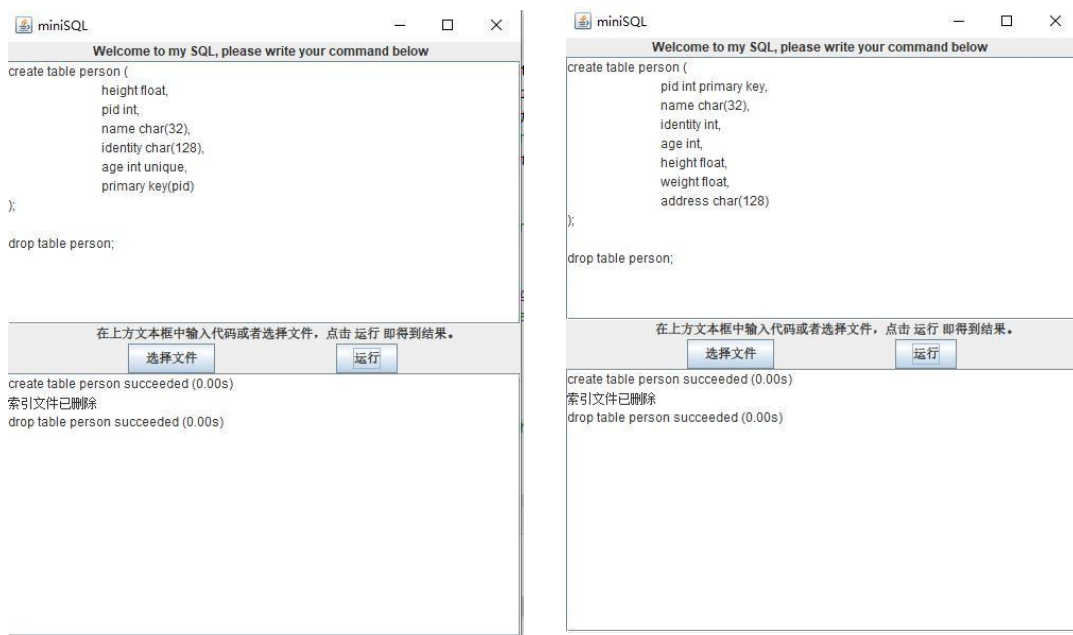


table-create-drop-0.sql

table-create-drop-1.sql



table-create-drop-2.sql

说明：表的建立与删除都是没有问题的，建立与删除时都会对primary key建立索引.根据1文件我们对Interpreter模块进行了修改，使得其可以识别primary key作为属性后缀的情况。

## 2. 元组的插入删除功能

对应文件的运行结果截图如下：

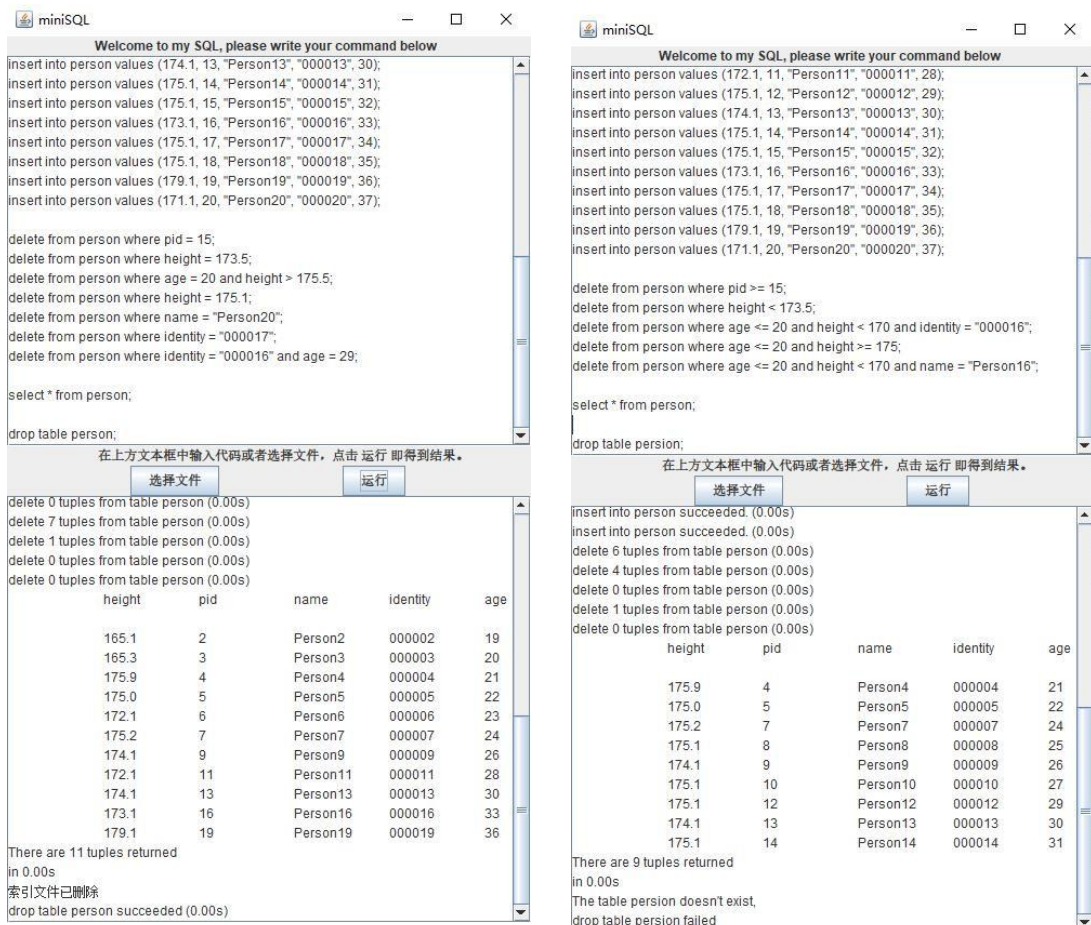


table-insert-delete-0.sql

table-insert-delete-1.sql

说明：对于table-insert-delete-0.sql，对于每一条删除语句，删除的元组数分别为1、0、0、7、1、0、0，这与删除的结果对应一致，说明插入与删除的执行结果是正确的。

对于table-insert-delete-1.sql，对于每一条删除语句，删除的元组数分别为6、4、0、1、0，删除的结果也是符合的，而且在最后删除表的时候，由于表名错误，删除失败了。

### 3. 元组的查询功能

The screenshot shows the miniSQL interface with the following SQL commands and results:

```
insert into person values (172.1, 6, 'Person6', '000006', 23);
insert into person values (175.2, 7, 'Person7', '000007', 24);
insert into person values (175.1, 8, 'Person8', '000008', 25);
insert into person values (174.1, 9, 'Person9', '000009', 26);
insert into person values (175.1, 10, 'Person10', '000010', 27);
insert into person values (172.1, 11, 'Person11', '000011', 28);
insert into person values (175.1, 12, 'Person12', '000012', 29);
insert into person values (174.1, 13, 'Person13', '000013', 30);
insert into person values (175.1, 14, 'Person14', '000014', 31);
insert into person values (175.1, 15, 'Person15', '000015', 32);
insert into person values (173.1, 16, 'Person16', '000016', 33);
insert into person values (175.1, 17, 'Person17', '000017', 34);
insert into person values (175.1, 18, 'Person18', '000018', 35);
insert into person values (179.1, 19, 'Person19', '000019', 36);
insert into person values (171.1, 20, 'Person20', '000020', 37);

select * from person where pid = 15;
select * from person where height = 173.5;
select * from person where age = 20 and height > 175.5;
select * from person where height = 175.1;
```

Results:

- select \* from person where pid = 15;: 1 tuple returned
- select \* from person where height = 173.5;: 0 tuples returned
- select \* from person where age = 20 and height > 175.5;: 0 tuples returned
- select \* from person where height = 175.1;: 8 tuples returned

height	pid	name	identity	age
175.1	15	Person15	000015	32

索引文件已删除  
drop table person succeeded (0.00s)

table-select-0.sql

The screenshot shows the miniSQL interface with the following SQL commands and results:

```
insert into person values (175.2, 7, 'Person7', '000007', 24);
insert into person values (175.1, 8, 'Person8', '000008', 25);
insert into person values (174.1, 9, 'Person9', '000009', 26);
insert into person values (175.1, 10, 'Person10', '000010', 27);
insert into person values (172.1, 11, 'Person11', '000011', 28);
insert into person values (175.1, 12, 'Person12', '000012', 29);
insert into person values (174.1, 13, 'Person13', '000013', 30);
insert into person values (175.1, 14, 'Person14', '000014', 31);
insert into person values (175.1, 15, 'Person15', '000015', 32);
insert into person values (173.1, 16, 'Person16', '000016', 33);
insert into person values (175.1, 17, 'Person17', '000017', 34);
insert into person values (175.1, 18, 'Person18', '000018', 35);
insert into person values (179.1, 19, 'Person19', '000019', 36);
insert into person values (171.1, 20, 'Person20', '000020', 37);

select * from person where pid >= 15;
select * from person where height < 173.5;
select * from person where age <= 20 and height < 170;
```

Results:

- select \* from person where pid >= 15;: 6 tuples returned
- select \* from person where height < 173.5;: 4 tuples returned
- select \* from person where age <= 20 and height < 170;: 2 tuples returned

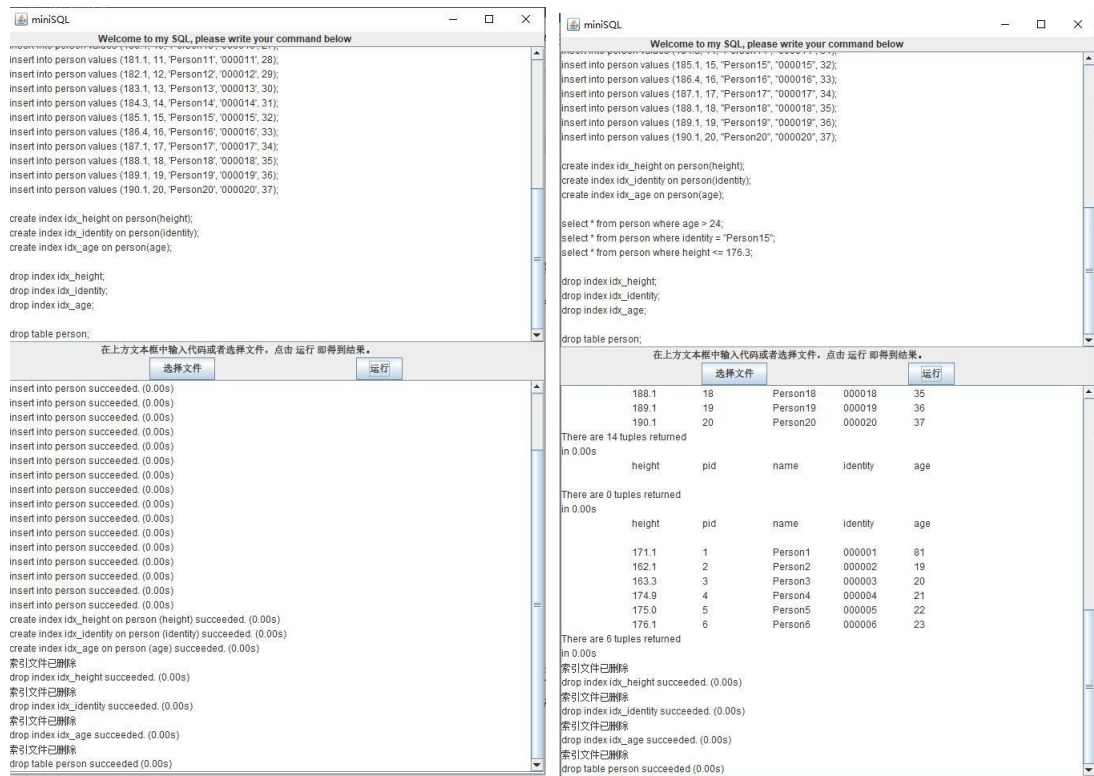
height	pid	name	identity	age
175.1	15	Person15	000015	32
173.1	16	Person16	000016	33
175.1	17	Person17	000017	34
175.1	18	Person18	000018	35
179.1	19	Person19	000019	36
171.1	20	Person20	000020	37

索引文件已删除

table-select-1.sql

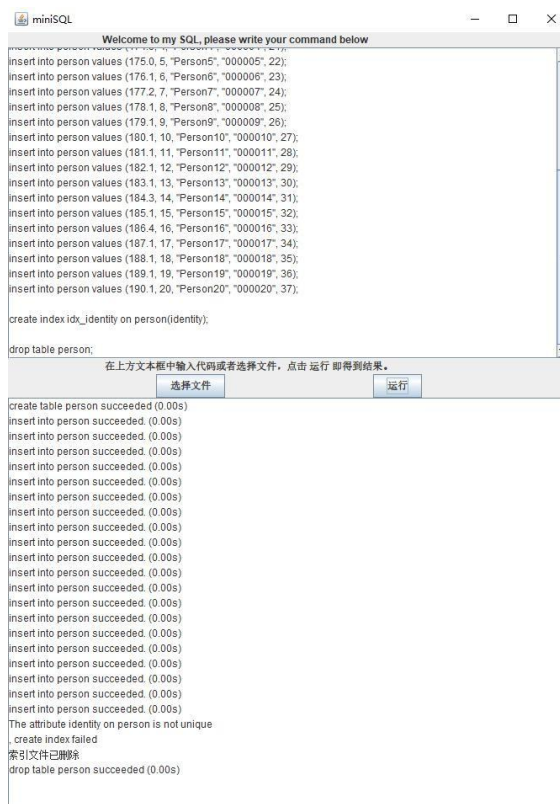
说明：根据上面两个文件，我们的查询功能已经实现了where中判断大小和相等关系的情况，而在之前的情况中，我们也已经验证了没有where的查询情况。结果均为正确的。并且会返回查询时间与结果元组数。

## 4. 索引的创建删除功能



index-create-drop-0.sql

index-create-drop-1.sql



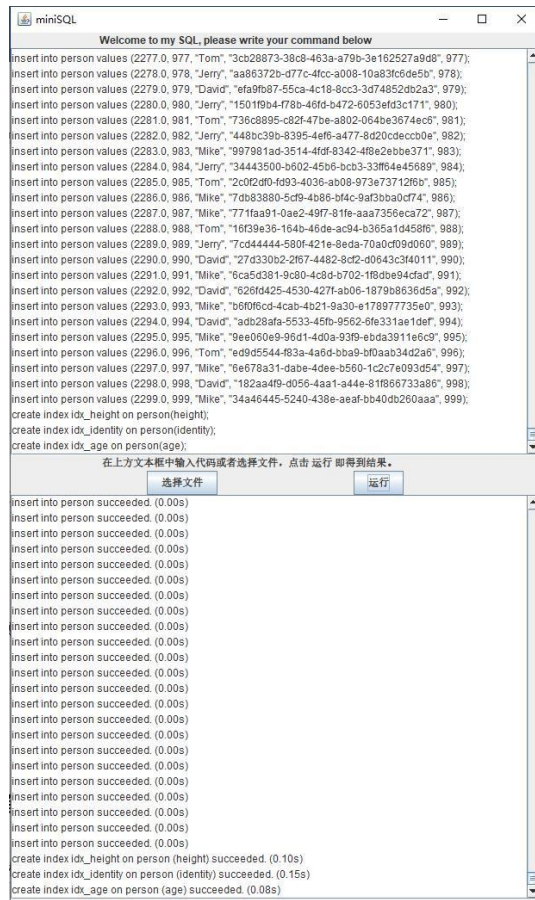
index-create-drop-2.sql

说明：第一个文件的结果说明索引的创建与删除均是没有问题的，第二个文件的结果说明通过创建所以可以使查询操作速度更快，第三个文件则是报错信息验证，未带有unique关键字的属性不能构建索引。



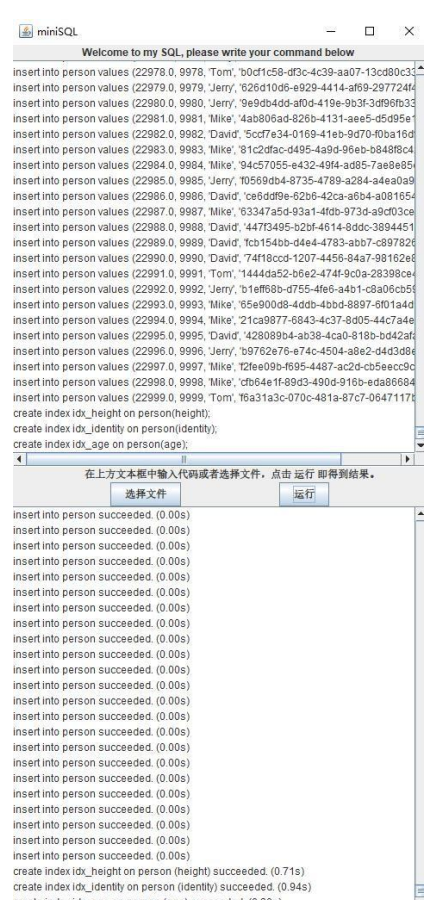
## B. 压力测试

采用提供的1000、1w、10w条的插入语句来对minisql程序的稳定性即效率作检验。



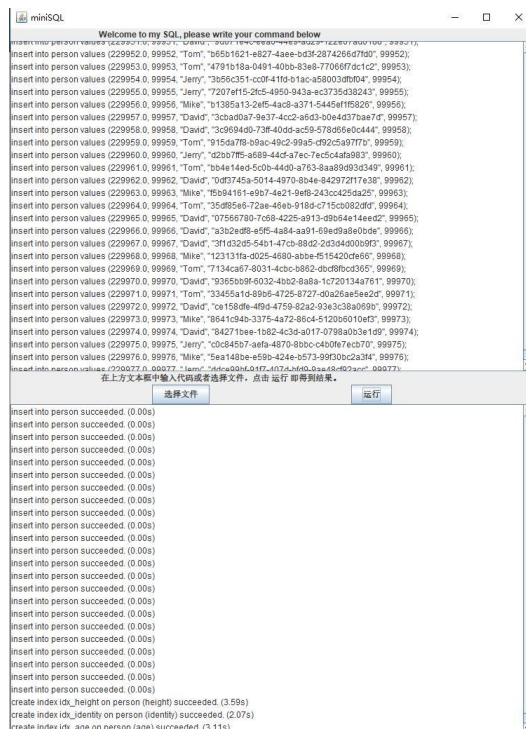
The screenshot shows the minisql application window. The top pane contains a list of 1000 INSERT statements, each with a unique ID and a name. The bottom pane shows the results of these statements, all of which were executed successfully. The results are displayed in a list format, with each statement's ID and name followed by a success message and a timestamp.

test-1000.sql



The screenshot shows the minisql application window. The top pane contains a list of 10,000 INSERT statements, each with a unique ID and a name. The bottom pane shows the results of these statements, all of which were executed successfully. The results are displayed in a list format, with each statement's ID and name followed by a success message and a timestamp.

test-1w.sql



The screenshot shows the minisql application window. The top pane contains a list of 100,000 INSERT statements, each with a unique ID and a name. The bottom pane shows the results of these statements, all of which were executed successfully. The results are displayed in a list format, with each statement's ID and name followed by a success message and a timestamp.

test-10w.sql

miniSQL				
Welcome to my SQL, please write your command below				
select * from person;				
在上方文本框中输入代码或者选择文件，点击 运行 即得到结果。				
选择文件		运行		
22951.0	9951	Mike	60120a9f-90ef-44e3-9844-8d11d91f5b0b	9951
22952.0	9952	David	36a71a41-e869-4251-8522-cd7cd7c2e58	9952
22953.0	9953	Mike	e5840ceb-9e45-4242-92b6-53926d0fa79	9953
22954.0	9954	David	d5d39051-b416-43bc-a2b6-451dcb0da197	9954
22955.0	9955	Tom	52664578-211f-42a2-b8fc-b57ef6ace296	9955
22956.0	9956	David	bd5c89dd-bd57-463d-88d5-6205d7eee3e1	9956
22957.0	9957	Tom	2eab3428-fc79-4b15-8206-671475829ee6	9957
22958.0	9958	Tom	2fe67c94-85a3-4880-95cb-c42ad28af81	9958
22959.0	9959	Tom	82ad2d29-ad00-4dff-8d34-5e0e4e977a27	9959
22960.0	9960	Mike	72f6e980-da50-49ec-a611-e4d5b6de63d0	9960
22961.0	9961	Tom	56736d4b-9db1-45ba-8c1e-4b256f041580	9961
22962.0	9962	David	3c9c7c47-f381-47eb-801a-0172390d935a	9962
22963.0	9963	David	6bfe35ad-9cfd-469a-a0fc-9cd2fd45590b	9963
22964.0	9964	David	1be9f1ef-78d2-430f-8c08-1aee0aafef0a	9964
22965.0	9965	Jerry	0e108ca2-6ff7-46ad-b9e3-d26898631017	9965
22966.0	9966	Jerry	5522f83d-143a-4978-932d-7a0ce0124811	9966
22967.0	9967	Mike	81ba0b0f-6864-49b1-9b40-d95a4267214	9967
22968.0	9968	Jerry	6273ee45-c5d3-4966-bf97-e93a2f8f93b6	9968
22969.0	9969	Mike	01080386-e8ba-4567-be3d-1cc36a8c865e	9969
22970.0	9970	Jerry	fdfd186f-a1cf-4408-88d4-a762e725fa77	9970
22971.0	9971	Mike	b6571c63-af13-460f-9f45-677f8afb1a42	9971
22972.0	9972	David	0fc3978f-7862-416e-9061-9f5a452628c	9972
22973.0	9973	Tom	80323296-f72a-4c2f-bd6e-aab17285862e	9973
22974.0	9974	Mike	a65b5ee6-3b32-4234-87b5-f9567856c798	9974
22975.0	9975	Tom	af7372a0-ad00-4fa6-b751-8271dbd1e45f	9975
22976.0	9976	Mike	e702e2d7-22ac-4f2e-81a3-e0757cdce555	9976
22977.0	9977	Jerry	9711b2a6-dbf6-44cf-9f37-4540d3c51347	9977
22978.0	9978	Tom	b0cf1c58-df3c-4c39-aa07-13cd80c335d4	9978
22979.0	9979	Jerry	626d10d6-e929-4414-af69-29772442700	9979
22980.0	9980	Jerry	9e9db4d6-af0d-419e-9b3f-3df96fb3385d	9980
22981.0	9981	Mike	4ab806ad-826b-4131-ae5e-d5d95e1f813c	9981
22982.0	9982	David	5ccf7e34-0169-41eb-9d70-f0ba16dfc974	9982
22983.0	9983	Mike	81c2dfac-d495-4a9d-96eb-b848f8c420d7	9983
22984.0	9984	Mike	94c57055-e432-49f4-ad85-7ae8e85e3a34	9984
22985.0	9985	Jerry	f0569db4-8735-4789-a284-a4ea0a999b89	9985
22986.0	9986	David	ce6ddff9e-62b6-42ca-a6b4-a081654e20cc	9986
22987.0	9987	Mike	63347a5d-93a1-4fdb-973d-a9cd03ce8bc8	9987
22988.0	9988	David	447f3495-b2bf-4614-8ddc-389445193574	9988
22989.0	9989	David	fcfb154bb-d4e4-4783-abb7-c89782618ec3	9989
22990.0	9990	David	74f18ccd-1207-4456-84a7-98162e8fce65	9990
22991.0	9991	Tom	1444da52-b6e2-474f-9c0a-28308ce43f88	9991
22992.0	9992	Jerry	b1eff88b-d755-4fe6-a4b1-c8a06cb59e88	9992
22993.0	9993	Mike	65e900d8-4ddb-4bbd-8897-f6f1a4d929f2	9993
22994.0	9994	Mike	21ca9877-6843-4c37-8d05-44c7a4e09dd2	9994
22995.0	9995	David	428089b4-ab38-4ca0-818b-bd42afa0f1b9	9995
22996.0	9996	Jerry	b9762e76-e74c-4504-a8e2-d4d3d8e5b226	9996
22997.0	9997	Mike	f2fee09b-f695-4487-ac2d-cb5eecc9c713	9997
22998.0	9998	Mike	cfb64e1f-89d3-490d-916b-eda86684feb6	9998
22999.0	9999	Tom	f6a31a3c-070c-481a-87c7-0647117ba229	9999
There are 10000 tuples returned in 0.47s				

## 关于1w条数据的查询操作

miniSQL	
Welcome to my SQL, please write your command below	
drop index idx_index;	
drop index idx_height;	
drop index idx_identity;	
drop table person;	
在上方文本框中输入代码或者选择文件，点击 运行 即得到结果。	
选择文件 运行	
The index idx_index doesn't exist, drop index idx_index failed 索引文件已删除 drop index idx_height succeeded. (0.01s) 索引文件已删除 drop index idx_identity succeeded. (0.00s) 索引文件已删除 索引文件已删除 drop table person succeeded (0.01s)	

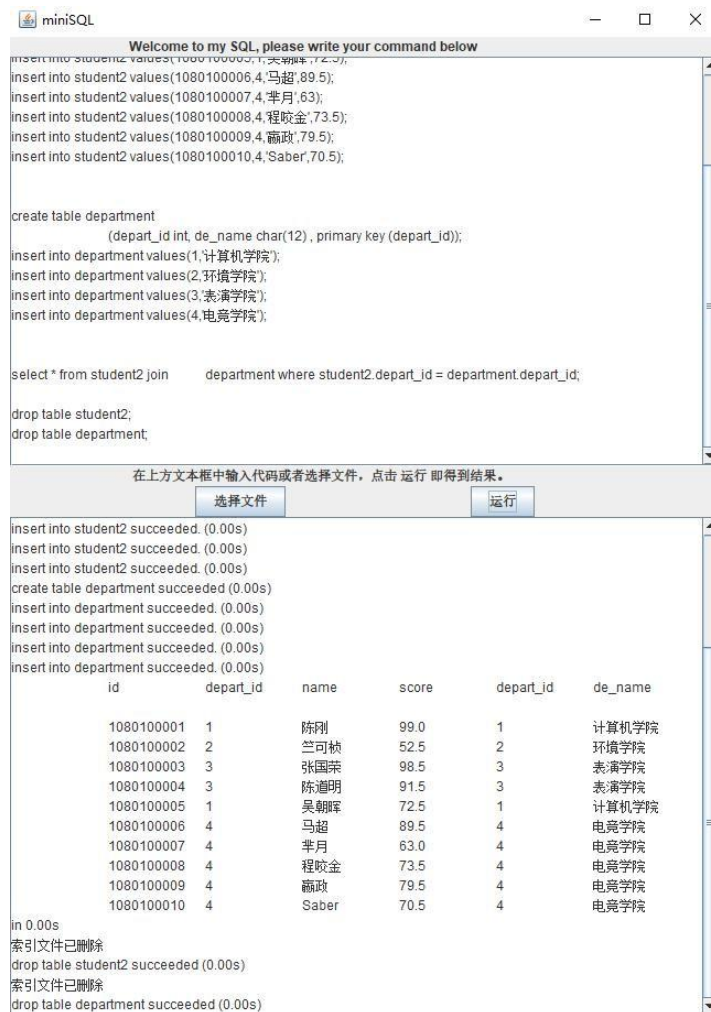
## 关于10w条数据的索引删除与表删除操作

由上述的压力测试可以分析得到，我们的程序在面对大数据量的时候运行结果是没有问题的，1000条插入操作在1s内可以完成，而1w条插入操作可以在数十秒内完成，10w条插入操作可以在数分钟内完成。效率也还是不错的。其中我们对1w条插入数据的表进行查询操作，结果也是正确的，没有影响。

### C. bonus功能测试

我们在基础功能之上实现了GUI(这个通过截图就可以看出来)，支持中文(下面的例子也将有中文的例子)和查询的额外操作，下面主要是针对查询的额外操作进行测试。

#### 1. 连接操作 (join)



The screenshot shows the miniSQL GUI with the following SQL commands in the input area:

```
insert into student2 values(1080100000,1,'文朝晖',72.5);
insert into student2 values(1080100006,4,'马超',89.5);
insert into student2 values(1080100007,4,'非月',63);
insert into student2 values(1080100008,4,'程咬金',73.5);
insert into student2 values(1080100009,4,'霸政',79.5);
insert into student2 values(1080100010,4,'Saber',70.5);

create table department
(
depart_id int, de_name char(12), primary key (depart_id));
insert into department values(1,'计算机学院');
insert into department values(2,'环境学院');
insert into department values(3,'表演学院');
insert into department values(4,'电竞学院');

select * from student2 join department where student2.depart_id = department.depart_id;

drop table student2;
drop table department;
```

The output area shows the execution results:

```
insert into student2 succeeded. (0.00s)
insert into student2 succeeded. (0.00s)
insert into student2 succeeded. (0.00s)
create table department succeeded. (0.00s)
insert into department succeeded. (0.00s)
insert into department succeeded. (0.00s)
insert into department succeeded. (0.00s)
insert into department succeeded. (0.00s)
insert into department succeeded. (0.00s)

  id      depart_id  name      score      depart_id  de_name
-----
1080100001 1      陈刚      99.0      1      计算机学院
1080100002 2      兰可桢    52.5      2      环境学院
1080100003 3      张国荣    98.5      3      表演学院
1080100004 3      陈道明    91.5      3      表演学院
1080100005 1      吴朝晖    72.5      1      计算机学院
1080100006 4      马超      89.5      4      电竞学院
1080100007 4      非月      63.0      4      电竞学院
1080100008 4      程咬金    73.5      4      电竞学院
1080100009 4      霸政      79.5      4      电竞学院
1080100010 4      Saber     70.5      4      电竞学院

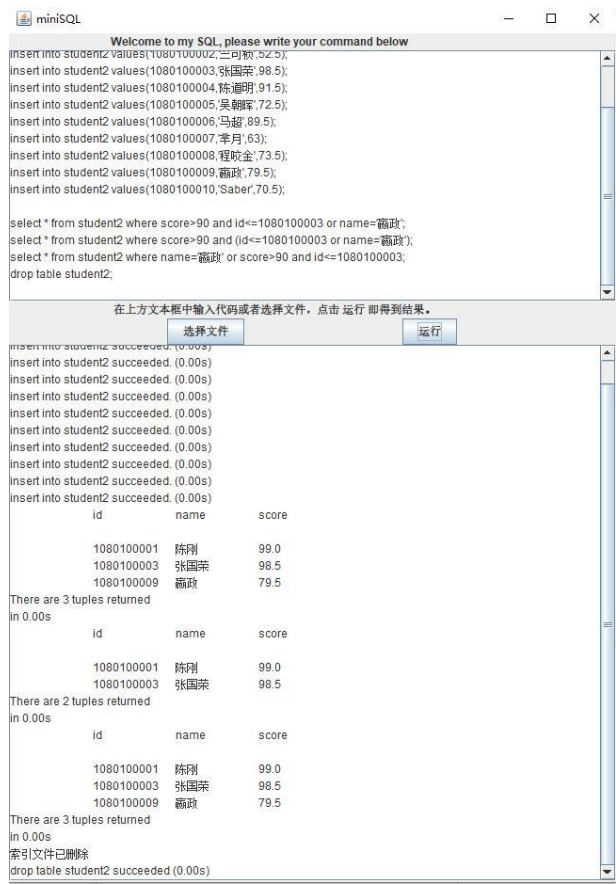
in 0.00s
索引文件已删除
drop table student2 succeeded (0.00s)
索引文件已删除
drop table department succeeded (0.00s)
```

test-join.sql

说明：创建了student2表和department表，在查询中询问了自然连接。得到的结果也正好是自然连接的结果。输出结果是按照连接的表的顺序来的，前面的表的属性在前，后面的表的属性在后。



## 2. 或操作 (or)



The screenshot shows the miniSQL application window. The top section contains a list of SQL commands: eight INSERT statements for the 'student2' table, followed by three SELECT statements using the OR operator, and a DROP TABLE statement. The bottom section shows the execution results for each command, including success messages and query results for the SELECT statements. The SELECT statements are: 1. 'select \* from student2 where score>90 and id<=1080100003 or name=薛政'; 2. 'select \* from student2 where score>90 and (id<=1080100003 or name=薛政)'; 3. 'select \* from student2 where name=薛政 or score>90 and id<=1080100003;'. The results show that the first and third queries return 3 tuples, while the second query returns 2 tuples.

```
insert into student2 values(1080100002,'王可欣',52.5);
insert into student2 values(1080100003,'张国荣',98.5);
insert into student2 values(1080100004,'陈道明',91.5);
insert into student2 values(1080100005,'吴刚',72.5);
insert into student2 values(1080100006,'马超',89.5);
insert into student2 values(1080100007,'李月',63);
insert into student2 values(1080100008,'程咬金',73.5);
insert into student2 values(1080100009,'薛政',79.5);
insert into student2 values(1080100010,'Saber',70.5);

select * from student2 where score>90 and id<=1080100003 or name=薛政;
select * from student2 where score>90 and (id<=1080100003 or name=薛政);
select * from student2 where name=薛政 or score>90 and id<=1080100003;
drop table student2;
```

在上方文本框中输入代码或者选择文件，点击 运行 即得到结果。

选择文件 运行

```
insert into student2 succeeded (0.00s)
insert into student2 succeeded (0.00s)
insert into student2 succeeded (0.00s)
insert into student2 succeeded (0.00s)
insert into student2 succeeded (0.00s)
insert into student2 succeeded (0.00s)
insert into student2 succeeded (0.00s)
insert into student2 succeeded (0.00s)
insert into student2 succeeded (0.00s)
insert into student2 succeeded (0.00s)

id      name      score
1080100001 陈刚      99.0
1080100003 张国荣    98.5
1080100009 薛政      79.5

There are 3 tuples returned
in 0.00s

id      name      score
1080100001 陈刚      99.0
1080100003 张国荣    98.5

There are 2 tuples returned
in 0.00s

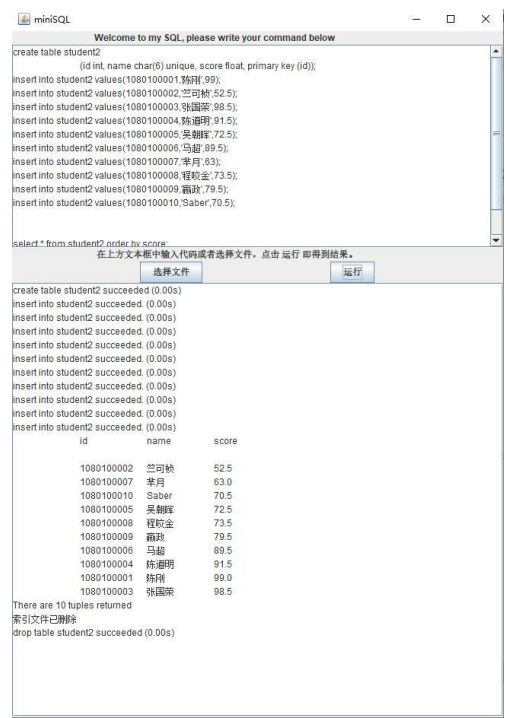
id      name      score
1080100001 陈刚      99.0
1080100003 张国荣    98.5
1080100009 薛政      79.5

There are 3 tuples returned
in 0.00s
索引文件已删除
drop table student2 succeeded (0.00s)
```

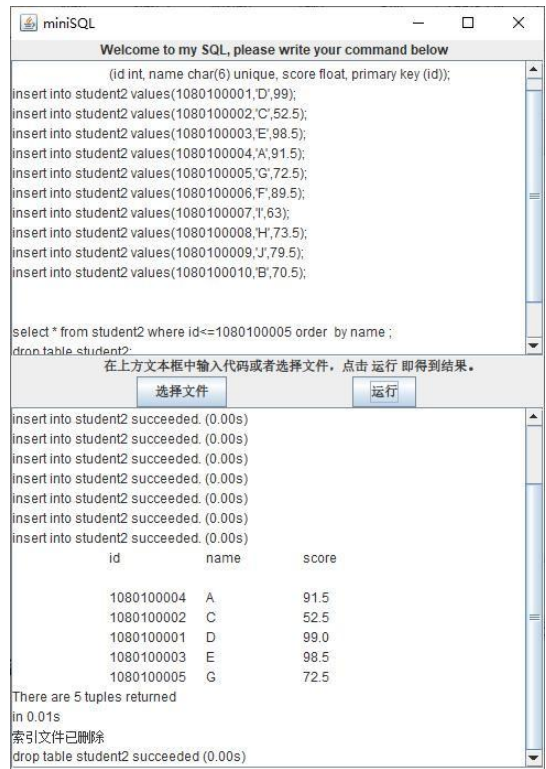
test-or.sql

说明：or即逻辑或，添加或运算后对应查询条件就不是简单的几个条件的重叠了，涉及到了逻辑表达式的优先级，and优先级比or高，此外还有括号()，上面的例子就检验了or的功能，由于and优先级高于or，因此第一个和第三个查询结果相同；第二个例子添加了括号，改变了对应逻辑，查询结果也变为了2个。检验了or的正确性。

3. 排序操作 (order by)



test-order-by.sql



test-order-by-1.sql

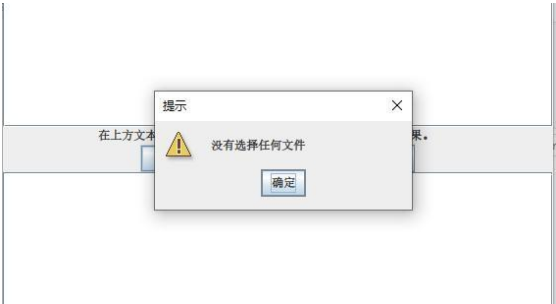
说明：添加了排序操作order by，可以针对某一个属性排序，以上即为排序结果。若需要对字符串进行排序，则是按照字典序。默认升序。



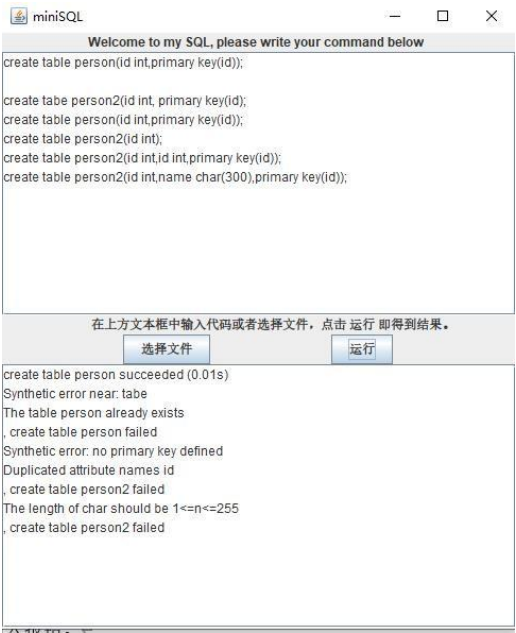
说明：该测试文件中的一条测试语句包含了投影、and、or、排序这4种操作，结果也是正确的输出出来了，可见这些新增的功能相互兼容，没有问题。

D. 语义错误测试

- 1. GUI相关报错。在选择文件过程中，若未选择文件，将会报错。

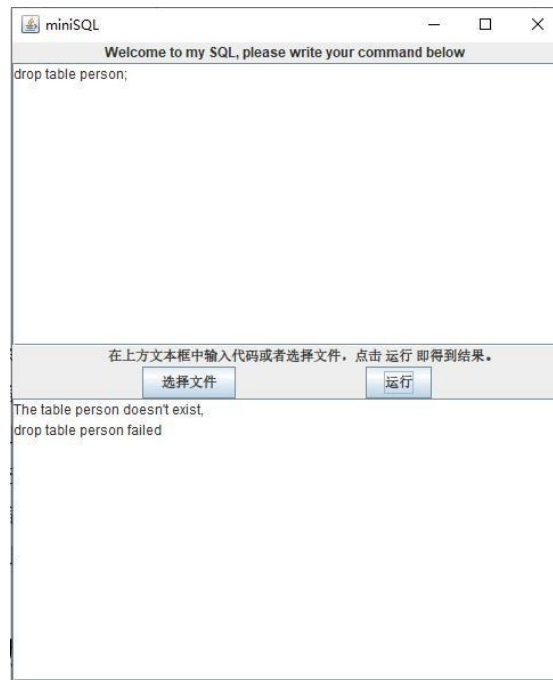


- 2. 表创建报错。有5种情况，包括未能识别的符号，table name已存在，primary key不存在，重复的attribute属性值，以及char(n)不满足 $1 \leq n \leq 255$ 。对应于下图的5条语句。



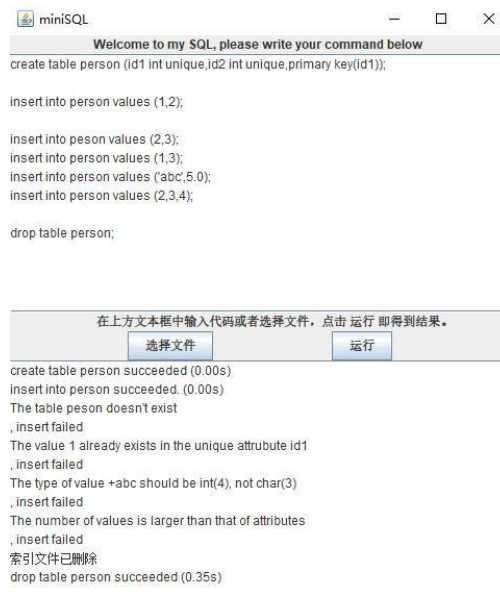
create-table-error.sql

3. 表删除报错。原因有1种，即table不存在。对应下图。



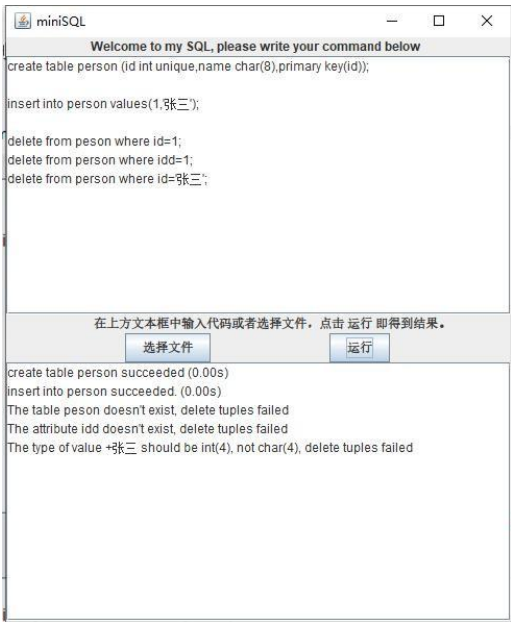
drop-table-error.sql

4. 元组插入报错。原因有4种，包括table不存在、unique属性值重复、tuple对应属性不对、tuple对应属性个数不对。对应下图：



insert-error.sql

5. 元组删除报错。原因有2种，即table不存在和where条件有误，而where条件有误有3种原因，属性名不存在，value格式错误。对应结果如下图的三条delete语句：



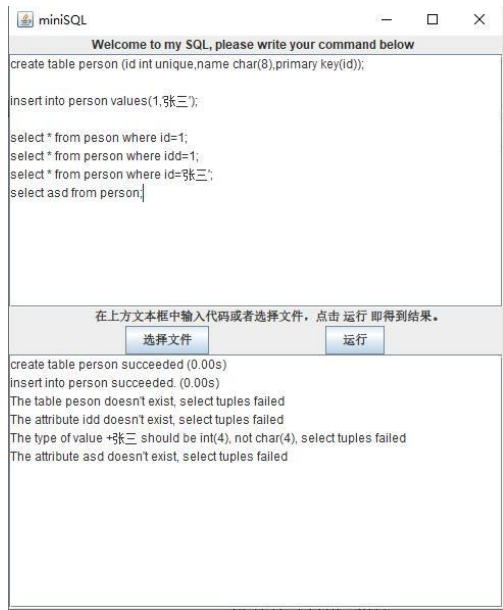
```
miniSQL
Welcome to my SQL, please write your command below
create table person (id int unique,name char(8),primary key(id));
insert into person values(1,张三);
delete from peson where id=1;
delete from person where idd=1;
delete from person where id=张三;
```

在上方文本框中输入代码或者选择文件，点击 运行 即得到结果。

create table person succeeded (0.00s)  
insert into person succeeded. (0.00s)  
The table peson doesn't exist, delete tuples failed  
The attribute idd doesn't exist, delete tuples failed  
The type of value +张三 should be int(4), not char(4), delete tuples failed

delete-error.sql

6. 元组查询报错。原因有3种，即table不存在和where条件有误（同delete）以及投影对属性不存在。对应报错结果如下：



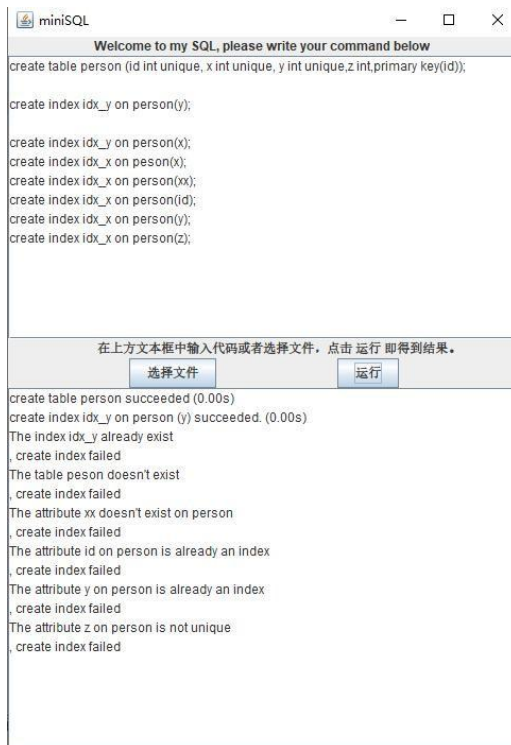
```
miniSQL
Welcome to my SQL, please write your command below
create table person (id int unique,name char(8),primary key(id));
insert into person values(1,张三);
select * from peson where id=1;
select * from person where idd=1;
select * from person where id=张三;
select asd from person;
```

在上方文本框中输入代码或者选择文件，点击 运行 即得到结果。

create table person succeeded (0.00s)  
insert into person succeeded. (0.00s)  
The table peson doesn't exist, select tuples failed  
The attribute idd doesn't exist, select tuples failed  
The type of value +张三 should be int(4), not char(4), select tuples failed  
The attribute asd doesn't exist, select tuples failed

select-error.sql

7. 索引插入报错。索引插入报错有5种情况，分别为索引名已存在，表不存在，属性不存在，该属性已经有索引（这个又可分为该属性为primary key和该属性建立过索引了），该属性不是unique的。详细结果如下图所示：



```
miniSQL
Welcome to my SQL, please write your command below

create table person (id int unique, x int unique, y int unique, z int, primary key(id));

create index idx_y on person(y);

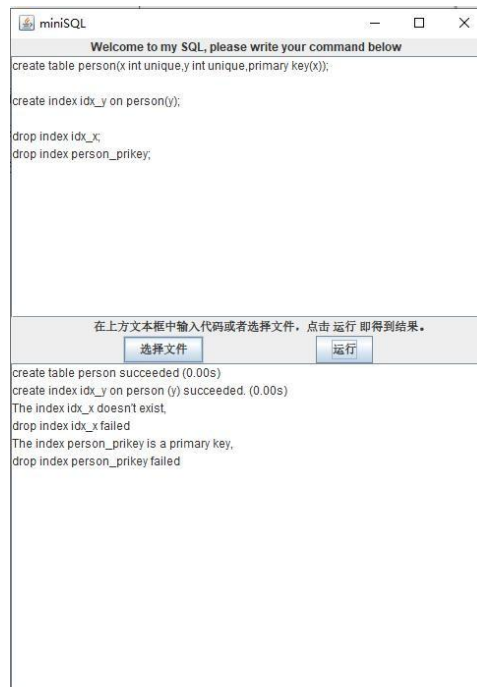
create index idx_y on person(x);
create index idx_x on peson(x);
create index idx_x on person(xx);
create index idx_x on person(id);
create index idx_x on person(y);
create index idx_x on person(z);

在上方文本框中输入代码或者选择文件，点击 运行 即得到结果。
选择文件 运行

create table person succeeded (0.00s)
create index idx_y on person (y) succeeded. (0.00s)
The index idx_y already exist
, create index failed
The table peson doesn't exist
, create index failed
The attribute xx doesn't exist on person
, create index failed
The attribute id on person is already an index
, create index failed
The attribute y on person is already an index
, create index failed
The attribute z on person is not unique
, create index failed
```

create-index-error.sql

8. 索引删除报错。报错原因有2种情况，即索引不存在和索引为主索引，对应截图如下：



```
miniSQL
Welcome to my SQL, please write your command below

create table person(x int unique,y int unique,primary key(x));

create index idx_y on person(y);

drop index idx_x;
drop index person_prikey;

在上方文本框中输入代码或者选择文件，点击 运行 即得到结果。
选择文件 运行

create table person succeeded (0.00s)
create index idx_y on person (y) succeeded. (0.00s)
The index idx_x doesn't exist
drop index idx_x failed
The index person_prikey is a primary key,
drop index person_prikey failed
```

drop-index-error.sql

# GUI设计

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。

实现方式：采用了swing库进行设计。GUI类各部的功能描述及实现原理如下表所示：

	名称	功能描述	实现原理
方法描述	<b>static int FirstWrite0, FirstWrite1;</b>	用于判断第一个文本框、第二个文本框是不是第一次打印。	
	<b>static JTextArea readArea, writeArea</b>	分别对应输入的文本框和输出的文本框。	
方法	<b>void Initialize()</b>	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
	<b>void GUImake(JFrame frame)</b>	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。



描述	<b>void FilePrintText(File file, int type)</b>	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。
	<b>void StringPrintText (String s)</b>	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。
	<b>void StringPrintAttribute (String s)</b>	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。
	<b>BufferedReader GetText (int type)</b>	与Interpreter交互的函数，在Interpreter中sql语句内容采用BufferedReader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。

## 4. 成员分工

姓名	分工
王俊	第一版本的整个minisql的搭建及后期的维护： Interpreter，API，Catalog Manager， Record Manager，Index Manager，Buffer Manager，后期添加了join，or，括号，order by 功能 总报告的撰写， 代码测试和测试文件设计
黎睿翔	GUI的设计和编写 Interpreter，Buffer Manager的重新设计 总报告的撰写， 代码测试和测试文件设计
黄磊	Catalog Manager的重新设计 join功能的完善， 总报告的撰写