RecordManager 详细报告

模块功能：

- 对外提供Record的select insert delete功能
- 对外提供drop table功能，并连带清除表格的索引文件。

类说明：

```
class RecordManager
{
public:
    DATA select(Table& T, vector<Condition>& con, char op = 'a');
    RecordResult insert(Table& T, vector <string>& value);
    RecordResult deleteR (Table& T, vector<Condition>&con, char op='a');
    using TableResult = RecordResult;
    TableResult drop(const Table& T);
private:
    void SetCondition(bool&, vector<Condition>&,char);
    void ScanTable(Table&,DATA&, vector<Condition>&, bool negative=0);
    void ScanTable(Table&, RecordResult&, vector<Condition>&, bool negative =
0);
    bool comp(Table&,Condition&, Condition&);
    void MySort(Table&, vector<Condition> &);
    void splitRow(const string&, Row&, Table&);
    void testifyOneRow(Row&, vector<Condition>&, bool&, DATA&);
    bool testifyOneRow(Row&, vector<Condition>&, bool&);
    void deleteIndex(Table&, Row&);
    void insertIndex(Table&, Row&,const Pointer&);
    void checkDuplicate(Table&, vector<string>&, RecordResult&);
};
```

对外接口：

- DATA select(Table& T, vector& con, char op = 'a');
    - 本接口提供对record的select服务。传入参数为描述表格的数据结构、描述所有where中字句条件的数据结构，以及一个标志op。当op为'a'时，表示vector con 中的元素都是用 合取操作连接的。可以把op置为 'o'，以表示析取操作。返回值是一个DATA类型的数据结构，里面包含了一个向量，此向量的每个元素是一个记录表格元组的向量。为减小测试开销，我们直接返回整个元组，在主调函数中输出被选择的属性。
- RecordResult insert(Table& T, vector & value);
    - 本接口提供对record的insert服务。本接口一次性只能插入一条记录。传入参数除了T外，还有一个依次存储元组属性的向量。
    - 插入时，会读入文件最后一个块。首先在块内寻找是否有剩余空间，如果没有则再寻找是否有元组已经被标记为删除，如果仍然没有则会为文件新建一个块，然后写回文件。
    - 插入之前应该对此元祖内的所有有索引的属性进行重复性测试。一旦检测到重复，则将错误写入存储插入结果的数据结构中。
- RecordResult deleteR (Table& T, vector&con, char op='a');

    - 本接口提供对record的delete服务。本接口一次性只能删除一条记录。与select相似，本接口也需要一个存储where字句中所有条件的向量con，以及一个标志用来标记con中元素的连接方式。
    - 删除时，仅仅将元组标记为删除，实际上内容仍然在磁盘中。
- TableResult drop(const Table& T);
    - 本接口对外提供删除表格的服务。本接口首先依次删除这个表格的所有索引，然后删除表格的record文件。

内部接口：

- void SetCondition(bool&, vector&,char);
    - 本接口对外部传入的con进行处理。当外部传入的op'o'时，代表要进行析取操作。此处选择使用德摩根定理，将所有子条件取反，再将总体结果取反。具体地，本函数会判定传入的bool值，当为1时，代表最终结果要取反，即op'o'，于是函数会遍历所有条件结构，并将操作符取反。
- void MySort(Table&, vector &);
    - 本接口对外部传入的条件结构进行排序。显然，我们应该先做有索引的部分的筛选。经过细致分析，对于该属性的操作符也对选择效率有关。综合来看，各个条件的大小顺序可以按照以下排列：
        - 1st：用于判定的条件是 '='符号，并且该元素有索引。
            - 当op=='a'时，此条件仅仅是where中的一个字条件。用索引取出此元组，再进行其他子条件测试。综合来看，只用对一个元祖进行条件测试。
            - 当op=='o'时，由于之前已经取过反，因此取反后的字句也是通过and进行连接的。最终结果要进行取反，故凡是不满足这个条件的元组都应该被返回；而满足这个条件的元组，应该有且仅有一个，我们应该对它测试其他条件，如果有其它条件也不满足，则此元组也应当被返回。只有当此元组满足所有条件时，由于最终结果取反，它不应当被返回。整个判定过程不可避免地需要对整张表格进行遍历，令人欣慰的是，只需要对绝大多数表格测试一个条件即可。
        - 2nd：用于判定的条件是 > / < / ≥ / ≤ ，并且该元素有索引。
            - 当op=='a'时，此条件仅仅是where字句中的一个子条件。我们可以知道，只要不满足这个条件的元组一定不会被返回。因此我们通过索引找到满足这个条件的所有元组，并对它们进行其它条件测试。概括地说，只需要对部分元组进行条件测试。
            - 当op=='o'时，由于之前已经取过反，因此取反后的字句也是通过and进行连接的。那么我们知道，凡是不满足这个条件的元组都应该被返回；而对于满足这个条件的元组，我们应该测试其它条件，并且只返回至少不满足1个条件的元组。概括地说，我们通过索引将表格划分成两部分，然后遍历整张表格，有些元组被直接push到结果中，有些元组则需要进行条件测试。
        - 3 rd：用于判定的条件是 != 或者 <>。
            - 无论op为 'a' 或者 'o' ，我们都需要遍历整张表格才能决定某个元组应不应该被push到最终的结果中。
        - 4 th：无任何索引可用。
- bool comp(Table&,Condition&, Condition&);
    - 本接口实现上述的4中rank的大小判定。在MySort中被调用。
- void splitRow(const string&, Row&, Table&);
    - 本接口将一个描述元组的字符串拆分为单个的属性，装入Row类型的数据对象中。
- void ScanTable(Table&,DATA&, vector&, bool negative=0);
    - 此重载版本由select调用，直接将结果写入DATA类型的数据中心，negative表示最终结果是否取反，即 op=='o' ?
    - 此函数根据排序后的con中第一个条件来判定是否需要遍历整个表格。

    - 会调用testify的一个重载版本。
- void testifyOneRow(Row&, vector&, bool&, DATA&);
    - 此重载版本由上面的ScanTable函数调用
    - 此函数对一个元组进行条件的测试，如果符合条件则直接push到DATA类型的数据结构中。
- void ScanTable(Table&, RecordResult&, vector&, bool negative = 0);
    - 此重载版本由deleteR调用，将删除状态写入RecordResult类型的数据结构中，这个数据结构用来记录删除是否成功以及失败原因。
    - 此函数会通过排序后的con的第一个条件来判定是否需要遍历整个表格。
- bool testifyOneRow(Row&, vector&, bool&);
    - 此重载版本由上一个ScanTable调用，只会返回此元组判定结果的布尔值。
- void checkDuplicate(Table&, vector&, RecordResult&);
    - 在插入元组到磁盘之前，应该对有索引的属性进行重复性测试。有的话则将重复情况写入RecordResult类型的数据结构中。
- void insertIndex(Table&, Row&,const Pointer&);
    - 如果此元组被成功插入到表格中，需要将每个有索引的元素插入到索引中去。
- void deleteIndex(Table&, Row&);
    - 如果此元组被成功删除，则需要对每个有索引的属性从对应的索引中移除。

数据结构定义：

```
struct RecordResult
{
    bool status;
    string Reason;
    RecordResult() :status(1) {}
};
using TableResult=RecordResult;
struct Row
{
    vector<string> DataField;
};

struct DATA
{
    vector<Row> ResultSet;
};
```

测试样例：

对于insert，delete，select的测试样例详见附件testInsert.txt, testDelete.txt, testSelect.txt 。

友情提示：解析器对部分查询语句的空格要求较为严格，输入查询语句前请详细参考解析器使用手册。