二进制文件存储系统FileDB使用说明文档（C++版）V3.0

一.说明：

FileDB为一款C++开发的类数据库系统，基于二进制文件，支持数据库基本的增删改查（CRUD）操作，并且可以多条件查询，模糊查询，自动生成对象关系映射（ORM）等。在此基础上可以轻松实现MVC设计模式以及UI、DAL、DLL三层架构。

二.使用方法

1.初始建表

初始使用请使用CreateTable.exe建立数据库表，表中字段支持int、float、char*三种类型，最多可设置10个字段，且必须有id字段作为主键。按照提示完成操作后， 系统会自动生成三个文件：.cpp为系统自动生成的实体类，为数据库中表的映射，两个.dat为数据存储文件。将这三个文件连同FileDB.cpp一起导入到工程中即可使用了。

2.插入数据

插入数据的函数原型为：

```
template<typename T>
static int insert(string DB_NAME, vector<T>& entity)
```

其中，DB_NAME为表名，entity为待增加实体数组，可以一次性增加多条数据，函数操作成功，返回受影响的行数，操作失败，返回-1。
调用方法：

```
vector<Person>entity;
Person person;
person.setName("小张");
person.setAge(20);
person.setHeight(150.5);
entity.push_back(person);
int res = FileDB::insert("person", entity);
if (res > 0) {
    //success
}
```

2.查询数据（精确）

查询数据的函数原型为：

```
template<typename T>
static int select(string DB_NAME, T& entity, vector<string>& VALUES, vector<T>& resultSet)
```

其中，DB_NAME为表名，entity为选择条件，VALUES为选择字段，VALUES第一个参数为选择方式，为"all"是选择全部，相当于"SELECT ** FROM TABLE",为其他值时，为条件查找，相当于"SELECT * * FROM TABLE WHERE.....",resultSet为结果集，函数操作成功，返回受影响的行数，操作失败，返回-1。
调用方法：

```
//选择全部
Person person;
vector<string>VALUES;
vector<Person>res;
VALUES.push_back("all");
FileDB::select("person", person, VALUES, res);
for (int i = 0; i < res.size(); i++) {
    cout << res[i].id << " " << res[i].age << " " << res[i].height << " " << res[i].name << endl;
}
//条件选择，选择id为1的person
Person person;
vector<string>VALUES;
vector<Person>res;
VALUES.push_back("one");
VALUES.push_back("id");
person.setId(1);
FileDB::select("person", person, VALUES, res);
for (int i = 0; i < res.size(); i++) {
    cout << res[i].id << " " << res[i].age << " " << res[i].height << " " << res[i].name << endl;
}
```

3.模糊查询（基于子串）

模糊查询的函数原型为：

```
template<typename T>
static int selectLike(string DB_NAME, string valueName,char* value, vector<T>& resultSet)
```

目前仅支持字符串的模糊查询，其中DB_NAME为表名，valueName是字段名称，value是模糊查询条件，resultSet为结果集，函数操作成功，返回受影响的行数，操作失败，返回-1。
调用方法：

```
//查找姓名中含有“张”的
vector<Person>res;
FileDB::selectLike("person", "name", "张", res);
for (int i = 0; i < res.size(); i++) {
    cout << res[i].id<<" "<< res[i].name << " " << res[i].age << endl;
}
```

4.模糊查询（基于正则表达式）

模糊查询的函数原型为：

```
template<typename T>
static int selectRegex(string DB_NAME, string valueName, regex& rx, vector<T>& resultSet)
```

目前仅支持字符串的模糊查询，其中DB_NAME为表名，valueName是字段名称，rx字符串匹配的正则表达式，resultSet为结果集，函数操作成功，返回受影响的行数，操作失败，返回-1。
调用方法：

```
//查找姓名中含有“小...明”的
vector<Person>res;
regex rx(".*小.*明.*");
FileDB::selectRegex("person", "name", rx, res);
for (int i = 0; i < res.size(); i++) {
    cout << res[i].id<<" "<< res[i].name << " " << res[i].age << endl;
}
```

5.删除数据

删除数据的函数原型为：

```
template<typename T>
static int Delete(string DB_NAME, T& entity, vector<string>& VALUES)
```

其中，DB_NAME为表名，entity为删除条件，VALUES为选择字段，VALUES第一个参数为删除方式，为"all"是删除全部，相当于"DELETE ** FROM TABLE",为其他值时，为条件删除，相当于"DELETE FROM TABLE WHERE.....",函数操作成功，返回受影响的行数，操作失败，返回-1。
调用方法：

```
//删除全部
Person person;
vector<string>VALUES;
VALUES.push_back("all");
int res = FileDB::Delete("person", person, VALUES);
if (res > 0) {
    //success
}
//条件删除，删除id为1的person
Person person;
vector<string>VALUES;
VALUES.push_back("one");
VALUES.push_back("id");
person.setId(1);
int res = FileDB::Delete("person", person, VALUES);
if (res > 0) {
    //success
}
```

6.修改数据

修改数据的函数原型为：

```
template<typename T>
static int update(string DB_NAME, T& Sentity, T& Uentity, vector<string>& VALUES)
```

其中，DB_NAME为表名，entity为修改条件，Uentity为更新内容，VALUES为选择字段，函数操作成功，返回受影响的行数，操作失败，返回-1。
调用方法：

```
Person person;
vector<string>VALUES;
VALUES.push_back("one");
VALUES.push_back("id");
person.setId(1);
Person newPeople;
newPeople.setAge(50);
newPeople.setHeight(165.5);
newPeople.setName("小张");
int res = FileDB::update("person", person,newPeople, VALUES);
if (res > 0) {
    //success
}
```

V2.0更新说明：

1、增加了模糊查询功能
2、增加了批量添加数据的功能
3、修复了当id%256==26时，出现文档结束标记EOF导致数据读取不完整的bug
4、增加了单条件查询的加速算法，id查询的加速算法，查询速度显著提高
5、运用了记忆化搜索的方法，处理多条件查询过程，查询速度明显提高

V3.0更新说明：

1、增加了基于正则表达式的模糊查询