数据库系统设计-MiniSQL

CatalogManager模块详细设计报告

黄昭阳 3130000696 - 11/6/15

catalogManager模块总体设计

一、实验要求：

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。

二、实验完成情况：

完美实现实验要求功能，catalogManager存储了实验要求的三项功能，并提供一下四个接口用来提供给API模块有关表和索引的消息：

```
string* getRecordFileName(string* tableName);
string* getIndexFileName(string* indexName);
SQLTable* getTableInfo(string* tableName);
SQLIndex* getIndexInfo(string* indexName);
```

这里需要注意的是，这四个接口只会被API所调用而并不会被Interpreter模块直接调用，Interpreter如需要获取相关信息，需要使用SQLCommandInterpreterDelegate的委托通过API模块来调用并使用相关的信息，这样可以有效的降低Interpreter和CatalogManager模块之间的耦合性并提高代码重用性，更符合MVVM设计理念，方便之后整合六个模块时的整体调试。

模块整体详细分析

一、CatalogManager整体结构分析和代码介绍：

```
class catalogManager
{
private:
    vector<SQLTable*> tables;
    vector<SQLIndex*> indexes; //用来存储表单和索引的信息

    bufferManager* buffer;  //catalog也将通过buffer来处理缓存和文件内容
    unsigned char* pointer; //辅助catalog处理的变量

    SQLTable* parseTable();
    SQLAttribute* parseAttribute();
    SQLIndex* parseIndex();
  //由于要从文件中读取相关的catalog信息，这三个函数分别是从缓存区指定位置（pointer）中解析出相关的数据结构SQLTable、SQLAttribute、SQLIndex来。

    void writeTable(SQLTable* table);
    void writeIndex(SQLIndex* index);
    void writeAttribute(SQLAttribute* attribute);
  //要将catalog的有效信息写入到缓冲区并最终写入磁盘中，这三个函数分别是将三种数据结构的内容以相应格式写入指定的缓冲区位置（pointer）中。

public:
    catalogManager(string* fileName);
    ~catalogManager();
    void createTable(SQLTable* table);
    void createIndex(SQLIndex* index);
    void dropTable(string* tableName);
    void dropIndex(string* indexName);
//提供四个接口给API调用，分别完成函数名对应的逻辑功能

    string* getRecordFileName(string* tableName);
    string* getIndexFileName(string* indexName);
    SQLTable* getTableInfo(string* tableName);
    SQLIndex* getIndexInfo(string* indexName);
//提供四个接口给API调用，使API能够获得catalog内的消息，并能够将消息分发给其他模块，完成来自其他模块的委托。

    void printTableInfo(string* tableName);
    void printIndexInfo(string* indexName);
    void printTables();
    void printIndexes(); //测试时使用的打印catalog内部消息的相关函数，不必了解
};
```

二、CatalogManager使用方法：

catalogManager的使用方法十分简单，函数内部的实现机制也不复杂，函数提供的逻辑功能也与函数名意义对应，一下为使用样例，一目了然。

```
    catalog = new catalogManager(new string("/Users/drinkingcoder/
Documents/university/Database Design/MiniSQL-final/lib/catalog"));
  //传入一个文件名来实例化一个catalogManager，它将从对应的文件内读取数据获得所有的表
单和索引信息。

    catalog->createTable(new SQLTable(*table));
  //创建一个新的表单，表单信息通过一个SQLTable结构传入catalog

    catalog->dropTable(new string(*tableName));
  //删除一个表单，表单的名字存在tableName内

    catalog->createIndex(new SQLIndex(*index));
  //创建一个新的索引，索引信息通过一个SQLIndex结构传入catalog

    catalog->dropIndex(new string(*indexName));
  //删除一个索引，索引的名字存在indexName内

    string* getRecordFileName(string* tableName);
    string* getIndexFileName(string* indexName);
    SQLTable* getTableInfo(string* tableName);
    SQLIndex* getIndexInfo(string* indexName);
  //四个接口的用途十分明了，以字符串的形式传入需要信息的名字，返回对应的信息
```

模块关键函数详细介绍与分析

由于catalogManager的实现十分简单，比如获取表单信息就是依据名字从表单容器中找到相应的表单输出等等，逻辑十分直白，这里仅举createTable和dropTable两个例子，不做过多赘述。

1. createTable函数：

```
void catalogManager::createTable(SQLTable* table)
{
    tables.push_back(table);
    for(vector<SQLAttribute*>::iterator itr = table->attributes.begin();
itr != table->attributes.end() ; itr++)
        if(((*itr)->attributeProperty & PRIMARY) != 0)
        {
            string* indexName = new string(*(*itr)->attributeName +
string("+index"));
            string* tableName = new string(*(table->tableName));
            string* attributeName = new string(*((*itr)->attributeName));
            SQLIndex* index = new
SQLIndex(indexName,tableName,attributeName);
            createIndex(index);
        }
}
```

分析：创立一个新的表单的同时需要对表单中的primary key建立相应的索引，并将表单加入表单容器tables中即可。注意，由于interpreter中并不支持在Create Table命令中直接建立非primary key的索引，想要建立相关索引需要在之后使用Create Index命令。

2. dropTable函数：

```
void catalogManager::dropTable(string* tableName)
{
    for(vector<SQLTable*>::iterator itr = tables.begin(); itr!
=tables.end() ; )
        if( *((*itr)->tableName) == *tableName)
        {
            for(vector<SQLIndex*>::iterator itr = indexes.begin(); itr !=
indexes.end() ; )
                if( *(*itr)->tableName == * tableName) dropIndex(new
string(*(*itr)->indexName));
                    else itr++;
            delete *itr;
            tables.erase(itr);
            break;
        } else itr++;
    delete tableName;
}
```

分析： 通过传入的表单名在表单容器tables中找到相应的表单，同时将表单上建立的索引使用dropIndex命令也全部删除掉。

catalogManager模块总结

1. 在老师提供的实验要求中，catalogManager需要为Interpreter和API两个模块提供消息反馈接口，但是在我们最终的设计中，catalogManager只为API一个调度模块提供消息反馈接口，Interpreter及其他模块如需要获得catalog中的表单和索引消息，需要使用委托通过API模块来获得，以降低模块间耦合度。
2. catalogManager模块的实现相对简单，只是读、写文件时的一些细节处理需要稍加主义，模块接口逻辑功能简单明了，且内部实现也十分方便。