MiniSQL

设计总报告

组员：

课程：数据库系统

专业：

目录

第 1 章　　MINISQL 总体框架

第 1.1 节　　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 <= 255；

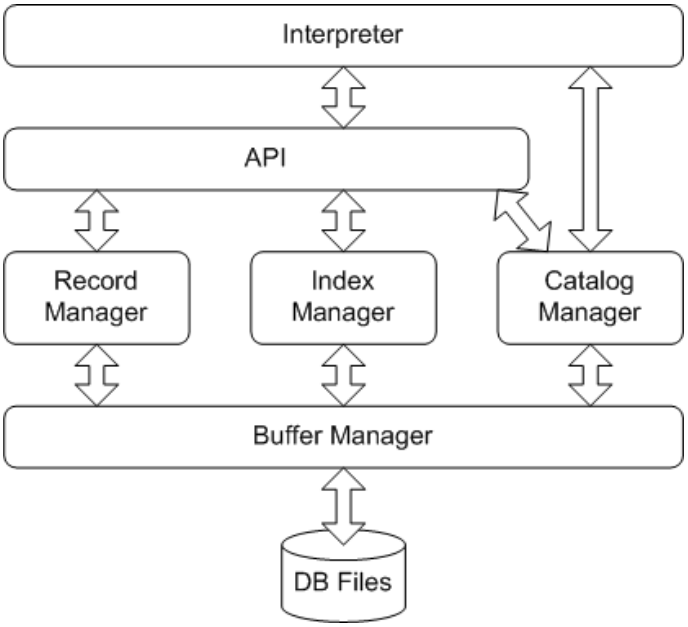
3）**表定义：**一个表最多可以定义 32 个属性，各属性可以指定是否为 UNIQUE；支持单属性的主键定义；

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。

第 1.2 节　　MINISQL 系统体系结构



第 1.3 节　　设计语言与运行环境

工具：JAVA JDK1.7　　环境：WIN7

第 2 章　　MINISQL 各模块实现功能

第 2.1 节　Interpreter 实现功能

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。

第 2.2 节　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 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。

第 2.3 节　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。

第 2.4 节　Record Manager 实现功能

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。

第 2.5 节　Index Manager 实现功能

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。

第 2.6 节　Buffer Manager 实现功能

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。

第 2.7 节　DB Files 实现功能

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。同时还有写回文件和读取文件的功能

第 3 章　内部数据形式及各模块提供的接口

第 3.1 节　内部数据形式

Interpreter 返回信息 SQL 字符串：

1. 如果输入有错，则 SQL=“99”
2. 否则 SQL=

1)创建数据库：00 数据库名；

2)删除数据库：01 数据库名；

3)创建表：10 表名 属性 1 类型 1（如果是 CHAR 则只保存数组的长度） 是否为 UNIQUE（是为 1,否为 0）属性 2 类型 2 是否为 UNIQUE…… # PRIMARYKEY 的列名（用空格隔开）；

4)删除表：11 表名；

5)创建索引：20 索引名 表名 属性列表（可能有多个，用空格隔开）；

6)删除索引：21 索引名；

7)选择语句：30 # 表名 # 选择的属性列表(如果为*,则存储的为*) # 条件 1 # 条件 2 #……（条件内部没有空格）；

8)插入记录：40 表名 # 记录(只有一条记录，按照属性顺序，用’,’隔开，中间没有空格）；

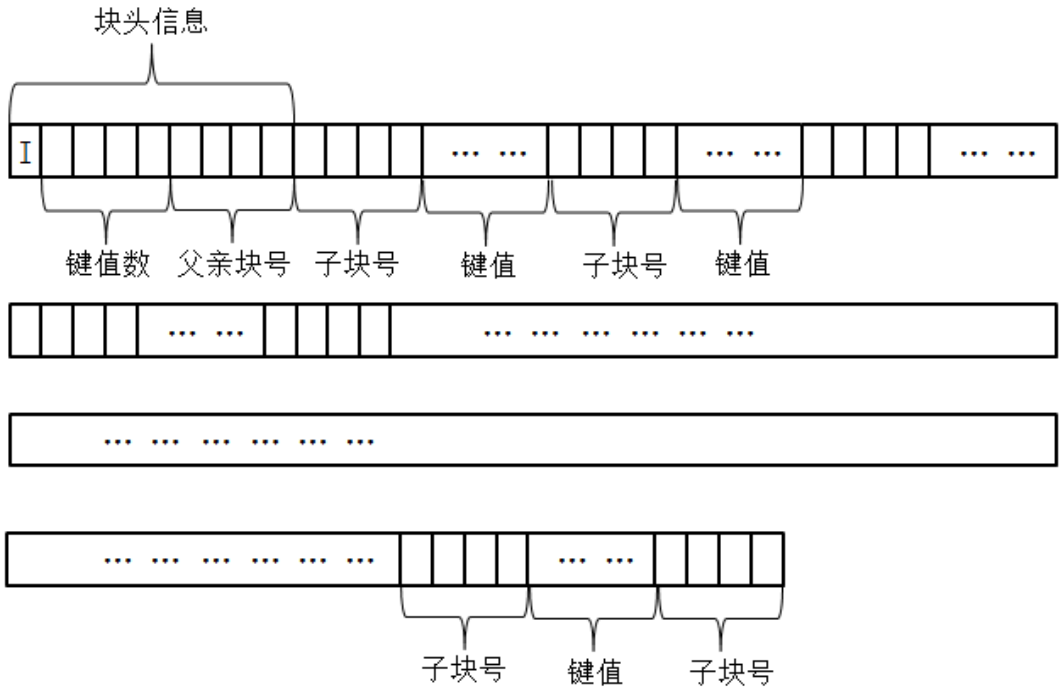
9)删除记录：41 表名 # 条件 1 # 条件 2 # ……（条件内部没有空格）；

10)退出系统：50；

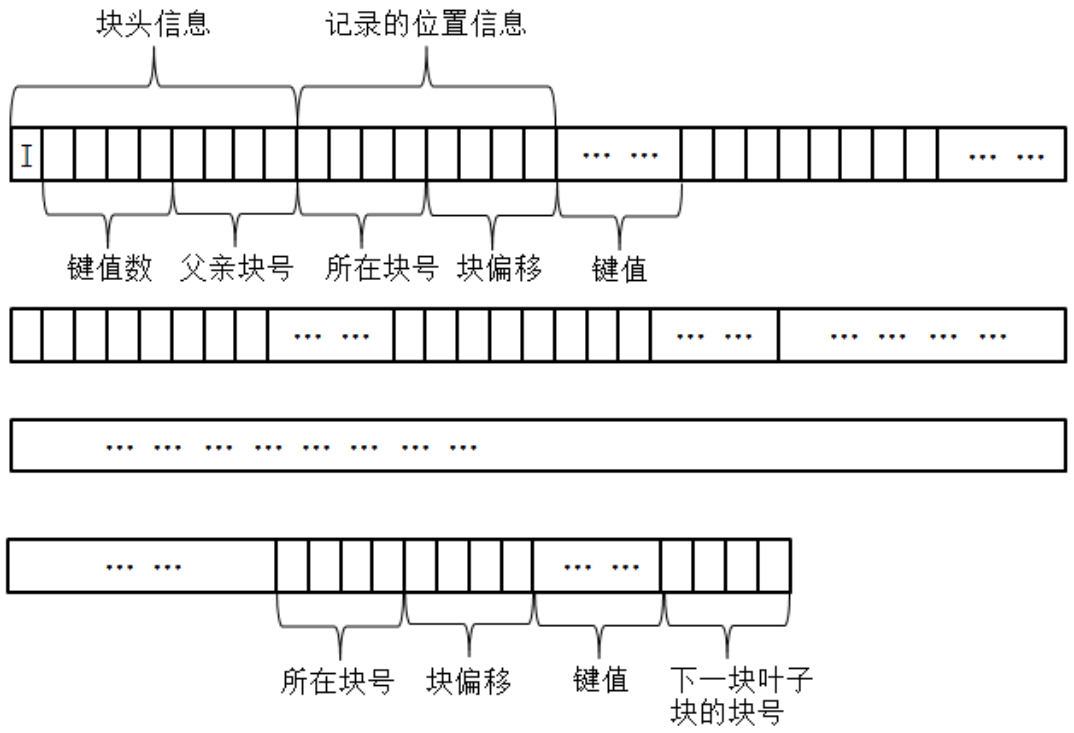
11)执行脚本文件："99"（因为解析的过程中已经调用了执行的函数，传递给 API 时已经执行完了）

PS: 所有属性用一个空格隔开，其中的“#”和“，”两边也是有空格的，故属性可用 splite 函数分离获得

B+树中间块结构设计如下：



B+树叶子块的结构设计如下



Catalog 数据：

存储在 table.xml 和 index.xml 文档中，在创建出 catalog 实例的时候即解析这两个文档并载入内存中由于这两个文档使用频繁并且占用内存并不大，所以由 catalog 单独保管，而不交给 buffermanager 管理。

BufferBlock 数据结构：

为一个链表结构，每个链表节点为一个缓冲块，内部存储 1024B 数据，同时记录了，该块数据的文件名：fileName，记录数：recordNum，是否脏数据标志：dirtyBit，是否上锁标志：lock，指向前一节点和后一节点的指针：previous，next。

第 3.2 节　主窗口及主函数设计

运行系统时界面如下：

```
**************************************************************************
                Welcome to MiiSQL!
                Version(1.0)

copyright(2013) all right reserved!
**************************************************************************


MiniSQL-->
```

SQL 命令在“-->”提示符后输入

第 3.3 节　Catalog Manager 接口

1. 将内存内变化写会 xml 文档

   public void writeDocument(Document d,int i)

2. 创建表

   public void Create_Table(Table table)

3. 创建索引

   public void Create_Index(Index index)

4. 判断表是否存在

   public boolean isTable(String tablename)

5. 判断索引是否存在

   public boolean isIndex(String indexname)

6. 判断该属性名是不是该表中的属性

   public boolean isAttribution(String tablename,String attributionname)

7. 删除表

   public void Drop_Table(String tableName)

8. 删除索引

   public void Drop_Index(String indexName)

9. 设置索引根块在文件中的 block 偏移量

   public void setIndexRoot(String indexName,int number)

10. 获得索引根块在文件中的 block 偏移量， 如果该 index 不存在则返回-1

    public int getIndexRoot(String indexName)

11. 获得表信息，表存在则返回表信息，表不存在则返回空

    public Table getTable(String tableName)

12. 获得索引信息，索引存在有则返回索引信息，没有则返回 null

    public Index getIndex(String indexName)

13. 当表增加了一个 block 时，修改对应的表信息中的 blockNum 增加一

    public void addTableBlockNum(String tableName)

14. 当索引增加了一个 block 时，修改对应的索引信息中的 blockNum 增加一

    public void addIndexBlockNum(String indexName)

15. 获取某张表中第 n 个属性的类型

    public int getAttrType(String tableName, int n)

16. 获取某张表中某个属性的类型

    public int getAttrType(String tableName, String attrName)

17. 判断输入的 word 是否可以转化为表 table 的第 n 个属性的类型

public boolean matchType(String word,String table,int n)

18. att 为 table 中的属性，判断 word 的类型是否和 att 的类型相同

public boolean Type(String att,String word,String table)

19. 根据表名和属性名查找索引，若存在则返回索引，若不存在则返回 NULL

static Index getIndexfromTable(String tableName, String attrName)

第 3.4 节 Record Manager 接口

1. 创建表
public void createTable(Table tableInfo)

2. 删除表，即删除表文件
public void dropTable(Table tableInfo)

3. 插入记录
public void insertValue(Table tableInfo,Record InfoLine)

4. 无条件查找全部属性输出
public void select(Table tableInfo)

5. 无条件查找部分属性输出
public void select(Table tableInfo,selectAttribute selections)

6. 条件查找全部属性输出

public void select(Table tableInfo,Vector<Condition> conditions)

7. 条件查找部分属性输出

public void select(Table tableInfo,selectAttribute

selections,Vector<Condition> conditions)

8. 无条件删除

public void delete(Table tableInfo)

9. 条件删除

public void delete(Table tableInfo,Vector<Condition> conditions)

10. 根据索引提供的位置信息进行查找全部属性输出

public void selectFromIndex(Table tableInfo,offsetInfo off)

11. 根据索引提供的位置信息进行查找部分属性输出

public void selectFromIndex(Table tableInfo,selectAttribute

selections,offsetInfo off)

第 3.5 节 Index Manager 接口

1. 创建索引
public void createIndex(Table tableInfo,Index indexInfo) //需要 API 提供表和索引信息结构

2. 删除索引，即删除索引文件
   public void dropIndex(Index indexinfor)

3. 等值查找
   public offsetInfo searchEqual(Index indexInfo, String key)

4. 插入新索引值，已有索引则更新位置信息
   public void insertKey(Index indexInfo,String key,int blockOffset,int offset)

5. 删除索引值，没有该索引则什么也不做
   public void deleteKey(Index indexInfo,String deleteKey)

第 3.6 节　Buffer Manager 接口

1. 将所有块写回到文件中去

   public static void writeBufferToFile()

2. 根据文件名和偏移量获取块，并将块上锁

   static public BufferBlock readBlock(String FileName, int Offset,boolean lock)

3. 根据文件名和偏移量获取块

   public BufferBlock readBlock(String FileName, int Offset)

4. 将块写回文件

   public void writeBlockToFile(BufferBlock block)

5. 获取表的可插入的位置，找到最后一个块，如果仍有空间则返回块，如果没有则返回 NULL

   BufferBlock getInsertPosition(Table tableInfo,String filename)

6. 删除给定表的给定块的给定位移处的一条记录

   void deleteValues(int blockOffset,int offset,Table tableInfo)

7. 查看 buff 是否为满

   public boolean isFull()

第 3.7 节　DB Files 接口

1. 创建索引

   public void BlocktoFile(String filename,byte[] content, int offset, int recordNum)//将一个缓冲块的内容写回到文件相应位置

2. 删除索引，即删除索引文件

   public BufferBlock FiletoBlock(String filename,int offset)//将文件指定块的内容加载到新的缓冲块中

第 4 章　MINISQL 系统测试

（1）创建表

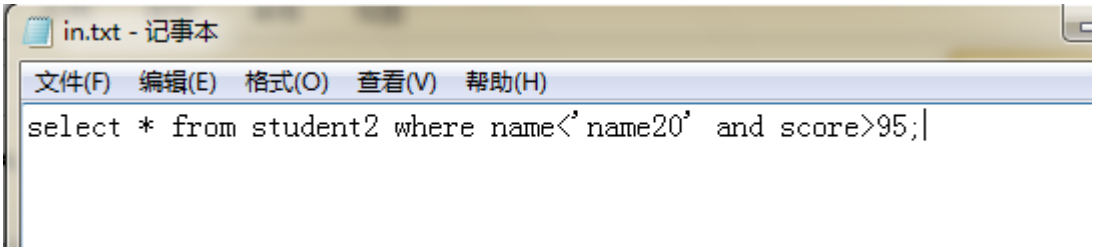
```
MiniSQL-->create table student2(
        id int,
        name char(12) unique,
        score float,
        primary key(id)
);
创建表成功!
创建索引成功!
```

（2）插入数据

```
MiniSQL-->insert into student2 values(1080100001,'name1',99);
插入成功!
```

（3）执行文件

文件内容：



执行结果：

```
MiniSQL-->execfile in.txt;
1
1080100001      name1   99.0
2
1080100107      name107 99.5
3
1080100109      name109 95.5
4
1080100122      name122 96.5
5
1080100123      name123 96.0
6
1080100129      name129 99.5
7
1080100137      name137 98.5
8
1080100159      name159 99.5
9
1080100176      name176 96.5
10
1080100180      name180 95.5
11
1080100184      name184 96.0
12
1080100185      name185 97.5
13
1080100189      name189 96.0
14
1080100194      name194 99.5
time cost without index: 10
```

（4）无索引条件查找：

```
MiniSQL-->select * from student2 where name='name345';
1
1080100345      name345 87.5
time cost without index: 6
```

（5）创建索引：

```
MiniSQL-->create index stuidx on student2 ( name );
创建索引成功!
```

（6）有索引条件查找：

```
MiniSQL-->select * from student2 where name='name345';
1
1080100345      name345 87.5
time cost with index: 1
```

（7）索引插入：

```
MiniSQL-->insert into student2 values(1080197998,'name97998',100);
插入成功!
MiniSQL-->select * from student2 where name='name97998';
1
1080197998      name97998       100.0
time cost with index: 0
```

（8）索引删除：

```
MiniSQL-->delete from student2 where name='name97998';
删除成功!
共删除1条记录!
MiniSQL-->select * from student2 where name='name97998';
不能从索引中找到
time cost with index: 0
```

（9）删除索引：

```
MiniSQL-->drop index stuidx;
索引文件已删除
删除索引成功!
MiniSQL-->select * from student2 where name='name345';
1
1080100345      name345 87.5
time cost without index: 2
```

（10）删除记录

条件删除：

```
MiniSQL-->delete from student2 where score>90;
删除成功!
```

全部删除：

```
MiniSQL-->delete from student2;
索引文件已删除
删除索引成功!
创建索引成功!
您已将表student2.table的内容清空了
```

（10）删除表：

```
MiniSQL-->drop table student2;
表文件已删除
索引student2-primary-idx.index删除
```

（10）退出：

```
MiniSQL-->quit;
Exit.
```

第 5 章　　分工说明

本系统的分工如下：