**Catalog\_Manager模块设计报告**

3120100964 杨瀚宇 计科1204

Catalog Manager负责管理数据库的所有结构信息，包括：

1. 数据库中所有表的定义信息，包括表的名称、占用的块数，记录数，表中字段（属性）数、每条记录的长度，主键名称，及字段（属性）的链表（vector），其中表中每个字段（属性）的定义信息，包括字段名称，索引名称，字段类型，字段长度，是否主键，是否唯一。
2. 数据库中所有索引的定义，包括索引名称，所属表名，索引开始位置，记录长度，字段长度长度，索引偏移，建立在那个字段上等。
3. 数据库内已建的表的数目。

CatalogManager还需要提供访问及操作上述信息的接口，供API模块使用。

为减小模块之间的耦合，CatalogManager模块采用直接存取磁盘文件的形式，不通过BufferManager存储，在每次建表和退出数据库系统的时候更新文件。

CatalogManager类的具体定义如下：

// CatalogManager类

class CatalogManager

{

vector<Table> tableList; // 表信息列表

vector<Index> indexList; // 索引信息列表

int tableNum; // 表的数目

public:

CatalogManager(){};

~CatalogManager(){};

bool API\_Catalog(SQLstatement &sql); // API对cm的调用

Table\* findTable(string tn); // 找到表返回指针

bool createTable(SQLstatement &sql); // 创建表

bool checkInsert(Table \*t, string value); // 检查插入

void pushBack\_tableList(Table &t); // 表加入tableList

void update\_tableNum(bool add); // 更新表数目

bool checkType(Attribute \*a, string v); // 检查类型

bool checkAttribute(Table \*t, vector<Attribute> \*a, bool index); // 检查属性

bool checkCondition(Table \*t, vector<Condition> \*c); // 检查条件

Index\* findIndex(string in); // 找到索引返回指针

bool createIndex(SQLstatement &sql, Table \*t); // 创建索引

bool clearIndex(Table \*t, string in, bool drop); // 清除索引

void pushBack\_indexList(Index &i); // 索引加入indexList

bool dropIndex(Table \*t, Index \*i); // 删除索引

bool dropTable(Table \*t); // 删除表

bool save\_tableInfo(Table \*t, bool add); // 存入表信息

void writeAttribute(string fn, Attribute \*a); // 存入属性

void read\_tableInfo(); // 读出表信息

void update\_tableInfo(); // 更新表信息

bool save\_indexInfo(Index \*i, bool add); // 存入索引信息

void read\_indexInfo(); // 读出索引信息

void update\_indexInfo(); // 更新索引信息

Attribute\* getAttributebya(Table \*t, string an); // 返回属性指针

Attribute\* getAttributebyi(Table \*t, string in); // 返回属性指针

string msg; // 传送消息

};

其具体实现也是按照语句类型进行分别处理：

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SQLstatement类对象中的表名信息，验证是否有重名现象。若无则建立表文件，同时将表类中的各种信息进行初始化，存在表类的对象中，再存入表类对象的链表中。若有重名表，则创建表命令失败，并打印出错信息。

创建索引语句

CREATE INDEX 索引名 ON 表名（列名）；

根据Interpreter处理生成的SQLstatement类对象中的表名和属性名信息，在tableList链表中检查是否存在该表，并检查表中有无该属性，并且验证是否可对该属性建立索引。若可建，则创建索引文件，同时将索引信息创建索引类对象，存入索引类对象的链表中。若不可建索引，则根据不同类型打印出错信息供用户参考。

选择语句

SELECT \* FROM 表名

SELECT 列1，列2，…列n FROM 表名 WHERE 条件1 AND 条件2

根据Interpreter处理生成的SQLstatement类对象中的表名和属性名信息，在tableList链表中检查是否存在该表，并检查表中有无该属性，若有条件，需要检查常量值的数据类型与属性的数据类型是否可进行关系运算。若条件中属性存在索引则先根据已建索引的属性名通过IndexManager来进行记录的查询；若不是索引属性则通过RecordManager来顺序查找记录信息。任何的不符都将打印出相应的出错信息供用户参考。

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SQLstatement类对象中的表名和属性名信息，在tableList链表中检查是否存在该表，并按顺序检查值的类型与属性的数据类型是否一致。任何的不符都将打印出相应的出错信息供用户参考。

删除记录语句

DELETE FORM 表名；

DELETE FORM 表名 WHERE 条件；

根据Interpreter处理生成的SQLstatement类对象中的表名和属性名信息，在tableList链表中检查是否存在该表。若存在，返回给API再通过调用IndexManager和RecordManager来删除索引及记录信息；若表不存在，则须打印出必要的出错信息供用户参考。条件的处理和SELECT WHERE语句基本相似。

删除索引语句

DROP INDEX 索引名；

根据Interpreter处理生成的SQLstatement类对象中的索引名，在索引类对象链表中查找是否有该名索引。若有，则删除该索引在链表中对象，并返回API调用IndexManager来删除索引文件。若无，则打印出相应的出错信息供用户参考。

**删除表语句**

DROP TALBE 表名；

根据Interpreter处理生成的SQLstatement类对象中的表名，在表类对象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，然后删除该表在链表中的对象，最后返回API调用RecordManager来删除该表中的记录，并删除表结构信息文件。若无该表的定义，则打印出相应的出错信息供用户参考。

其主要函数的功能描述如下：

// API对cm的调用

bool API\_Catalog(SQLstatement &sql);

// 找到表返回指针

Table\* findTable(string tn);

// 创建表

bool createTable(SQLstatement &sql);

// 检查插入

bool checkInsert(Table \*t, string value);

// 表加入tableList

void pushBack\_tableList(Table &t);

// 更新表数目

void update\_tableNum(bool add);

// 检查类型

bool checkType(Attribute \*a, string v);

// 检查属性

bool checkAttribute(Table \*t, vector<Attribute> \*a, bool index);

// 检查条件

bool checkCondition(Table \*t, vector<Condition> \*c);

// 找到索引返回指针

Index\* findIndex(string in);

// 创建索引

bool createIndex(SQLstatement &sql, Table \*t);

// 清除索引

bool clearIndex(Table \*t, string in, bool drop);

// 索引加入indexList

void pushBack\_indexList(Index &i);

// 删除索引

bool dropIndex(Table \*t, Index \*i);

// 删除表

bool dropTable(Table \*t);

// 存入表信息

bool save\_tableInfo(Table \*t, bool add);

// 存入属性

void writeAttribute(string fn, Attribute \*a);

// 读出表信息

void read\_tableInfo();

// 更新表信息

void update\_tableInfo();

// 存入索引信息

bool save\_indexInfo(Index \*i, bool add);

// 读出索引信息

void read\_indexInfo();

// 更新索引信息

void update\_indexInfo();

// 返回属性指针

Attribute\* getAttributebya(Table \*t, string an);

// 返回属性指针

Attribute\* getAttributebyi(Table \*t, string in);

通过Catalog Manager 模块的处理，已完成了语句的部分功能，即对于语句的语义进行了检测，然后通过构建表和索引的对象，给API接口提供调用，即可将信息传递给Record Manager模块和Index Manager模块，为他们的实现提供了充足的信息。

以下是表结构信息的类以及索引结构信息的类：

// 定义索引信息

class Index

{

string index\_name; // 索引名

string table\_name; // 表名

int startposition; // 属性在记录中偏移位置

int tuplelength; // 记录长

int columnLength; // 属性长

int IndexOffset; // 索引占块数

public:

Index():IndexOffset(0){}

~Index(){}

};

// 定义表信息

class Table

{

string name; // 表名

int blockNum; // 占用的块数

int recordNum; // 记录条数

int attriNum; // 属性数

int tupleLength; // 每条记录的长度

string primaryKey; // 主键名字

vector<Attribute> attributes; // 指向元数据链表的指针

public:

Table():blockNum(0),recordNum(0),attriNum(0),tupleLength(0){}

Table(string name, string primaryKey) :name(name), primaryKey(primaryKey){}

~Table(){}

};

以下是interpreter和CatalogManager单独运行的测试的结果实例：

成功建表，在相应目录中生成文件表结构信息文件：



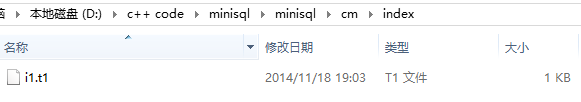


再次建表时提示表已存在：

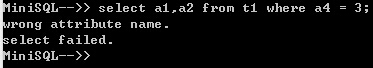
C:\Users\Henry\AppData\Roaming\Tencent\Users\523156933\QQ\WinTemp\RichOle\XX323LU4RR7{KA]B1JGVU6M.jpg

创建索引，产生索引结构信息文件：

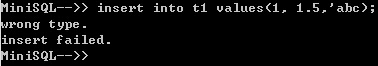
C:\Users\Henry\AppData\Roaming\Tencent\Users\523156933\QQ\WinTemp\RichOle\PWH)`0WFIBPC0}US(9)T7BE.jpg



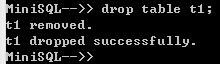
选择语句中有非表中的属性提示错误：



插入语句中值的类型与属性类型不匹配提示错误：



删除表信息，若重复删除则报错：

 C:\Users\Henry\AppData\Roaming\Tencent\Users\523156933\QQ\WinTemp\RichOle\9X]F2HXM%$46D}GUKY(OH$G.jpg