数据库中的文件有三种类型，

第一种是数据文件，后缀是.data，存储表格（tables）

数据文件的格式示例

[int][Char][int][float][…][…][int][Char][int][float][…][…][int][Char][int][float][…][…][int][Char][int][float][…][…]……………………………………………#

即数据是连续存储的，如果block没有存满，那么最后一条数据的结尾是#(Empty)

第二种是索引文件，后缀是.index，存储索引

第三种是目录文件，后缀是.catalog， 存储表格文件和索引文件的定义，目录文件不通过buffer加载

目录文件的格式为

<tableNum>

对每一个表格

<name> <attriNum> <blockNum>

对每一个属性

<名字>，<类型>，<长度>、<是否是主键>，<是否唯一>

对于数据文件和索引文件，每个文件在磁盘中都以block（大小固定为4kb即4096字节）的方式存储，每次加载1个block到buffer（内存）中，buffer由整数个（1000个)block组成。对于目录文件，数据库开始运行的时候加载，存储不是以block的形式，这样可以是系统简化，因为文件定义占据的空间一般很小，所以不会影响效率。

=================== Buffer Manager =======================

bufferblock的定义

class BufferBlock

{

public:

BufferBlock();

~BufferBlock();

char \*contents() const;//返回数据开头的指针

char \*contents(int position) const;//返回数据开头指定偏移量的指针

std::string filename;//此block对应的磁盘文件的名

int blockoffset;//此block在该磁盘文件中的blockoffset

bool used;//此block是否正在被使用

int LRU\_Val;//此block从最近一次加载数据被访问的次数

void clear();//将block中的数据清空

private:

char block[BLOCKSIZE+1];//存储数据

};

Position的定义

struct Position

{

int bufferID;

int position;

};

插入的位置取决于block的ID和最末一条数据在block的偏移字节数

bufferManager类的定义

class BufferManager {

private:

BufferBlock bufferblocks[MAXBLOCKNUMBER];//buffer memory

int GetBufferID(std::string filename, int blockoffset); //查找某文件block在buffer block的哪个位置

void LoadToBufferBlock(std::string filename, int blockoffset, int bufferID);//将文件加载到某个buffer block中去

void UseBufferBlock(int bufferID);

int GetEmptyBufferBlock();//ApplyBufferBlock的子程序

int GetEmptyBufferBlock(std::string filename);

int ApplyBufferBlock(Index& index);//申请buffer块用于加载文件

int ApplyBufferBlock(Table& table);

void StoreToDiskBlock(int bufferID);//写回到磁盘中

void BufferManager::Set(std::string filename); //将指定的block设置为没被使用

InsertPosition GetInsertPosition(Table& table);

public:

friend class RecordManager;

friend class IndexManager;

//Add some B+ tree class friends here

//friend class Leaf;

//friend class Branch;

BufferManager();

virtual ~BufferManager();

};

GetBufferID查找一个磁盘上的block在buffer中的位置，如果不在buffer上就返回NOTFOUND

LoadToBufferBlock将一个磁盘上的block加载到buffer中的一个block中

StoreToDiskBlock将一个buffer中的block写回到磁盘上的一个block中

UseBufferBlock将指定的buffer block设置为使用的状态

GetEmptyBufferBlock寻找一个可用的buffer block

ApplyBufferBlock申请使用一个buffer block

GetInsertPosition找目标表格的末尾的位置

============================================================

===================== Catalog Manager ============================

Attribute的定义

struct Attribute

{

std::string name;

int type; //INT FLOAT CHAR

int length;

bool isPrimeryKey;

bool isUnique;

Attribute();

Attribute(std::string n, int t, int l, bool isP, bool isU);

};

Attribute由名字，类型，长度、是否是主键，是否唯一组成

struct Table

{

std::string filename;

int blockNum;

int attriNum;

int tupleLength;

vector<Attribute> attributes;

Table();

};

表格包含（名字、block的数量、属性的数量、一个元组的长度、属性的序列）

每张表都对应于一个文件

索引的定义

struct Index

{

std::stringfilename;        //文件名 all the datas is store in file index\_name.index

std::string table\_name;        //索引对应的表 the name of the table on which the index is create

int column = 0;                        //根据哪一个属性建立索引排序 on which column the index is created

int columnLength = 0;

int blockNum;                //number of block the datas of the index occupied in the file index\_name.table

};

索引包含（索引文件名，索引对应的表格、索引基于的列、列的一个实例的长度、索引文件的占据block数量）

CatalogManager 的定义

class CatalogManager {

private:

// 存储表的容器

std::vector<Table> tables;

// 数据库中的表的数量，应该与table.size()一致。

int tableNum;

// 存储索引的容器

std::vector<Index> indices;

// 索引的数量

int indexNum;

private:

void InitialTableCatalog();

void InitialIndexCatalog();

// 把这些东西存回去

void StoreTableCatalog();

void StoreIndexCatalog();

public:

CatalogManager();

~CatalogManager();

void createTable(Table& table);

void createIndex(Index index);

void dropTable(Table table);

void dropIndex(Index index);

void dropTable(std::string tablename);

void dropIndex(std::string index\_name);

void update(Table& tableinfor);

void update(Index& index);

bool ExistTable(std::string tablename);

bool ExistIndex(std::string tablename, int column);

bool ExistIndex(std::string indexname);

Table getTableInfo(std::string tablename);

Index getIndexInfo(std::string tablename, int column);

Index getIndexInfo(std::string indexName);

void ShowCatalog();

void ShowTableCatalog();

void ShowIndexCatalog();

int GetColumnNumber(Table& tableinfo, std::string columnname);

int GetColumnAmount(Table& tableinfo);