索引文件IndexFile的命名规则为，”<tableName>\_<attributeName>”。存储在indexes文件夹中。

索引文件都有一个256B的文件头，其内容为。

|  |  |  |
| --- | --- | --- |
| 变量属性 | 变量类型 | 变量字节长 |
| 树节点键值类型keyType | int | 4B |
| 树节点键值长度keyLength | int | 4B |
| 树节点的指针数N值 | int | 4B |
| 树节点的跟节点指针rootPointer | long | 8B |
| 下一个可插入树节点的位置nextPointer | long | 8B |

树节点的大小同为256B，上文中的指针是虚拟的。代表的是以256B为一块的文件的块数。

从零开始计数，由于第0个块为文件头并不是一个有效的树节点。所以，0在指针中我们代表空指针，是无效值。

上文中同时规定了，我们规定的树节点用指针为long型在Java中为64位，其中我们用第63位标记该指针是否指向一个叶节点。规定当该位为0时不是指向叶节点，为1时指向的事叶节点。

这样指针的有效位就是63位，其在文件中的真实位置的计算 Pointer \* 256 B。相当于左移8位，则有效位减小为56位，但是对于10k的数据来说已经够用了。

对于索引文件的读取我们拟使用RandomAccessFile类进行。使用seek进行文件信息的定位。

注：用int表示的键值类型，其中1为int（8B），2为float（8B），3为char（2B/字符）

在对索引文件进行修改时会对文件头中的数据进行更新。

表文件命名规则为，”<tableName>”。存储在tables文件夹中。

用ObjectInputStream和ObjectOutputStream进行访问。

存储信息载体为TableInfo类。

public class TableInfo {  
 //表的名字  
 public String tableName;  
  
 //单条数据的长度  
 public int dataSize;  
  
 //以属性名为键值，第一个值为类型，第二个值为偏移量，第三个值为数据长度,  
 // 使用LinkedHashMap来保证迭代遍历的有序性。  
 // 为后面进行数据转换提供方便。  
 public LinkedHashMap<String , ArrayList<Integer>> attributes;  
  
 public TableInfo(String tableName , int dataSize , LinkedHashMap<String , ArrayList<Integer>> attributes) {  
 this.tableName = tableName;  
 this.dataSize = dataSize;  
 this.attributes = attributes;  
 }  
}

需要注意的是单条数据长度总是4字节对齐的。

ShabbyDB类负责读入执行文件并对相应的创建，维护，检索功能进行调用。

CommandInterpreter类负责将执行命令中的信息从字符串翻译成具体的数据，供接下来的处理方法操作。

StorageEngine类负责提供创建，维护，检索功能。其接受的执行命令为字符串，要调用CommandInterpreter类的方法对命令进行翻译后

BPlusTree类是对一个检索项进行检索的单位。

通过接口BPlusTreeBasicMethods和BPlusTreeNodeMethods将底层的文件操作，与B+树的具体算法分离开来。实现对B+树基本原理算法的一一对应实现。

public interface BPlusTreeBasicMethods {  
  
  
 */\*\*这些方法返回指向树节点的指针  
 \* 我们对树节点的读取都需要通过指针\*/* Object getNewBPlusTreeNodePointer();  
 Object getRootPointer();  
  
  
  
  
 */\*\*判断一个指针是否指向叶节点\*/* boolean isLeaf(Object bPlusTreeNodePointer);  
  
  
  
 */\*\*通过给定的树节点指针读取出树节点  
 \* 再通过树节点类访问和修改其中的内容\*/* Object getNode(Object bPlusTreeNodePointer);  
 */\*\*给出一个临时的树节点对象用于方便之后树节点分裂\*/* Object getTempBPlusTreeNode();  
  
  
  
  
  
  
 */\*\*通过比较键值进行目标的搜索\*/* Object getKeyValue(Object key);  
 int compareKeyValue(Object keyValue1 , Object keyValue2);  
 public static final int *GREATER\_THAN* = 1;  
 public static final int *LESSER\_THAN* = -1;  
 public static final int *EQUALS* = 0;  
  
  
  
 */\*\*保存你所改变的树节点  
 \* 这里需要你要保存的位置和你要保存的内容\*/* void saveTreeNodeChanges(Object bPlusTreeNodePointer , Object bPlusTreeNode);  
  
  
 */\*\*在树结构发生变动后当根节点发生改变时要改变根节点的指针\*/* void setTreeRoot(Object bPlusTreeNodePointer);  
  
  
 */\*\*得到该树的索引名\*/* String getAttributeName();  
}

public interface BPlusTreeNodeMethods {  
  
  
 */\*\*一个树节点的最大长度\*/* public static final int *nodeSize* = 256;  
  
 */\*\*一个指针的长度\*/* public static final int *pointerSize* = 8;  
  
  
 */\*\*未赋值与越界情况下返回空值\*/* public static final Object *EMPTY* = null;  
  
  
 */\*\*初始规定每个树节点规定有N个指针，随数据长度的变化可能会减少\*/  
  
  
 /\*\*取出第index个键值，  
 \* 与修改第index个键值\*/* public /\*abstract\*/ Object getKey(int index);  
 public /\*abstract\*/ void setKey(int index , Object key);  
  
  
  
 */\*\*得到第index个指针  
 \* 与设置第index个指针\*/* public /\*abstract\*/ Object getPointer(int index);  
 public /\*abstract\*/ void setPointer(int index , Object pointer);  
  
  
  
 */\*\*得到当前树节点所存储的指针数\*/* public int getSize();  
  
 */\*\*得到树节点的N值\*/* public int getN();  
}

DataFile用于单独存放表中的全部数据。其与IndexFile相似的地方在于，他也拥有一个256B的文件头用于存放必要信息，且定位方式同为用事先规定好的虚拟指针规则找到其在文件中的真实地址。文件头内容为：

|  |  |  |
| --- | --- | --- |
| 变量属性 | 变量类型 | 变量字节长 |
| 单条数据长度dataSize | int | 4 |
| 下一个可存数据的地址nextPointer | long | 8 |
|  |  |  |

真实地址的计算方式是

真实地址 = (虚拟指针地址 - 1) \* dataSize + 256B；

虚拟指针地址从1开始计数，0为空值。

在存入数据后会对文件头中的数据进行更新。