Record Manager 设计报告

数字媒体 3180101939 陆子仪

一、模块概述

Record Manager 负责记录表中的数据。主要功能有记录键值、元组和单条记录、元组数据的插入、删除和更新。此外，Record Manager 提供对键值的比较，对键值数据和类型的访问功能，供上层的 API 调用。

二、设计思路

使用类记录每一条单独的记录数据，每一个记录数据包括数据类型（Int，Float，Char）以及数据本身，并提供对针对不同数据类型之间的操作符重载便于索引文件和 B+树的建立。在使用 Column_Head 记录一串通过指针相连的 Column_Cell 类以储存一个元组的数据，其中需要记录每一个单独数据的数据大小以及总的元组的数据大小，以便将其写入文件。

三、数据结构以及类定义

1，数据类型定义

I 表示 Int 类型数据，F 表示 Float 类型数据，C 表示 Char 类型数据

```
enum class Value_Type { I, F, C, ERROR = -1 };
```

2，数据的联合数据类型

使用 union 类型，该结构体中的每一个字段共享一个内存，因此可以节约一定的内存开销，并且方便储存数据。

```
union KeyValue {
    int      IntValue;
    char     CharValue[16];
    double    FloatValue;
};
```

3，键值类（Key_Attr）

记录一个键值数据，并且带有<,>,==,<=,>=,!=的操作符重载，便于对键值进行比较，以建立索引文件。同时使用<<的重载函数可以将数据输出到标准流中。

```
class Key_Attr {
public:

    Value_Type type;
    KeyValue value;
    Key_Attr();
    Key_Attr(int value) {
        type = Value_Type::I;
        this->value.IntValue = value;
    }
    Key_Attr(double value) {
        type = Value_Type::F;
        this->value.FloatValue = value;
    }
    Key_Attr(char* value) {
        type = Value_Type::C;
        strcpy(this->value.CharValue, value);
    }
    Key_Attr(const Key_Attr& k);
    Key_Attr& operator=(const Key_Attr& k);

    bool operator<(const Key_Attr& k)const;
    bool operator>(const Key_Attr& k)const;
    bool operator==(const Key_Attr& k)const;
    bool operator<=(const Key_Attr& k)const;
    bool operator>=(const Key_Attr& k)const;
    bool operator!=(const Key_Attr& k)const;
```

```
    friend ostream& operator<<(std::ostream& os, const Key_Attr& key);
};
```

4，单元类（Column_Cell)

记录一个表中的单个单元的数据，包括数据类型，数据以及大小等信息

```
class Column_Cell
{
public:
    Value_Type        column_type;
    string            column_name;
    Column_Value      column_value;
    Column_Cell* next;
    size_t            sz;

    Column_Cell();
    Column_Cell(Key_Attr k);
    Column_Cell(const Column_Cell& c);
    Column_Cell& operator=(const Column_Cell& c);

    size_t size()const {
        if (column_type == Value_Type::I) {
            return sizeof(int);
        }else if (column_type == Value_Type::F) {
            return sizeof(double);
        }
        else {
            return sz;
        }
    }
    void* data()const;

    operator Key_Attr()const;
};
```

5，元组类（Column_Head)

记录一个元组的数据，形式是以指针相连的一串单元（Column_Cell)，并且记录了该元组的数据大小

```
class Column_Head {
private:
    Column_Cell* front;
    Column_Cell* rear;
public:
    Column_Head();
    Column_Head(Column_Cell c);
    Column_Head(const Column_Head& h);
    Column_Head& operator=(const Column_Head& h);

    void attachCell(Column_Cell& c);
    size_t size();
    Column_Cell* firstCell();
};
```

四、函数接口

Record Manager 中的主要函数接口全是 Record 类的成员函数，在调用时先获取全局的 Record 类，然后就可以调用以下的函数了。

Record 类:

```
class Record
{
public:
    FileAddress InsertRecord(const string file_name, Column_Head& h);
    FileAddress DeleteRecord(const string file_name, FileAddress fa,
size_t);
    bool UpdateRecord(const string file_name, Column_Head& h,
FileAddress fa);
private:
    std::pair<unsigned long, char*> GetRecordData(Column_Head& h);
};
```

1. 插入一条记录

FileAddress InsertRecord(const string file_name, Column_Head& h);

2. 删除一条记录

FileAddress DeleteRecord(const string file_name, FileAddress fa, size_t);

3. 更新一条记录

bool UpdateRecord(const string file_name, Column_Head& h, FileAddress fa);

五、关键函数和代码

1. GetRecordData

输入参数为一个 Column_Head 即一个元组，返回元组的数据大小（size_t）以及将所有数据写入一个字符串（char*）进行返回。在返回数据时，为返回多条不同类型的记录，使用了 STL 中的 pair 进行返回。

```
std::pair<unsigned long, char*> Record::GetRecordData(Column_Head& h)
{
    unsigned long data_size = h.size();
    char* data = new char[data_size];
    memset(data, 0, data_size);
    Column_Cell* p = h.firstCell();

    unsigned long offset = 0;
    while (p)
    {
        memcpy(data + offset, p->data(), p->size());
        offset += p->size();
        p = p->next;
    }
    if (offset != data_size)
    {
        cout << "ERROR OCCURS (-1)" << endl;
    }
```

```
    //解决返回多条记录的问题
    pair<unsigned long, char*> sp(data_size, data);
    return sp;
}
```