**Catalog Manager 模块设计说明**

作者：2016级软件工程专业

吕明远 3160105001

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。
4. 数据表中的记录条数及空记录串的头记录号。
5. 数据库内已建的表的数目。

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。

为减小模块之间的耦合，Catalog模块采用直接访问磁盘文件的形式，不通过Buffer Manager，Catalog中的数据也不要求分块存储。

数据结构：

class catalogManager {

public:

struct database {

int numberOfIndex;

int numberOfTable;

};

struct index {

string nameOfIndex;

string tableOfIndex;

string keyOfIndex;

};

struct table {

string nameOfTable;

int numberOfKey;

vector<string> nameOfKey;

vector<string> nameOfType;

vector<int> isUnique;

vector<string> nameOfIndex;

int numberOfRecord;

int lengthOfRecord;

string primaryKey;

};

vector<database> mydb;

vector<table> myt;

vector<index> myi;

int tableNum;

int indexNum;

catalogManager();

~catalogManager() {};}

**其具体实现也是按照语句类型进行分别处理：**

创建数据库语句

CREATE DATABASE 数据库名；

根据Interpreter处理生成的初步内部数据形式，提取数据库名，验证是否有重名现象。若无则建立数据库文件夹及数据字典文件，否则不能创建该数据库并打印出错信息供用户参考。

创建表语句

CREATE TABLE 表名 （

列名 类型

列名 类型

…………..

PRIMARY KEY(列名)

）；

根据Interpreter处理生成的初步内部数据形式，提取表名，验证是否有重名现象。若无则建立表文件，同时将表的各种信息记录在数据字典内，若有主键定义，应对主键生成索引，并将索引信息记录在数据字典内。若有重名表，则创建表命令失败，并打印出错信息。

int createTable(string nameOfTable, vector<string> nameOfKey, vector<string> nameOfType, vector<int> isUnique);

int createTable(string nameOfTable, vector<string> nameOfKey, vector<string> nameOfType, vector<int> isUnique, string primaryKey);

创建索引语句

CREATE INDEX 索引名 ON 表名（列名）；

根据Interpreter处理生成的初步内部数据形式，提取表名及列名，在数据字典中查找有无该表及属性的定义，且验证是否可对该属性建立索引。若可建，则创建索引文件，同时将索引信息记录在数据字典内。若不可建索引，则根据不同类型打印出错信息供用户参考。

int createIndex(string nameOfIndex, string tableOfIndex, string keyOfIndex);

选择语句

SELECT \* FROM 表名； SELECT 列1，列2，….列n FROM 表名；

SELECT \* FROM 表名 WHERE 条件； SELECT 列1，列2，…列n FROM 表名 WHERE 条件

根据Interpreter处理生成的初步内部数据形式，提取表名及各列名，在数据字典中查找有无该表及各属性的定义，同时须查找条件中属性名是否在表中有定义，且常量值的数据类型与属性的数据类型是否可比。若有条件，须先根据条件及条件中已建索引的属性名来选择一个合适的索引供Record Manager使用。任何的不符都将打印出相应的出错信息供用户参考。

int isTableExist(string tablename);

int isIndexExist(string indexname, string tablename);

int isIndexExist(string indexname);

插入记录语句

INSERT INTO 表名 VALUES ( 值1，值2，…….,值n)；

根据Interpreter处理生成的初步内部数据形式，提取表名及各属性值，在数据字典中查找该表的定义，并验证各属性值是否满足各属性的数据类型定义。同时将各属性值转化为记录形式。任何的不符都将打印出相应的出错信息供用户参考。

int numberOfRecordAdd(string nameOfTable, int a);

删除记录语句

DELETE FORM 表名；

根据Interpreter处理生成的初步内部数据形式，提取表名，在数据字典中查找该表的定义。若有，删除该表文件内的所有记录并更新该表在数据字典中的信息,同时须查找建立在该表之上的所有索引，并将所有索引文件内的节点删除且须更新索引文件在数据字典中的信息。若在数据字典中无该表的定义，则须打印出必要的出错信息供用户参考。

DELETE FORM 表名 WHERE 条件；

根据Interpreter处理生成的初步内部数据形式，提取表名及WHERE条件，，在数据字典中查找有无该表的定义，同时须查找条件中属性名是否在表中有定义，且常量值的数据类型与属性的数据类型是否可比。若有条件，须先根据条件及条件中已建索引的属性名来选择一个合适的索引供Record Manager 使用。任何的不符都视为语句的语义错误，并打印出相应的错误信息供用户参考。

int deleteTableInfo(string nameOfTable);

int deleteTableInfo(string nameOfTable, int d);

**退出MiniSQL系统语句**

EXECFILE 脚本文件名；

根据Interpreter处理生成的初步内部数据形式，提取脚本文件名，并查找有无该文件。若有，则打开文件，依次读取每条命令并执行。若不存在该文件，则打印出相应的出错信息供用户参考。

无

删除索引语句

DROP INDEX 索引名；

根据Interpreter处理生成的初步内部数据形式，提取索引名，在数据字典中查找有无该索引的定义。若有，则删除该索引在数据字典中的信息，并删除索引文件。若无，则打印出相应的出错信息供用户参考。

int deleteIndex(string nameOfIndex);

**删除表语句**

DROP TALBE 表名；

根据Interpreter处理生成的初步内部数据形式，提取表名，在数据字典中查找有无该表的定义。若有，则先查找该表上的所有索引并删除，然后删除该表在数据字典中的信息，最后删除该表文件。若在数据字典中无该表的定义，则打印出相应的出错信息供用户参考。

int dropTable(string nameOfTable);

删除数据库语句

DROP DATABASE 数据库名；

根据Interpreter处理生成的初步内部数据形式，提取数据库名，查找有无以数据库为名的文件夹。若有，删除文件夹内所有的文件并删除文件夹。若无此文件夹，则打印出相应的出错信息供用户参考。

无

使用数据库语句

USE 数据库名；

根据Interpreter处理生成的初步内部数据形式，提取数据库名，并查找有无以数据库为名的文件夹。若有，则先判断是否与当前使用的数据库一致，若不一致，才须写回当前数据库的缓冲区内的内存块，并将当前数据库更新为所指定的数据库名。若不存在所查找的文件夹，则表明不存在用户指定的数据库，则须打印出相应的出错信息供用户参考。

无

**数据字典文件格式：**

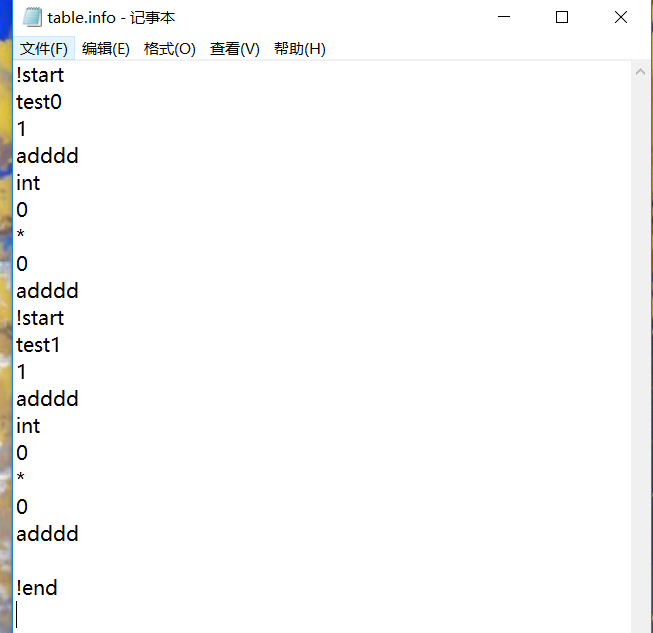
Db.info 用于储存数据库中表数量与索引数量



Index.info用于储存索引信息



Table.info用于储存表信息



其主要函数的功能描述如下：

int isTableExist(string tablename);

判断表是否存在

int isIndexExist(string indexname, string tablename);

判断索引是否存在

int isIndexExist(string indexname);

判断索引是否存在

int myprint(string tablename);

打印表的信息（测试用

int createTable(string nameOfTable, vector<string> nameOfKey, vector<string> nameOfType, vector<int> isUnique);

建表函数

int createTable(string nameOfTable, vector<string> nameOfKey, vector<string> nameOfType, vector<int> isUnique, string primaryKey);

建表函数

int deleteTableInfo(string nameOfTable);

int deleteTableInfo(string nameOfTable, int d);

在表中删除信息

int createIndex(string nameOfIndex, string tableOfIndex, string keyOfIndex);

建索引语句

int deleteIndex(string nameOfIndex);

删除索引函数

int numberOfRecordAdd(string nameOfTable, int a);

返回表中语句数量

int dropTable(string nameOfTable);

删除一个表

int updateCatalog();

将内存中的信息返回到数据字典文件中

int getFullLength(string nameOfTable);

得到总长度

int getRecordNum(string tableName);

得到总记录数量

int getLength(vector<string> x);

获取该变量的长度

int getLength(string type);

获取该类型的长度

vector<string> nameOfKey(string nameOfTable);

返回表中元素名

vector<string> nameOfType(string nameOfTable);

返回表中类型信息

vector<int> isUnique(string nameOfTable);

返回IsUnique信息

vector<string> nameOfIndex(string nameOfTable);

返回表中索引信息

通过Catalog Manager 模块的处理，已完成了部分语句的功能，对于未完成的语句，也对语句的内部表示形式进行了扩充，通过API接口的功能，即可将信息传递给Record Manager模块和Index Manager模块，为Record Manager模块及Index Manager模块的实现提供了充足的信息。